# PRG17

O problema fundamental da Ciência da Computação (CC): dado um vetor e uma chave, encontrar a localização (índice) da chave no vetor ou a informação NÃO EXISTE.

Vetor aqui deve ser entendido em sentido amplo: um arquivo, um banco de dados, um relatório, um site na web, ...

Busca simples:

```
int x[1000]={devidamente inicializado};
int busca(int x[1000], int key){
   int i;
   i=0;
   while (i<1000)&&(x[i]!=key){
       i++;
   }
   if (i==1000){return -1}
   else {return i}
}
```

Note que como vetores precisam ser definidos com seu tamanho final (eles não podem crescer), é costume definir uma área MAIOR e sinalizar onde ela realmente acaba (em termos lógicos). Pode-se fazer isto de muitas maneiras, aqui vai-se colocar um contador na posição 0 do vetor. Assim:

```
int x[1001]; // espaço para 1000 ocorrências
x[0]=contador de quantas realmente estão em uso.
x[1]...x[x[0]] // as ocorrências reais
x[x[0]+1]...x[1000] // espaço para crescimento.
```

Neste tipo de vetor, a busca acima ficaria:

```
int x[1000]={devidamente inicializado};
int busca(int x[1001], int key){
   int i;
   i=0;
   while (i<x[0])&&(x[i+1]!=key){
       i++;
   }
   if (i==x[0]){return -1}
   else {return i}
}
```

Uma maneira de acelerar a busca é com uma sentinela. Trata-se de inserir no final do vetor o elemento que vai ser buscado. Depois disso, a busca sempre será bem sucedida. (Diminui a quantidade de testes em 50%). Achado o elemento buscado, se ele for a sentinela, isso sinaliza que o elemento não estava presente no vetor. Se não for a sentinela, é porque o elemento existe no vetor.

Acompanhe: Seja X[15]=10,1,6,3,8,12,9,5,21,17,10,0,0,0,0,0,0 O primeiro sinaliza o tamanho Quero pesquisar a chave 77. Na busca simples, eu preciso fazer 20 testes: i<10 (10 vezes) e 77==1, 6, 3, ...

Na busca com sentinela, se eu quero procurar o 77, eu insiro ele na posição 10 (x[0], mas sem incrementar x[0], ou seja o 77 sofreu uma inserção FAKE. O vetor agora fica X[15]=10,1,6,3,8,12,9,5,21,17,10,77,0,0,0,0,0 A busca por 77, só precisa testar se 77=1,6,3,...,77. Como se pode ver a busca sempre vai ser bem sucedida. Quando ela terminar, testa-se se o 77 achado está fora do vetor (o que sinaliza insucesso) ou dentro (sucesso).

Uma melhoria substancial é obtida na busca binária: imagine pesquisar em um dicionário de maneira sequencial (como fizemos até agora). A busca binária divide o universo em 2 e daí escolhe uma das duas metades e assim recursivamente até o final. A contrapartida é a exigência de ordenação (ordenador !) previa. Veja o algoritmo:

```
int x[20]={16,3,5,8,17,22,29,36,44,47,48,49,50, \
           60,70,81,99};
int buscab(int x[], int key){
   int ini,fim,met,lixo;
   ini=0; fim=x[0];
   while (1==1){
      met=(ini+fim)/2;
      if (x[met]==key) {return met;}
      if (ini>fim){return -1;}
      if (key < x[met]){
         fim=met-1;
      }
```

```
      else {
         ini=met+1;
      }
   }
}
int main(){
   cout<<buscab(x,99);
}
```

Só que para poder usar a busca binária o vetor tem que ser ordenado...

```
int x[20]={16,55,62,2,7,6,44,21,99,78,61,63,59,\
          35,39,5,91,0,0,0};
int ordena(int v[]){
   int i,j,maxi,imax;
   j=v[0];
   while (j>1){
      i=1;
      maxi=-999999;
      while (i<=j){
         if (v[i]>maxi){maxi=v[i];imax=i;}
         i++;
      }
      swap(v[j],v[imax]);
      j--;
   }
}
int main(){
  int i;
  ordena(x);
  for (i=0;i<20;i++){cout<<x[i]<<" ";} }
```

A inclusão no final (vetor desordenado é facil:

```
int x[20]={16,3,5,8,17,22,29,36,44,47,48,49,50, \
           60,70,81,99};
int inclui(int v[], int key){
   if (v[0]>19){return -1;}
   v[0]++;
   v[v[0]]=key;
   return v[0];
}
int main(){
   int i;
   inclui(x,201);
   for (i=0;i<20;i++){
    cout<<x[i]<<" ";
   }  }
```

A inclusão, mantendo o vetor em ordem é mais complicada, pois há que dar um "chega pra lá" nos elementos maiores do que a chave entrante...

```
int x[20]={16,3,5,8,17,22,29,36,44,47,48,49,50, \
           60,70,81,99,0,0,0};
int incluiord(int v[], int key){
   int i,j;
   if (v[0]>=19){return -1;}
   i=1;
   while ((v[i]<key)&&(i<=v[0])){i++;}
   j=v[0];
   while (j>=i){
        v[j+1]=v[j];
        j--;
    }
   v[i]=key;
```

```
   v[0]++;
   return i;
}
int main(){
   int i;
   incluiord(x,1000);
   for (i=0;i<20;i++){
    cout<<x[i]<<" ";
   } }
```

A exclusão pode ser deixando lixo (fácil, basta colocar um número identificável como vazio, por exemplo -0, ou -99999, ou 0 ou qualquer outra coisa inconfundível), mas daí a função de busca precisa levar em consideração este valor.

A função mais correta é a que traz os elementos uma posição à esquerda

```
int x[20]={16,3,5,8,17,22,29,36,44,47,48,49,50, \
           60,70,81,99,0,0,0};
int exclui(int v[], int key){
   int i,j;
   for (i=1;i<v[0];i++){
      if (v[i]==key){break;}
   }
   if (i==v[0]+1){return -1;}
   j=i;
   while (j<v[0]){
        v[j]=v[j+1];
        j++;
    }
    v[0]--;
}
int main(){
    int i;
   exclui(x,3);
   for (i=0;i<20;i++){
      cout<<x[i]<<" ";
   } }
```