

## Estructura de Datos y Algoritmos

### Grados de la Facultad de Informática (UCM)

Examen FINAL SEPTIEMBRE 2018 (Modificado. Parte primer trimestre).

1. (2 puntos) Se dice que dos vectores de enteros del mismo tamaño y cada uno con todos sus elementos distintos entre sí son *desplazamiento circular* el uno del otro cuando la secuencia de elementos de ambos son iguales si no se tiene en cuenta la posición de comienzo de las mismas.

Por ejemplo, [7, 9, 1, 3, 5] es desplazamiento circular de [1, 3, 5, 7, 9] y de [7, 9, 1, 3, 5], pero no lo es de [3, 5, 1, 7, 9].

Se ha desarrollado el siguiente algoritmo para comprobar si dos vectores son desplazamiento circular el uno del otro:

```
bool desplazamiento(const int V[], const int W[], const int N) {
    int i, j;
    i = 0;
    while (i < N && !(V[0] == W[i]))
        i = i + 1;
    j = 0;
    while (j < N && (V[j] == W[(i + j) % N]))
        j = j + 1;
    return (j == N);
}
```

Se pide:

1. Especificar la función *desplazamiento*.
  2. Escribir una *postcondición* para el primer bucle.
  3. Escribir sendos *invariantes y funciones de cota* para los dos bucles de forma que permitan demostrar la corrección de la función.
  4. Indicar y justificar el *coste asintótico* en el caso peor de la función.
2. (3 puntos) Tenemos dos cadenas de caracteres de la misma longitud y no vacías. La primera se denomina *cadena principal* y la segunda es la *cadena auxiliar*, con la que podemos intercambiar los elementos de la cadena principal que se encuentren en la misma posición. El objetivo es minimizar el número de caracteres diferentes de la cadena principal haciendo intercambios con la cadena auxiliar.

Por ejemplo, dada la *cadena principal* aba, con dos caracteres diferentes: a y b y la cadena auxiliar eao, podemos intercambiar los elementos en la segunda posición (el carácter b de la *cadena principal* por el carácter a de la *cadena auxiliar*) y obtenemos una cadena con un único carácter diferente: a.

Se pide implementar un algoritmo de *vuelta atrás* que devuelva el mínimo número de caracteres diferentes que se puede conseguir en la cadena principal. Completa para ello el código del fichero main2Final.cpp.

*Ayuda:* Ten en cuenta que las cadenas están formadas por caracteres de la 'a' a la 'z' del código ASCII. La posición de una letra, car, en el alfabeto teniendo en cuenta la distribución de los caracteres del código ASCII se calcula como: car - 'a'.

le

Annotated Program:

```
{P}
m,n=0,0
{I1}
while B1: ( m<N and v[0]!=w[m])
    m=m+1
{I2}
while B2: ( n<N and v[n]==w[(n+m)%M])
    n=n+1
{Q}
```

where

A :  $m = \min j : 0 \leq j \leq N \ \&\& \ (j < N \rightarrow W[j] = V[0]) : j$

P :  $N \geq 0$  and  $\text{Diff}(V, 0, N)$  and  $\text{Diff}(W, 0, N)$   
Q :  $N = \min i : 0 \leq i \leq N \ \&\& \ (i < N \rightarrow V[i] \neq W[(m+i)\%N]) : i$  and A

I1 :  $A[N/m]$  and  $0 \leq m \leq M$  and  $n=0$

I2 :  $Q[N/n]$  and  $0 \leq n \leq N$

Informally, Q formalizes the first number  $n < N$  "not rotated", or N if all of them are ... In former case, return false, otherwise true.

A keeps the first  $m < N$  s.t  $W[n] = V[0]$ . N if all are different,

Proof (NOT REQUIRED)

Partial correctness.

{P}n,m=0,0;while B1 do...done; while B2 do..done;{Q}

```
0 . ( n<N \&\& V[n] \neq W[(m+n)\%N] || n=N \ \&\& Q[N/n] -> Q (Lemma)
1 . ( m<N \&\& V[0] == W[m] || m=N) \ \&\& A[N/m] -> A (Lemma)

2 . {I1[n/0,m/0]} m,n = 0,0 {I1} (Axiom. Assign)
3 . P -> I1[n/0,m/0]
4 . {P}m,n=0,0{I1} (Strength Pre 2,3)
5 . I1 and B1 -> I1[m/m+1]
6 . {I1} while B1 do m=m+1 {I1 and !B1} (Rule while and 5)
7 . I1 and !B1 -> A (Lemma 1 and Def I1)
8 . I1 and !B1 -> 0 <= n <= N and Q[N/n] (since n=0, and 7)
9 . I1 and !B1 -> I2 (Def I2 and 7 , 8 )
10. {I1} while B1 do m=m+1 done {I2} (Weak Pos 6,9)
11. I2 and B2 -> I2[n/n+1]
12. {I2} while B2 do n=n+1 done {I2 and !B2} (Rule while and 11)
13. I2 -> Q[N/n] (Def I2)
14. I2 and !B2 -> Q (13 and Lemma 0)
15. {I2} while B2 do n=n+1 done {Q} (Weak Pos 12,14)
16. {I1} while B1 do ... done; while B2 do ... done {Q} (Comp.10,15)
17. {P}n,m=0,0;while B1 do...done; while B2 do..done;{Q} (Comp.4,16)
```

Termination:

1. I1 and B1 and  $N-n=T \rightarrow N-(n+1) < T$  (Aritmeth.)
2. I2 and B2 and  $N-m=T \rightarrow N-(m+1) < T$  (Aritmeth.)



/\*

OPTION 1:

PERMUTATION (NOT REQUIRED)

Hard to specify . So 4 points:  
1.5/4 to specify

P : N=M >= 0  
\forall i : 0 <= i, j < N : (i!=j -> V[i]!=V[j] and W[i]!=W[j])

circPerm(V[0..N], W[0..M]) dev b:bool

Q : N = min i : 0 <= i <= N and (i<N->V[i]!=W[(m+i)%N]): i

where

m = min j : 0 <= j <= N && and (j<N -> W[j]=V[0]) : j

!B = n==N || V[n]!=V[(m+n)%M]

B = (n<N && && V[n]==V[(m+n)%M])

I : Q[N/n] and 0 <= n <= N

Invariant Snapshot:

0            n=2            N  
+-----+-----+-----+-----+  
| 8 | 1 | 4 | 6 | 2 |  
+-----+-----+-----+-----+

(m+n)%N            m=3  
+-----+-----+-----+-----+  
| 4 | 6 | 2 | 8 | 1 |  
+-----+-----+-----+-----+  
0                      N

Cuote(N,n) = N-n >= 0

Init:

n,m = 0, min j : 0 <= j <= N && and (j<N -> W[j]=V[0]) : j

(See refinement to implement)

Restore:

Skip:

Step:

n = n + 1 (Clearly cuota decreases)

Pseudocode:

```
n,m = 0, min j : 0 <= j <= N && and (j<N -> W[j]=V[0]) : j
while (n<N && && V[n]==V[(m+n)%M])
    n = n + 1
done
```

Pseudocode: (Init refined)

```
n,m = 0,0
while m<N and V[n]!=V[m]
    m=m+1
    while (n<N && V[n]==V[(m+n)%M])
        n = n + 1
    done
```

I1:  $0 \leq m \leq N$  and  $m = \min j : 0 \leq j \leq m \text{ and } (j < m \rightarrow W[j] = V[0]) : j$

Complexity  $O(N+N)$  i.e  $O(N)$

OPTION 2: (One only loop!)

Q :  $N = \min i : 0 \leq i \leq N \text{ and } (i < N \rightarrow V[i] \neq W[m \% N]) : i$

where

$m = \min j : 0 \leq j \leq N \text{ and } (j < N \rightarrow W[j] = V[0]) : j$

$\neg B : n == N \mid\mid n = 0 \text{ and } m == N \mid\mid n > 0 \text{ and } V[n] \neq V[m \% M]$

$B : n < N \text{ and } (n > 0 \mid\mid m < N) \text{ and } (n == 0 \mid\mid V[n] == V[m \% M])$

$I : (n == N \mid\mid n > 0) \rightarrow Q[N/n]$

Cuote( $N, n, m$ ) =  $2*N - n - \min(N, m) \geq 0$

Init:

```
n,m = 0, 0
```

Step:

```
n,m = n + \chi(V[n]==V[m \% M]), m + 1
```

Pseudocode:

```
n,m = 0, 0
while (n < N && (n > 0 \mid\mid m < N) \text{ and } (n == 0 \mid\mid V[n] == V[m \% M]))
    n,m = n + \chi(V[n]==V[m \% M]), m + 1
/*
#endif OPTION1
int cyrclePermut(const int V[], const int W[], const int N)
{
    int n,m ;
    for (m=0; m < N \text{ and } (V[m] != W[0]); m++ ) ;
    for (n=0; n < N \text{ and } (V[(m+n)\%N] == W[n]); n++ );
    return n==N;
}
#else
int cyrclePermut(const int V[], const int W[], const int N)
{
    int n, m;
    for (n = m = 0; n < N \text{ and } (n \mid\mid m < N) \text{ and } (!n \mid\mid V[n] == W[m \% N]); n+=V[n]==W[m \% N], m++ );
    return n==N;
}
#endif

#define MAX 1000
#include <iostream>
using namespace std;
int main(int argc, char **args)
```

```
{  
    int V[MAX],W[MAX];  
    int N;  
    for ( ; cin >> N; )  
    {  
        for (int n = 0; n < N; n++) cin >> V[n];  
        for (int n = 0; n < N; n++) cin >> W[n];  
        cout << (cyriclePermut(V, W, N)?"TRUE":"FALSE") << endl;  
    }  
    return 1;  
}  
  
/*  
g++ sol1.rafa.cc -o main && ./main  
5  
8 1 4 6 2  
4 6 2 8 1  
TRUE  
5  
2 8 1 4 6  
1 4 6 2 8  
TRUE  
5  
5  
1 5 8 3 2  
2 1 5 8 3  
TRUE  
5  
1 5 8 3 2  
1 5 8 2 3  
FALSE  
0  
TRUE  
3  
1 2 3  
2 3 1  
TRUE  
3  
1 2 3  
2 1 3  
FALSE  
*/
```



```

/*
Backtracking problem.
-----
Solutions are c=[c1,c2...,cn] with ci in A[0..N) or B[..N)

FEASIBILITY: (TRUE)
Sol : n+1 = N
Optimization Criteria : s= #i,j:0<=i,j<N:V[i]!=V[j] as minimum
Marking Criteria for O(1) : M[int(i)]=#i:0<=i<k:sol[i]=char
    ↑
    jin

*/
#include <iostream> // cin, cout
#include <algorithm> // min.
using namespace std;

void minimize(const char V[], const char W[], const int N,
              const int k,
              int mark[], int diff, int &m)
{
    for(int n=0; n < 2 ; n++) // 0 stands for V, 1 stands for W
    {
        if (1) // Pedantic, but according to schema
        {
            const char c = (n==0)?V[k]:W[k];
            mark[c-'a']+=1;
            diff += (mark[c-'a']==1);
            if (k+1==N) // sol
                m = min(diff,m);
            else
                minimize(V,W,N,k+1,mark,diff,m);
            mark[c-'a']-=1;
            diff -= (mark[c-'a']==0);
        }
    }
    return ;
}

#define MAX 1000
int main(int argc, char **args)
{
    char V[MAX],W[MAX];
    int mark[MAX] ; // indexing. mark[c-'a'].
    int N;
    int m,k,diff;
    for ( ; cin >> N; )
    {
        for (int n = 0; n < N; n++) cin >> V[n];
        for (int n = 0; n < N; n++) cin >> W[n];
        k = diff = 0 ; m=32;
        for (char c = 'a'; c <= 'z' ; c++) mark[c-'a']=0;
        minimize(V,W,N,k,mark,diff,m);
        cout << m << endl;
    }
    return 1;
}

```



# Estructura de Datos y Algoritmos

## Grados de la Facultad de Informática (UCM)

Examen FINAL SEPTIEMBRE (Grupo C), 6 de septiembre de 2018

1. (2 puntos) Se dice que dos vectores de enteros del mismo tamaño y cada uno con todos sus elementos distintos entre sí son *desplazamiento circular* el uno del otro cuando la secuencia de elementos de ambos son iguales si no se tiene en cuenta la posición de comienzo de las mismas.

Por ejemplo, [7, 9, 1, 3, 5] es desplazamiento circular de [1, 3, 5, 7, 9] y de [7, 9, 1, 3, 5], pero no lo es de [3, 5, 1, 7, 9].

Se ha desarrollado el siguiente algoritmo para comprobar si dos vectores son desplazamiento circular el uno del otro:

```
bool desplazamiento(const int V[], const int W[], const int N) {
    int i, j;
    i = 0;
    while (i < N && !(V[0] == W[i]))
        i = i + 1;
    j = 0;
    while (j < N && (V[j] == W[(i + j) % N]))
        j = j + 1;
    return (j == N);
}
```

Se pide:

1. Especificar la función *desplazamiento*.
2. Escribir una *postcondición* para el primer bucle.
3. Escribir sendos *invariantes y funciones de cota* para los dos bucles de forma que permitan demostrar la corrección de la función.
4. Indicar y justificar el *coste asintótico* en el caso peor de la función.

2. (3 puntos) Tenemos dos cadenas de caracteres de la misma longitud y no vacías. La primera se denomina *cadena principal* y la segunda es la *cadena auxiliar*, con la que podemos intercambiar los elementos de la cadena principal que se encuentren en la misma posición. El objetivo es minimizar el número de caracteres diferentes de la cadena principal haciendo intercambios con la cadena auxiliar.

Por ejemplo, dada la *cadena principal* aba, con dos caracteres diferentes: a y b y la cadena auxiliar eao, podemos intercambiar los elementos en la segunda posición (el carácter b de la *cadena principal* por el carácter a de la *cadena auxiliar*) y obtenemos una cadena con un único carácter diferente: a. El intercambio tanto de los primeros como de los últimos elementos incrementa el número de elementos diferentes por lo que no es conveniente realizarlo.

Se pide implementar un algoritmo de *vuelta atrás* que devuelva el mínimo número de caracteres diferentes que se puede conseguir en la cadena principal. Completa para ello el código del fichero main2Final.cpp.

Ten en cuenta que las cadenas están formadas por caracteres de la 'a' a la 'z' del código ASCII. La posición de una letra, *car*, en el alfabeto teniendo en cuenta la distribución de los caracteres del código ASCII se calcula como: *car* - 'a'.

- INT  
- PEARLBE  
- MARKIN  
- ARBOL

3. (3 puntos) Queremos añadir al TAD `List` una nueva operación interna de nombre `engordar` que añada a una lista el contenido de otra lista dada de la siguiente forma: los nodos de la segunda lista se colocarán alternativamente al principio y al final de la primera lista. La lista recibida como argumento pasará a ser vacía.

Para resolver este ejercicio no se puede crear ni destruir memoria dinámica (hacer `new` o `delete`), ni tampoco modificar los valores almacenados en las listas enlazadas.

Completa el código del fichero `main3.cpp` con la implementación de esta nueva operación.

4. (2 puntos) Dada una secuencia de números enteros positivos (no necesariamente ordenados y con posibles repeticiones) y un rango  $[inf, sup]$ , se quieren imprimir en orden creciente todos los números del rango que no estén en la secuencia. Para ello se pide implementar una función `enRangoYNoEnSecuencia` que dada la secuencia de entrada `sec` como `list<int>`, y los enteros `inf` y `sup`, devuelva la secuencia de salida, también como `list<int>`, con los números del rango que no estén en `sec`, ordenados crecientemente. Se valorará el coste del algoritmo implementado, el cual debes indicar y justificar. Aclaración: Puedes usar TADs auxiliares. Ejemplo: dada la secuencia 3 9 1 3 7 2 y el rango [3,8] se debe imprimir la secuencia 4 5 6 8.

Completa el código del fichero `main4.cpp` con la implementación de esta operación.

## Normas de realización del examen

1. El material que debes utilizar para la realización del examen (TADs, plantillas y casos de prueba de cada ejercicio) se obtiene pulsando en el ícono del Escritorio “Publicacion docente ...”, después en “Alumno recogida docente”, y en el programa que se abre, abriendo en la parte derecha la carpeta EDA-C, arrastrando los ficheros a hlocal (en la izqda).
2. Escribe tu **nombre, apellidos, DNI y puesto de laboratorio** en un comentario al principio de cada fichero que entregues.
3. Escribe comentarios que expliquen tu solución, justifiquen por qué se ha hecho así y ayuden a entenderla. Calcula la complejidad de todas las funciones que implementes.
4. La entrega se realiza pulsando en el ícono del escritorio “Examenes en Labs ...”, y posteriormente, utilizando el programa que se abre, colocando los ficheros a entregar en la carpeta de vuestro puesto (en el lado derecho).
5. Si la necesitas, puedes encontrar ayuda sobre la librería estándar de C++ en <http://exacrc/cpp/reference/en/>.