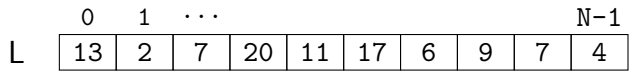
Hierarquia de Controle
Professor Wladimir

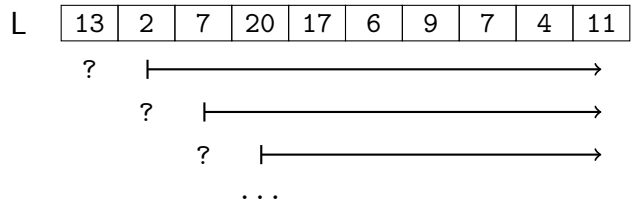
Elemento repetido

Agora imagine que nós temos uma lista



E imagine que a tarefa é Verificar se existe algum elemento repetido

Estratégia: Para cada elemento da lista, verificamos se ele aparece outra vez mais adiante. Se sim, encontramos um elemento repetido e podemos parar nossa busca.




```c
#include <stdio.h>
#include <stdlib.h>
#define  N  10
int  k = 18;
int L[N] = { 13,2,7,20,17,6,9,7,4,11 };
int main()
{

    int  i, j, resp = 0;

    for(i = 0; i < N; i++){
        for(j = i+1; j < N; j++){
            if(L[j] == L[i]){
                resp = 1;
                break;
            }
        }

        if(resp == 1) break;
    }

}
```

Elemento mais frequente

Imagine que nós temos uma lista que contŕm números inteiros

2	2	3	4	2	3	4	5	7	8

Imagine que a nossa tarefa seja encontrar o elemento que aparece mais vezes.

Estratégia: Utilizaremos um vetor auxiliar C tal que C[i] armazena o número de vezes que L[i] aparece no vetor. Em seguida, percorremos o vetor auxiliar para descobrir o elemento mais frequente

A coisa fica assim

```c
#include <stdio.h>
#include <stdlib.h>
#define  N  10
int  k = 18;
int L[N] = { 2,2,3,4,2,3,4,5,7,8 };
int main()
{
    int  i, j;
    int C[N];
    int M, posM;
    for(i = 0; i < N; i++){
        C[i] = 0;
        for(j = i+1; j < N; j++){
            if(L[j] == L[i]){
                C[i]++;

            }
        }
    }
    M = C[0];
    posM = 0;
    for(i = 1; i < N; i++){
        if( C[i] > M){
            M = C[i];
            posM = i;
        }
    }
    printf("O elemento que mais aparece é %d ", L[posM]);

}
```

Note que podemos reduzir o número de instruções executadas contando apenas as ocorrências adiante do vetor para cada elemento e não precisamos realizar em duas etapas e a solução fica assim:

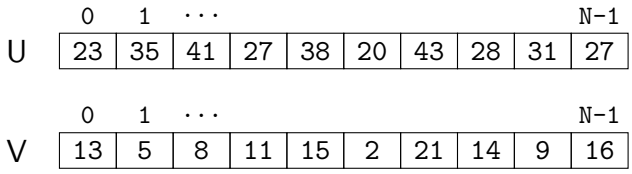
```c
#include <stdio.h>
#include <stdlib.h>
#define  N  10
int  k = 18;
int L[N] = { 2,2,3,4,2,3,4,5,7,8 };
int main()
{
    int  i, j;
    int C[N];
    int M, posM;
    M = -1; posM = -1;
    for(i = 0; i < N; i++){
        C[i] = 0;
        for(j = i+1; j < N; j++){
            if(L[j] == L[i]){
                C[i]++;

            }
        }
        if(C[i] > M){
            M = C[i];
            posM = i;
        }
    }

}
```

Além disso, não precisamos do vetor auxiliar e solução fica assim:

```c
#include <stdio.h>
#include <stdlib.h>
#define  N  10
int  k = 18;
int L[N] = { 2,2,3,4,2,3,4,5,7,8 };
int main()
{
    int  i, j;
    int cont;
    int M, posM;
    M = -1; posM = -1;
    for(i = 0; i < N; i++){
        cont = 0;
        for(j = i+1; j < N; j++){
            if(L[j] == L[i]){
                cont++;
            }
        }
        if(cont > M){
            M = cont;
            posM = i;
        }
    }

}
```

Todos em U são maiores que todos em V

Imagine que nós temos duas listas U e V, ambas com tamanho N. Por exemplo,



A tarefa consiste em verificar se todos os elementos da lista U
são maiores do que todos os elementos da lista V
No exemplo acima, a resposta é: `Não`.
Certo.
Bom, a seguinte lógica resolve o problema

```
    Para cada elemento U[i]
       Verificar se U[i] é maior do que todos os elementos na lista V
```

```c
#define  N  10

int  U[N] = { 23, 35, 41, 27, 38, 20, 42, 28, 31, 27 };

int  V[N] = { 13, 5, 8, 11, 15, 2, 21, 14, 8, 16 };

int main()
{
 int  i, j, resp = 1;          // dois dedos e variável auxiliar

 for ( i=0; i<N; i++ )
  {
    for ( j=0; j<N; j++ )
     {
        if ( V[j] >= U[i] )
         {
            resp = 0;   break;
         }
     }
    if ( resp == 0 )    break;
  }

 if ( resp == 1 )     printf ("A resposta é Sim!");
 else                 printf ("A resposta é Não ...");
}
```

Mas, existe uma maneira bem mais esperta de resolver o problema.
Quer dizer, basta

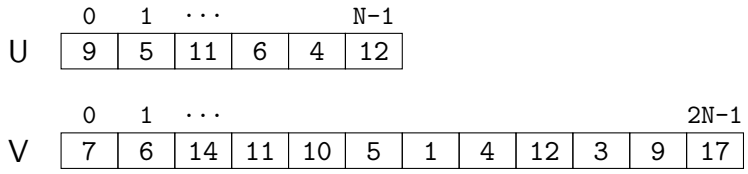
1. Encontrar o menor elemento x da lista `U`
2. Encontrar o maior elemento y da lista `V`
3. Verificar se `x` $>$ `y`

A coisa fica assim

```
( . . . )

 int main()
  {
     int  i;              // apenas 1 dedo
     int  x, y;

     // 1a ETAPA: encontrar o menor elemento de U

     x  =  U[0];
     for ( i=1; i<N; i++ )
      {
         if ( U[i] < x )   x  =  U[i];
      }
     // 2a ETAPA: encontrar o maior elemento de V

     y  =  V[0];
     for ( i=1; i<N; i++ )
      {
         if ( V[i] > y )   y  =  V[i];
      }

     // 3a ETAPA: comparação e resposta

     if ( x > y )      printf ("A resposta é Sim!");
     else              printf ("A resposta é Não ...");
  }

```

Lista dentro de lista

Agora imagine que nós temos duas listas U e V, onde `V` tem o dobro do tamanho de U.
Por exemplo,



A tarefa consiste em verificar se todos os elementos da lista U aparecem na lista V

No exemplo acima, a resposta é: `Sim`.
Certo.
Mas como é que a gente faz isso?
Bom, isso é mais uma tarefa repetitiva

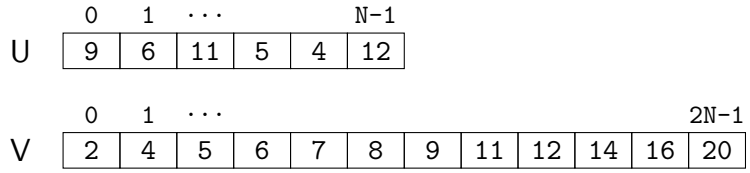
```
    Para cada elemento U[i]
       Verificar se U[i] aparece na lista V
```

E a coisa fica assim

```c
#include <stdio.h>
#include <stdlib.h>

#define  N   6
#define  M  12

int  U[N] = { 9, 5, 11, 6, 4, 12 };

int  V[M] = { 7, 6, 14, 11, 10, 5, 1, 4, 12, 3, 9, 17 };

int main()
{
 int  i, j, achei;          // dois dedos e um indicador


 for ( i=0; i<N; i++ )      // Para cada elemento U[i]
  {
    achei  =  0;

    for ( j=0; j<M; j++ )
     {
        if ( V[j] == U[i] )
         {
            achei = 1;    break;
         }
     }
    if ( achei == 0 ) {
        resp = 0;
        break;
    }
  }

  if(achei == 1){
    printf("Todos os elementos de U aparecem em V.");
  }else{
    printf("Existe pelo menos um elemento de U que não está em V.");
  }
}
```

Lista dentro de lista

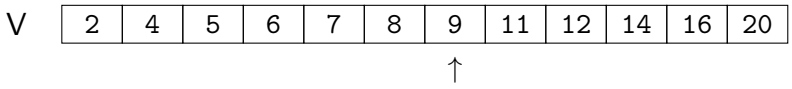
Agora imagine que a lista V está ordenada — (*mas a lista* U *não está*)
Por exemplo,



A tarefa continua sendo verificar se todos os elementos da lista U aparecem na lista V

E a ideia é tirar vantagem do fato de que a lista V está ordenada.

Por exemplo, imagine que a gente já encontrou o elemento 9

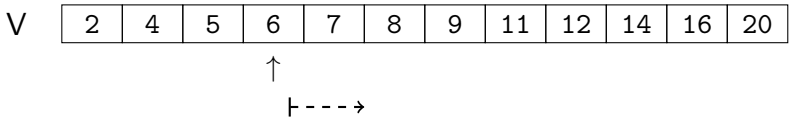


E agora, é preciso encontrar o 6.

A esperteza consiste em começar a procurar a partir daonde a gente parou
— (*ao invés de voltar para o início da lista*)

E como o 6 é menor que o 9, a gente vai andar para trás.

Daí, uma vez que o 6 já foi encontrado, é preciso procurar o 11.
E como o 11 é maior que o 6, a gente vai andar para frente



A coisa fica assim

```c
#define  N   6
#define  M  12

int  U[N] = { 9, 5, 11, 6, 4, 12 };
int  V[M] = { 8, 9, 14, 7, 5, 11, 20, 6, 4, 16, 2, 12 };

int main()
{
 int  i, j, achei;            // dois dedos e variável auxiliar

 j  =  0;

 for ( i=0; i<N; i++ )        // Para cada elemento U[i]
  {
     achei = 0;

     if ( V[j] < U[i] )       // procura para frente
      {
          while ( j < N  &&  V[j] <= U[j] )
           {
              if ( V[j] == U[i] )
               {
                   achei = 1;   break;
               }
              else
                   j++;
      }    }
     else                     // procura para trás
      {
          while ( j >= 0  &&  V[j] >= U[j] )
           {
              if ( V[j] == U[i] )
               {
                   achei = 1;   break;
               }
              else
                   j--;
      }    }

     if ( achei == 0 )    break;
  }

 if ( achei == 1 )     printf ("A resposta é Sim!");
 else                  printf ("A resposta é Não ...");
}
```