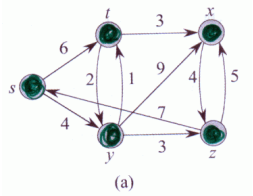
PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69215 ALLAN MORINAGA
19FFU116 - 1 apos 28/05, 50% _____ / _____ /

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

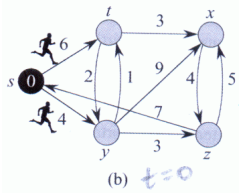


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

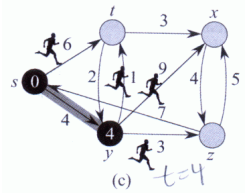
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b) t=0

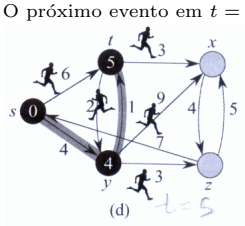
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c) t=4

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

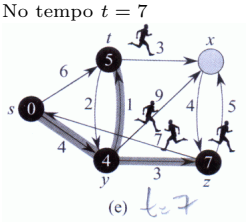


(d) t=5

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

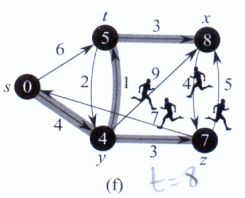
No tempo $t = 7$



(e) t=7

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f) t=8

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distância conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distâncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distância até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 6 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 4 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	30	57	12	38	16	1	27	18
2	5	99	53	19	17	8	50	29	56
3	8	24	99	18	12	40	2	7	22
4	21	16	45	99	2	14	11	11	3
5	26	5	55	3	99	37	4	6	22
6	46	49	13	42	35	99	20	54	28
7	33	31	9	32	1	47	99	41	15
8	39	6	14	4	9	43	36	99	34
9	48	15	52	58	44	25	59	10	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

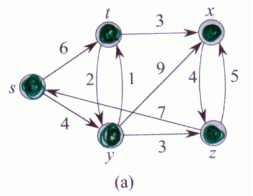
116-69215 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69822 ANDREA A ALVES
19FFU116 - 2 apos 28/05, 50% _____ / _____ /

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

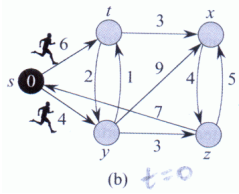


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

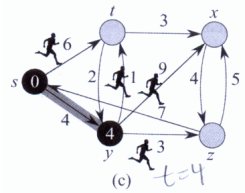
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

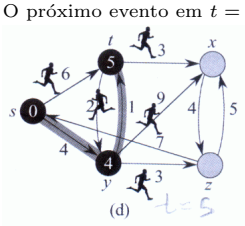
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

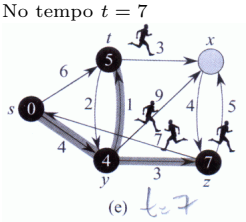


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

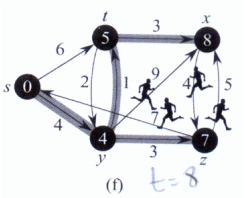
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distância conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 3 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 4 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	33	48	2	13	36	1	41	56
2	10	99	24	13	4	34	8	22	14
3	58	19	99	19	6	5	30	38	4
4	46	20	50	99	28	7	6	53	39
5	21	16	54	26	99	9	23	47	17
6	12	59	32	42	8	99	20	55	11
7	15	10	49	3	43	22	99	29	16
8	31	25	18	57	11	40	7	99	9
9	51	2	18	52	45	5	14	12	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

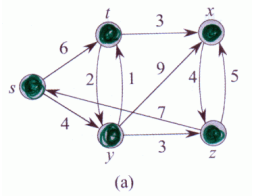
116-69822 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69084 EDSON RODRIGUES VELOSO
19FFU116 - 3 apos 28/05, 50% _____ / _____ /

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

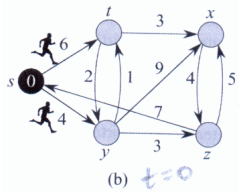


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

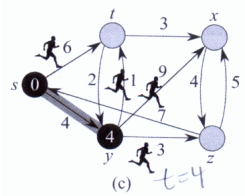
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

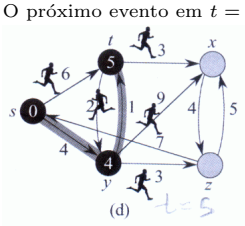
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

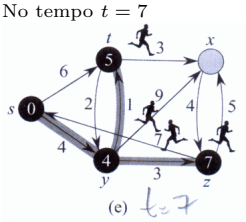


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

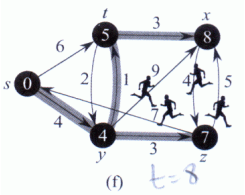
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distância conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 5 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 3 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	9	55	59	4	9	24	53	33
2	26	99	32	6	15	42	20	19	50
3	38	3	99	21	31	43	56	14	16
4	1	13	54	99	41	23	19	3	39
5	10	5	40	1	99	12	10	22	45
6	25	2	4	52	7	99	18	8	2
7	35	11	14	28	21	17	99	58	36
8	22	29	15	12	27	48	49	99	47
9	18	37	34	5	44	20	6	17	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

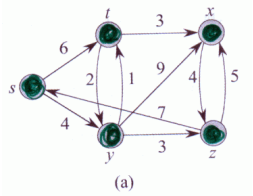
116-69084 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69222 ELAINE GREBOGY (OUV)
19FFU116 - 4 apos 28/05, 50% _____ / _____ /

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

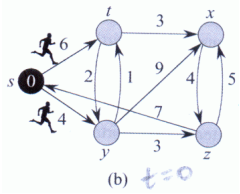


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

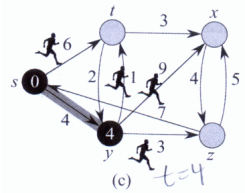
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

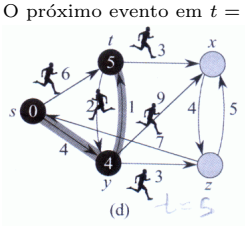
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

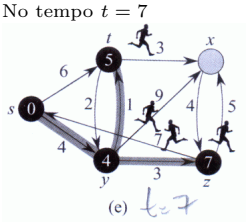


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

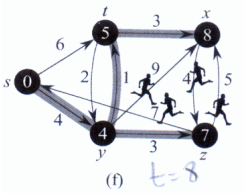
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distância conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 6 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 7 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	50	10	22	7	30	33	12	19
2	22	99	18	35	15	8	31	58	19
3	51	21	99	2	20	10	11	40	43
4	37	2	27	99	9	16	1	38	29
5	46	44	23	52	99	47	25	15	45
6	6	7	42	53	11	99	21	36	32
7	20	18	5	8	5	3	99	17	26
8	4	56	59	6	4	17	57	99	1
9	3	12	48	24	13	41	28	49	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

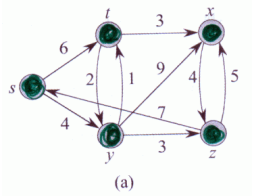
116-69222 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69091 FAGNER FERREIRA
19FFU116 - 5 apos 28/05, 50% _____ / _____ /

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

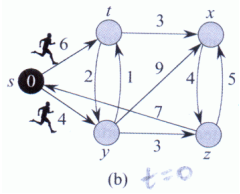


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

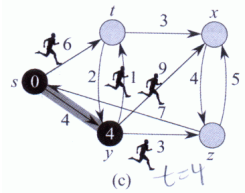
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

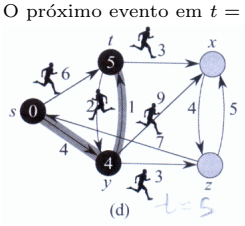
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

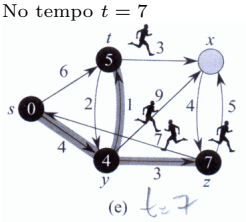


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

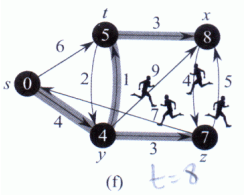
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distância conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distâncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distância até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 2 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 4 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	20	48	33	25	14	22	22	1
2	3	99	9	24	18	8	17	19	12
3	51	17	99	4	15	53	7	32	10
4	4	35	13	99	2	39	13	41	46
5	19	6	58	47	99	34	31	37	36
6	2	50	10	9	18	99	12	16	43
7	30	42	44	56	29	8	99	16	55
8	5	11	38	5	21	52	7	99	45
9	59	28	3	57	23	21	40	6	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

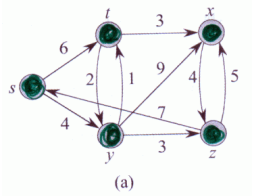
116-69091 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69860 FELIPE GABRIEL
19FFU116 - 6 apos 28/05, 50% _____ / _____ /

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

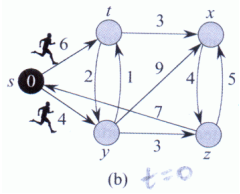


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

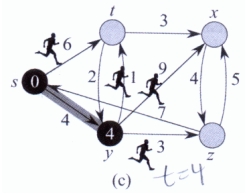
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

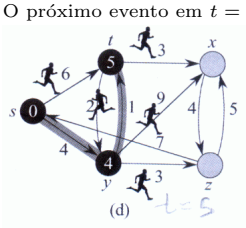
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

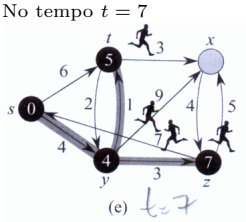


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

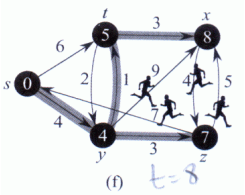
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distãncia conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 9 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 5 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	3	21	22	36	6	23	59	20
2	5	99	55	32	7	11	15	15	49
3	39	20	99	16	1	17	14	43	54
4	12	37	13	99	26	33	4	18	35
5	22	4	5	46	99	45	18	12	8
6	10	25	41	40	47	99	51	2	56
7	38	2	28	30	9	31	99	14	57
8	19	16	7	44	13	6	8	99	50
9	11	21	58	52	53	1	19	29	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

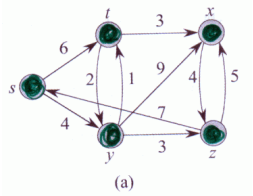
116-69860 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69103 FELLIPE RICHTER NEVES
19FFU116 - 7 apos 28/05, 50% _____ / _____ /

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

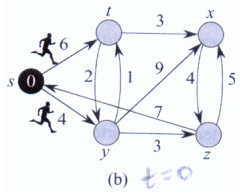


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

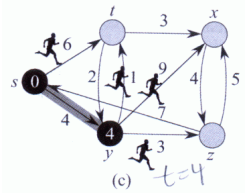
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

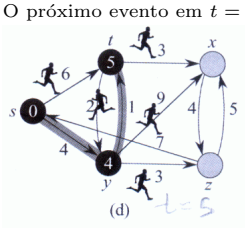
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

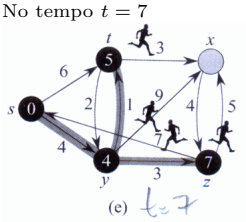


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

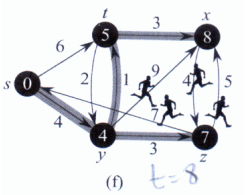
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distãncia conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 1 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 3 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	6	33	13	10	48	21	29	19
2	5	99	47	46	58	53	7	8	2
3	54	21	99	15	31	17	50	4	6
4	5	28	37	99	41	35	45	10	20
5	20	34	14	22	99	19	57	12	26
6	16	17	11	23	30	99	12	27	1
7	1	55	52	7	38	18	99	25	3
8	16	51	9	15	11	39	2	99	9
9	44	42	32	8	36	59	14	4	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

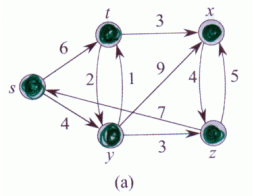
116-69103 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69110 GABRIEL MELEK DOS SANTOS
19FFU116 - 8 apos 28/05, 50% _____ / _____ /

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

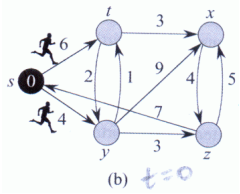


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

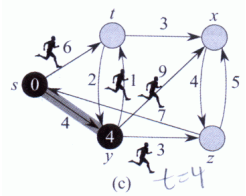
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

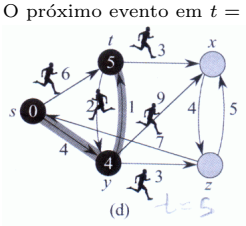
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

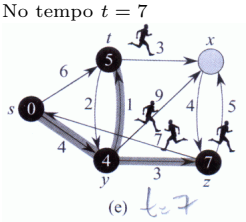


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

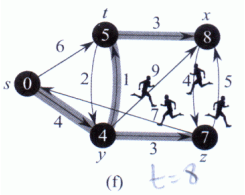
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distância conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 5 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 3 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	48	3	32	18	7	6	12	10
2	7	99	6	1	54	22	9	10	3
3	59	5	99	45	11	21	30	19	31
4	55	14	52	99	46	8	51	17	16
5	33	37	42	15	99	4	58	49	28
6	44	17	43	1	24	99	23	53	50
7	5	36	2	38	12	26	99	19	47
8	20	4	16	15	27	14	41	99	29
9	2	20	57	40	39	35	11	25	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

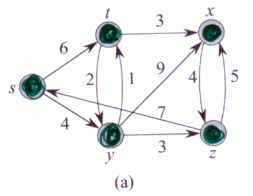
116-69110 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69127 GABRIEL ROBERTO DA ROSA
19FFU116 - 9 apos 28/05, 50% _____ / _____ /

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

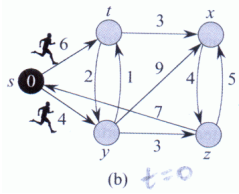


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

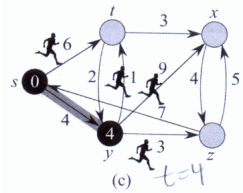
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

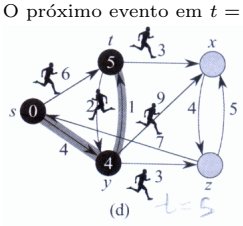
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

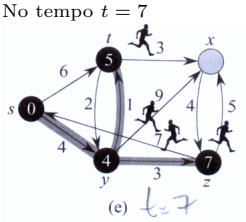


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

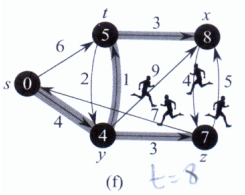
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distância conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distâncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distância até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 2 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 3 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	13	37	6	21	59	4	57	15
2	43	99	47	24	13	7	8	50	58
3	42	40	99	19	46	39	44	11	18
4	5	8	4	99	27	1	51	2	16
5	49	55	52	10	99	9	38	34	20
6	54	19	31	26	32	99	11	48	1
7	45	10	21	53	56	33	99	18	3
8	28	14	17	22	30	9	6	99	20
9	5	12	22	16	7	25	36	12	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

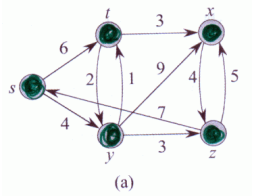
116-69127 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69134 GABRIEL SOUZA BRAGA
19FFU116 - 10 apos 28/05, 50% _____ / / _____

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

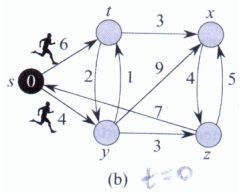


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

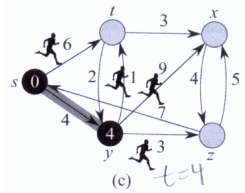
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

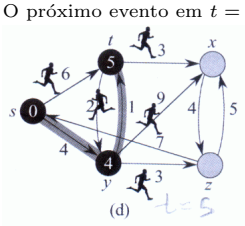
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

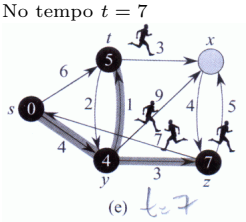


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

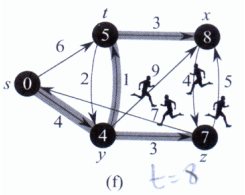
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distância conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 3 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 1 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	31	17	20	37	36	52	57	8
2	29	99	53	6	21	14	12	19	10
3	18	26	99	22	16	6	35	34	3
4	54	11	41	99	4	28	59	14	55
5	23	15	16	2	99	45	20	15	8
6	11	17	5	9	18	99	3	58	43
7	7	56	33	46	27	2	99	49	5
8	9	42	50	32	47	1	39	99	10
9	38	24	7	25	19	1	13	22	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

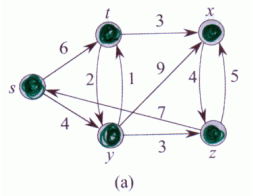
116-69134 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69141 GUILHERME SAITO BANDEIRA
19FFU116 - 11 apos 28/05, 50% _____ / / _____

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

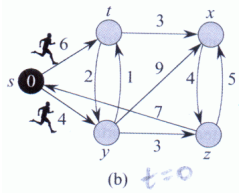


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

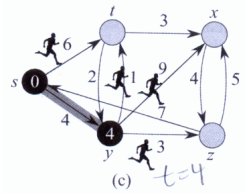
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

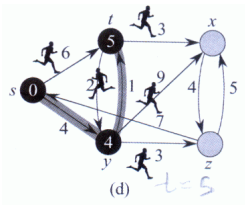
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

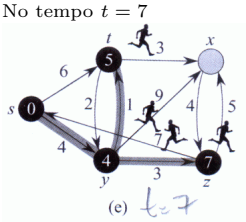


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

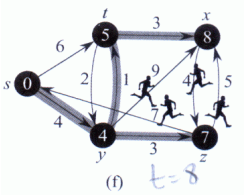
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distância conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 9 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 6 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	16	35	7	40	53	44	19	6
2	30	99	17	45	8	10	57	3	48
3	50	19	99	7	34	13	4	6	52
4	16	24	2	99	56	12	17	22	39
5	18	9	31	22	99	28	12	14	49
6	5	55	21	3	38	99	8	54	15
7	20	26	36	42	58	1	99	9	2
8	11	4	18	51	21	20	15	99	5
9	13	47	43	14	10	27	32	33	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

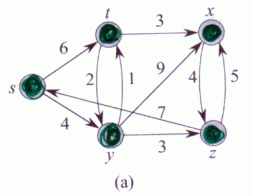
116-69141 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69158 HARON AURELIANO TRAVASSOS
19FFU116 - 12 apos 28/05, 50% _____ / / _____

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

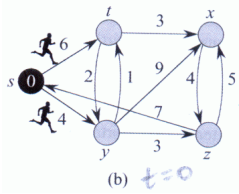


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

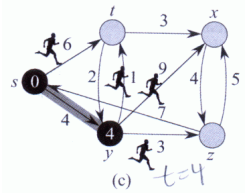
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

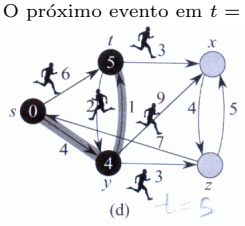
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

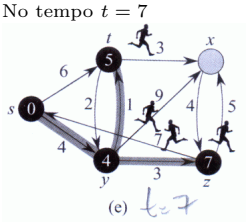


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

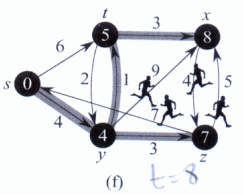
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distância conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 6 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 8 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	13	38	4	11	8	32	24	37
2	34	99	29	53	47	44	7	1	35
3	18	23	99	36	48	25	17	56	22
4	5	58	10	99	14	45	27	21	28
5	3	1	26	16	99	16	18	5	19
6	15	52	54	30	19	99	4	33	20
7	17	10	15	7	2	40	99	55	41
8	22	59	11	12	9	57	39	99	50
9	46	8	43	3	31	6	42	6	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

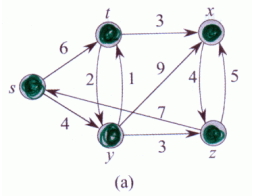
116-69158 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69239 ICLEIA SANTOS (OUV)
19FFU116 - 13 apos 28/05, 50% _____ / / _____

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

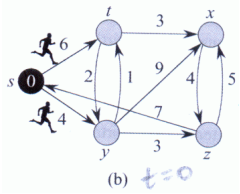


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

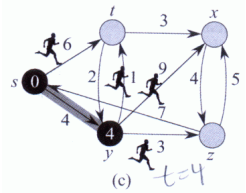
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

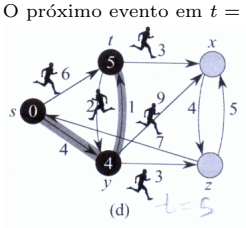
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

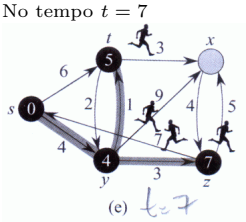


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

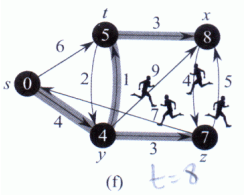
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distância conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 4 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 5 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	23	38	48	49	24	2	16	10
2	3	99	36	33	2	37	5	45	18
3	51	17	99	59	52	35	20	15	53
4	1	20	1	99	26	14	55	44	50
5	14	29	19	8	99	7	6	25	22
6	17	5	21	56	27	99	13	32	57
7	28	43	40	9	47	3	99	9	13
8	58	11	12	21	18	34	39	99	8
9	54	6	42	16	10	31	4	22	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

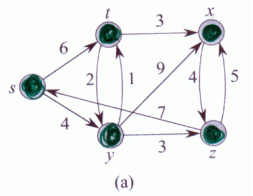
116-69239 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69877 ISABELA CASTILHO STIVAL
19FFU116 - 14 apos 28/05, 50% _____ / / _____

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

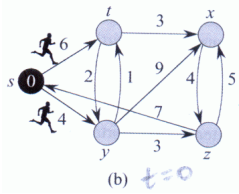


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

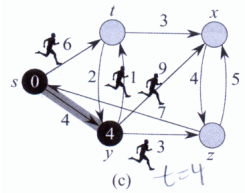
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

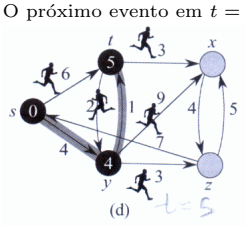
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

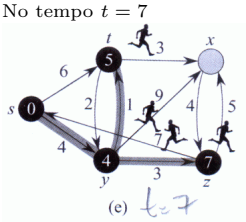


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

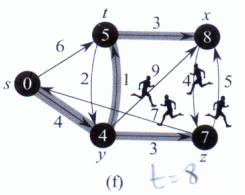
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distância conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 8 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 5 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	23	7	6	13	24	35	9	10
2	55	99	21	25	20	41	33	18	17
3	10	22	99	7	6	53	1	57	44
4	4	37	5	99	2	59	5	1	16
5	22	20	36	17	99	8	40	4	2
6	3	14	9	21	52	99	43	48	15
7	14	38	49	27	30	56	99	51	15
8	12	42	11	29	39	3	47	99	34
9	32	13	12	54	45	58	26	11	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

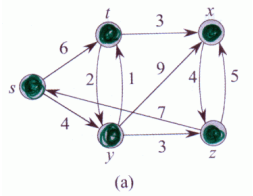
116-69877 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69246 MAURICIO KOBAY DO AMARAL
19FFU116 - 15 apos 28/05, 50% _____ / / _____

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

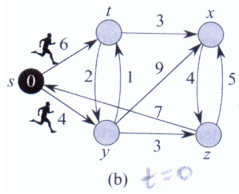


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

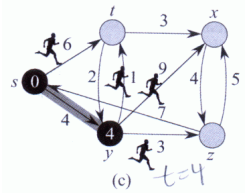
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

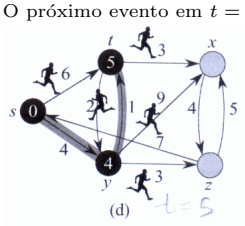
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

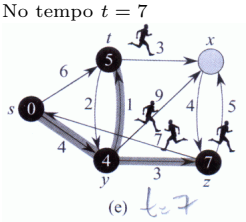


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

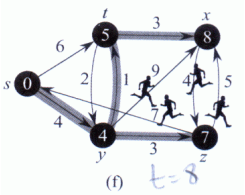
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distãncia conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 8 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 5 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	4	18	35	2	45	10	3	6
2	33	99	11	20	16	22	14	1	28
3	10	5	99	26	17	43	21	56	12
4	18	47	3	99	9	12	22	15	15
5	40	32	16	34	99	25	7	8	13
6	17	54	2	1	6	99	41	36	8
7	19	29	23	5	31	13	99	51	24
8	58	42	44	57	50	46	53	99	14
9	39	49	38	48	30	59	11	37	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

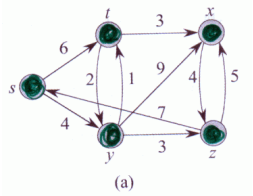
116-69246 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69253 PEDRO HENNIG
19FFU116 - 16 apos 28/05, 50% _____ / / _____

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

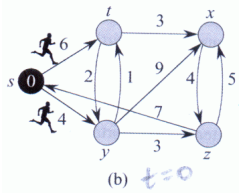


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

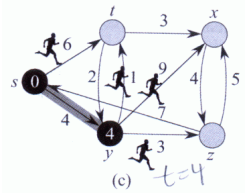
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

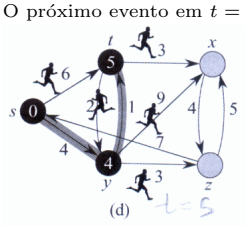
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

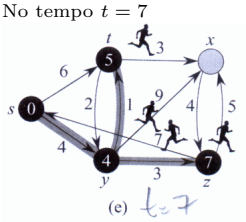


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

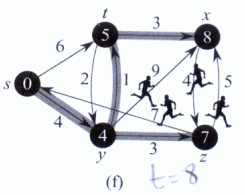
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distância conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 4 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 7 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	19	10	41	11	14	7	24	4
2	22	99	17	53	1	58	18	1	49
3	16	55	99	17	48	8	23	9	10
4	30	22	12	99	57	3	47	54	5
5	59	52	29	21	99	33	34	56	43
6	18	8	26	6	21	99	15	20	9
7	31	44	15	50	46	28	99	42	39
8	3	5	2	20	45	40	4	99	13
9	36	51	2	19	6	38	14	27	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

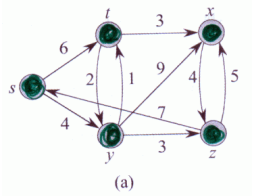
116-69253 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69165 RAFAEL DO NASCIMENTO BARBOSA
19FFU116 - 17 apos 28/05, 50% _____ / / _____

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

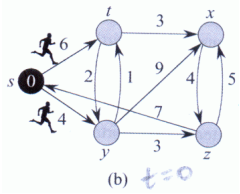


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

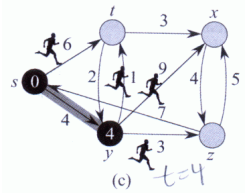
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b)

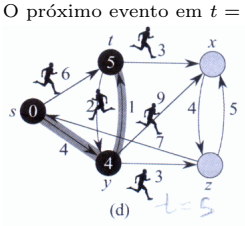
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c)

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

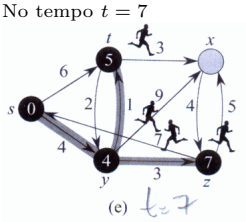


(d)

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

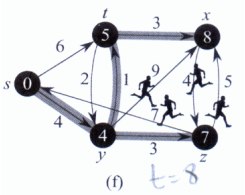
No tempo $t = 7$



(e)

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f)

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distância conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 3 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 7 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	43	45	54	12	36	13	19	25
2	20	99	15	50	16	46	33	32	56
3	42	8	99	5	19	18	57	40	4
4	23	3	13	99	35	12	21	2	38
5	1	31	30	17	99	41	3	4	24
6	11	9	1	22	18	99	14	34	7
7	10	51	59	15	37	47	99	16	29
8	52	2	55	48	7	11	28	99	22
9	10	8	6	26	17	39	53	9	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

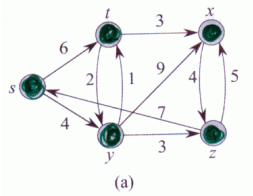
116-69165 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69172 RODRIGO PORTANTIOLO WAGNER 19FFU116 - 18 apos 28/05, 50% _____ / / _____

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

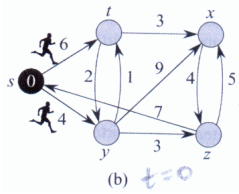


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

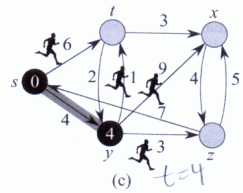
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b) t=0

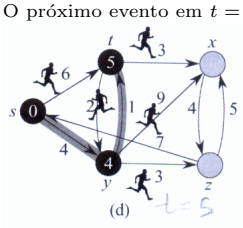
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c) t=4

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

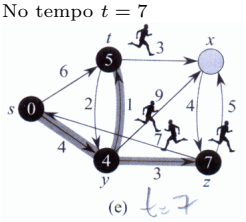


(d) t=5

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

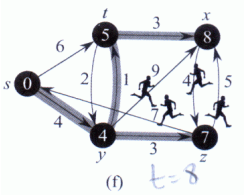
No tempo $t = 7$



(e) t=7

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f) t=8

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distância conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 4 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 5 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	40	8	34	46	12	58	18	27
2	30	99	36	45	3	43	49	5	16
3	17	35	99	20	37	33	4	42	17
4	47	24	15	99	59	22	2	14	51
5	39	9	53	1	99	11	7	15	9
6	18	1	52	14	19	99	13	44	32
7	26	50	29	55	13	6	99	12	48
8	3	22	38	7	20	28	4	99	57
9	31	23	11	21	41	2	54	8	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

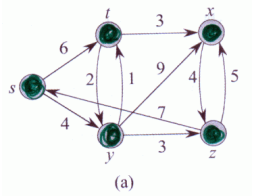
116-69172 - 28/05

PUC/Pr Pontifícia Universidade Católica do Paraná
Curso de Bacharelado em Informática 04/05/2019 - 22:24:50.6
Cadeira de Estruturas de Dados e Arquivos Prof Dr P Kantek
Caminho mínimo: algoritmo de Dijkstra
VIVO622a V: 3.06
69189 WILSON DE MELO
19FFU116 - 19 apos 28/05, 50% _____ / / _____

Caminho mínimo: algoritmo de Dijkstra

(dica: O autor é um holandes e seu nome é lido como "Dicstêr").

Para entender este problema é util imaginar o grafo espalhado em uma área qualquer e pensar em corredores humanos fazendo trajetos sobre o grafo. Embora o algoritmo de Dijkstra funcione de maneira ligeiramente diferente é útil fazer esta analogia. Suponha um grafo como este:

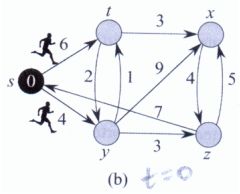


(a)

Por evidente, pode-se construir uma matriz de adjacência como segue

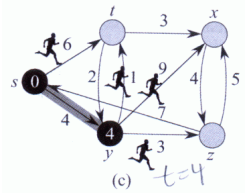
	s	t	x	y	z
s	-	6	-	4	-
t	-	-	3	2	-
x	-	-	-	-	4
y	-	1	9	-	3
z	7	-	5	-	-

Escolhido o nodo origem (neste caso o nodo=s) enviam-se os corredores deste nodo a todos os nodos adjacentes. Na simulação, a cada instante que um corredor chega a um vértice, novos corredores saem do vértice sempre se dirigindo a todos os vértices adjacentes. Supondo que os pesos das arestas indicam a quantidade de tempo em minutos que os corredores demoram. O vértice s em negor indica que sabemos que a demora para ir de s a s vale 0. Assim, no instante $t = 0$ tem-se



(b) t=0

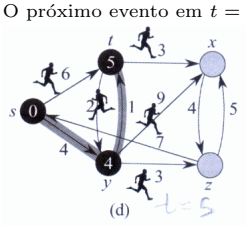
Quatro minutos mais tarde (no tempo $t = 4$) chega o corredor que vai ao vértice y o que é mostrado em



(c) t=4

Como esse corredor é o primeiro a chegar a y sabe-se que $dist[y] = 4$ e portanto o vértice y fica negro na figura. A aresta sombreada (s, y) indica que o primeiro corredor a chegar a y veio de s e portanto $precede[y] = s$. No tempo 4, o corredor que vem de s ainda está em trânsito e nessa hora corredores saem em direção a t, x e z.

O próximo evento em $t = 5$

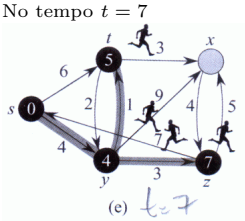


(d) t=5

Aqui o corredor que vem do vértice y chega ao vértice t. O que vem de s a t ainda está no caminho. Como o primeiro a chegar a t veio de y no tempo 5, iguala-se $dist[t] = 5$ e $precede[t] = y$ isto indicado pela aresta sombreada (y, t). Os corredores saem de t em direção a x e y.

O corredor que vem de s finalmente chega a t no tempo 6, mas o corredor que veio de y já chegou lá antes e portanto o esforço do corredor que foi de s a t foi em vão e ele é desprezado.

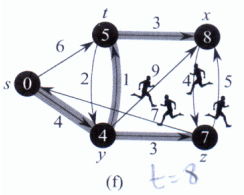
No tempo $t = 7$



(e) t=7

dois corredores chegam ao seu destino. O que fez (t, y) chega, mas o que veio de (s, y) chegou no tempo 4 e portanto a simulação esquece de (t, y). Mas, o corredor que vem de y chega a z e é o primeiro. Fazemos $dist[z] = 7$ e $precede[z] = y$ e os corredores saem de z a caminho de s e x.

O próximo evento ocorre no tempo 8, como em



(f) t=8

quando o corredor que vem de t chega a x. Fazemos $dist[x] = 8$ e $precede[x] = t$ e um corredor sai de x em direção a z. Neste ponto, um corredor já chegou a todos os vértices e a simulação pode parar.

O algoritmo de Dijsktra trata todas as arestas do mesmo jeito. Assim ele processa todos os vértices adjacentes junto, sem nenhuma ordem em particular. Quando ele processa as arestas que saem de s ele declara $dist[y] = 4$, $dist[t] = 6$ e $precede[y] = precede[t] = s$ **até aqui**. Quando o algoritmo considerar mais adiante a aresta(y, t) ele diminuirá o peso do caminho mínimo até o vértice t que encontrou até então, de modo que $dist[t]$ vai de 6 para 5 e $precede[t]$ troca de s para y. Examine o algoritmo aqui descrito em pseudo-código, extraído do livro do Tennenbaum, (Estruturas de dados em C) pág 679-681.

```
algoritmo DIJSKTRA
{dist é a distãncia conhecida desde INI até i (onde i ≠ INI)}
{perm é um vetor de distãncias mínimas permanentes }
{n é o número de vértices do grafo}
{corrente é o nodo que está sendo analisado}
{INI é o nodo inicial e FIN é o final}
{menordist é a menor distãncia até aqui}
para i = 1 ate n
    perm [i] ← 0 {0 significa que perm[i] ainda não é conhecida}
    dist[i] ← 9999
fim{para}
corrente ← INI
perm [INI] ← 1 {já é conhecida}
dist[INI] ← 0
enquanto (corrente ≠ FIN)
    menordist ← 99
    dc ← dist[corrente]
    para i = 1 até n
        se perm[i] = 0
            novadist ← dc + ADJ[corrente,i]
            se novadist < dist[i]
                dist[i] ← novadist
                precede [i] ← corrente
            fim{se}
            se dist[i] < menordist
                menordist ← dist[i]
                k ← i
            fim{se}
        fim{se}
    fim{para}
    corrente ← k
    perm[corrente] ← 1
fim{enquanto}
escreva (dist[FIN])
G ← FIN
enquanto INI ≠ G
    escreva (G)
    G ← precede[G]
fim{enquanto}
escreva(INI)
fim{algoritmo}
```

Acompanhe no quadro negro a implementação deste algoritmo em Python. Se preferir peça para o professor a implementação em C, Maple ou ainda em APL que aliás é a implementação que vai corrigir sua resposta. Por óbvio (e aqui está a maravilha da programação) todas têm que dar a mesma resposta.

Para você fazer

Aqui está um grafo dirigido, sem laços, dado através de sua matriz de custos. Dado o vértice inicial 7 calcule através do algoritmo de Dijkstra, a distância mínima até o nodo 9 . Escreva qual o caminho percorrido

	1	2	3	4	5	6	7	8	9
1	99	21	24	17	25	19	29	15	58
2	5	99	11	30	49	7	19	36	46
3	51	15	99	28	22	18	9	5	21
4	52	53	35	99	12	2	3	17	14
5	42	16	20	2	99	10	12	57	50
6	11	37	27	4	14	99	8	6	56
7	31	20	7	4	33	59	99	40	18
8	1	41	47	9	16	6	48	99	1
9	3	39	38	34	13	54	43	8	99

Respostas

Valor do caminho mínimo: _____.

Roteiro do caminho mínimo:

Observação: nem todas as casas abaixo serão utilizadas.

1	2	3	4	5	6	7	8	9

116-69189 - 28/05