Índice general

Capítulo 1

Ad-hoc

1.1. Invertir Bytes

```
// Entrada: Un entero sin signo de 32 bits
// Salida : El mismo numero pero con los bytes invertidos como si fuera una cadena
int reverse_bytes(unsigned int i) {
    return ((i >> 24) ) |
        ((i >> 8) & 0xFF00) |
        ((i << 8) & 0xFF0000) |
        ((i << 24));
}
```

1.2. Año bisiesto

```
bool esBisiesto(int y){
    return ( ((y%4)==0 && (y%100)!=0) || (y%400)==0 );
}
```

1.3. Doomsday

```
//Entrada: Una fecha (ene = 1, ..., dic = 12) Salida: Dia de la semana
string vec[7] = {"DOM","LUN","MAR","MIE","JUE","VIE","SAB"};

string doomsday(int dia, int mes, int anio){
    int a, y, m, d;
    a = (14 - mes) / 12;
    y = anio - a;
    m = mes + 12 * a - 2;
    d = (dia + y + y/4 - y/100 + y/400 + (31*m)/12) % 7;
    return vec[d];
}
```

Capítulo 2

Estructuras

2.1. Union - Find

```
#define MAX 100010

using namespace std;

int padre[ MAX ];
int rango[ MAX ];

void MakeSet( int n ){for(int i = 0 ; i <= n ; ++i )padre[ i ] = i, rango[ i ] =
    0;}
int Find(int x){return (padre[x]==x?x:padre[x]=Find(padre[x]));}
bool sameComponent(int x, int y){ return Find(x) == Find(y);}

void Union( int x , int y ){
    int xRoot = Find( x );
    int yRoot = Find( y );
    if( rango[ xRoot ] > rango[ yRoot ] )
        padre[ yRoot ] = xRoot;
    else{
        padre[ xRoot ] = yRoot;
        if( rango[ xRoot ] == rango[ yRoot ] )
            rango[ yRoot ]++;
    }
}
```

2.2. BIT

```
#include <iostream>
#include <cstring>

using namespace std;

#define MAXN 1000000

int T[MAXN + 1];
int A[MAXN];
int N;
int lowbit(int i) {
    return (i & -i);
}
```

```
int sum(int i){
    int value = 0;
    for(; i > 0; i-= lowbit(i))
        value+= T[i];
    return value;
}
int sum(int i, int j){
    return i > 1 ? sum(j) - sum(i-1) : sum(j);
}
void update(int i, int value){
    for(; i <= N ; i += lowbit(i))
        T[i] += value;
}
void build(){
    memset(T, 0, sizeof(T));
    for(int i=0; i < N; i++)
        update(i+1, A[i]);
}
int main(){
    cin >> N;
    for(int i = 0; i < N; ++i)
        cin >> A[i];

    build();
    cout << sum( 1, N ) << endl;
    return 0;
}
```

2.3. Segment Tree

```
#include <iostream>
#include <cstdio>
#include <algorithm>
using namespace std;

#define MAXN 50001

struct nodo{
    int sum;
    nodo() { }
    nodo(int _sum){
        sum = _sum;
    }
}T[MAXN*4];
int n, a[MAXN];

void update(int b, int e, int node, int i, int val){
    if(i < b || i > e) return;

    if( b == e ) T[node].sum = a[i] = val;
    else{
        int mid = (b + e)/2, le = 2*node, ri = 2*node+1;

        update(b, mid, le, i, val);
        update(mid + 1, e, ri, i, val);

        T[node].sum  = T[le].sum + T[ri].sum;
```

```
    }
}

void init(int b, int e, int node){
    if(b == e) T[node].sum = a[b];
    else{
        int mid = (b + e)/2, le = 2*node, ri = 2*node+1;
        init(b, mid, le);
        init(mid + 1, e, ri);

        T[node].sum  = T[le].sum + T[ri].sum;
    }
}

nodo query(int b, int e, int node, int i, int j){
    if(i <= b && e <= j) return T[node];

    int mid = (b + e) / 2, le = 2*node + 1, ri = 2*node + 2;

    if(j <= mid) return query(b, mid, le, i, j);
    else if(mid < i) return query(mid + 1, e, ri, i, j);
    else{
        nodo ret1 = query(b, mid, le, i, j);
        nodo ret2 = query(mid + 1, e, ri, i, j);

        nodo ret;
        ret.sum  = ret1.sum + ret2.sum;
        return ret;
    }
}

int main(){
    init(0, n, 0);
    printf("%d\n", query(1, n, 1, x, y).sum);
    update(1, n, 1, x, y);
    return 0;
}
```

2.4. Segment Tree con Lazy Propagatión

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;

struct Node{
    long long sum;  //suma de hijos
    long long offset;//suma que no propaga de total nodo
}Tree[500000];

void Update(long long node,long long lo,long long hi,long long i,long long j,
    long long val){
    if(lo>j || hi<i)
        return;

    if(lo>=i && hi<=j){
```

```
17          Tree[node].sum+=(hi-lo+1)*val;
18          Tree[node].offset+=val;
19      }else{
20          long long mid=(lo+hi)>>1;
21          Update(2*node,lo,mid,i,j,val);
22          Update(2*node+1,mid+1,hi,i,j,val);
23          Tree[node].sum=Tree[2*node].sum+Tree[2*node+1].sum+(hi-lo+1)*Tree[node].
                offset;
24      }
25  }
26
27  long long Query(long long node,long long lo,long long hi,long long i,long long j
        ,long long offst){
28      if(lo>j || hi<i)
29          return 0;
30
31      if(lo>=i && hi<=j){
32          return offst*(hi-lo+1)+Tree[node].sum;
33      }else{
34          long long mid=(lo+hi)>>1;
35          offst+=Tree[node].offset;
36          long long q1=Query(2*node,lo,mid,i,j,offst);
37          long long q2=Query(2*node+1,mid+1,hi,i,j,offst);
38          return q1+q2;
39      }
40  }
41  void Clear(long long N){
42      for(long long i=0;i<4*N;i++){
43          Tree[i].offset=Tree[i].sum=0;
44      }
45  }
46
47  int main(){
48      int N,ofst=0;
49      Clear(N);
50      cout<<Query(1,1,N,x,y,ofst)<<endl;
51      Update(1,1,N,x,y,val);
52      return 0;
53  }
```

2.5. Trie

```
1   #include <iostream>
2   #include <vector>
3   #include <algorithm>
4   #include <cstdio>
5   #include <fstream>
6
7   using namespace std;
8   struct Trie{
9       Trie* hijos[26];
10      bool esfin;
11      vector<int> idhijos;
12      Trie(){
13          esfin = false;
14          for(int i = 0; i < 26; ++i)
15              hijos[i] = NULL;
```

```
16      }
17  };
18  void insertar(Trie* v, char* c){
19      if(c[0] == '\0')
20          v->esfin = true;
21      else{
22          int k = c[0] - 'a';
23          if(v->hijos[k] == NULL){
24              v->hijos[k] = new Trie();
25              (v->idhijos).push_back(k);
26          }
27          insertar(v->hijos[k], c+1);
28      }
29  }
30  void ordenarids(Trie *t){
31      sort((t->idhijos).begin(), (t->idhijos).end());
32      int tam = (t->idhijos).size();
33      for(int i = 0; i < tam; i++){
34          int k = t->idhijos[i];
35          ordenarids(t->hijos[k]);
36      }
37  }
38  bool haysol = false;
39  int tmp, caso;
40  char vacio[25]="";
41  string cad;
42  void mostrar(Trie* v, string p, char* pr){
43      if(pr[0] == '\0'){
44          if(v->esfin){
45              haysol = true;
46              puts(p.c_str());
47          }
48          int tam = (v->idhijos).size();
49
50          for(int i = 0; i < tam; ++i){
51              int k = v->idhijos[i];
52              mostrar(v->hijos[k], p + ((char)('a'+k)), vacio);
53          }
54      }
55      else{
56          int k = pr[0] - 'a';
57          if(v->hijos[k] != NULL){
58              bool aux = (v->hijos[k])->esfin;
59              (v->hijos[k])->esfin = false;
60
61              mostrar(v->hijos[k], p, pr+1);
62
63              (v->hijos[k])->esfin = aux;
64
65          }
66      }
67  }
68  int main(){
69      //freopen("entrada.in","r",stdin);
70      Trie* t;
71      t = new Trie();
72      scanf("%d", &tmp);
73      char cadena[25];
```

```
74
75      while(tmp--){
76          scanf("%s", cadena);
77          insertar(t, cadena);
78      }
79      ordenarids(t);;
80      cin >> tmp;
81      for(caso = 1; caso <= tmp; caso++){
82          scanf("%s", cadena);
83          cad = cadena;
84          haysol = false;
85          printf("Case #%d:\n", caso);
86          mostrar(t, cad, cadena);
87          if(!haysol)
88              puts("No match.");
89      }
90      return 0;
91  }
```

Capítulo 3

Complete Search

3.1. El problema de las ocho reinas

```
1
2  int sol[10];
3  bool sePuede(int fila, int col){
4      for(int i = col - 1 ; i >= 1; i--)
5          if(sol[i] == fila || (abs(sol[i] - fila) == abs(i - col)))
6              return false;
7      return true;
8  }
9  // Llamada solve(1)
10 void solve(int col){
11     if(col == 9){
12         for(int i = 1; i <= 8; i++)
13             printf(" %d", sol[i]);
14         printf("\n");
15         return ;
16     }
17     for(int i = 1; i <= 8; i++)
18         if(sePuede(i, col)){
19             sol[col] = i;
20             solve(col + 1);
21         }
22 }
```

Capítulo 4

Divide y venceras

4.1. Busqueda Binaria (Recursivo)

```
1  template <class T>
2  int binsearch(const T array[], int len, T what)
3  {
4      if (len == 0) return -1;
5      int mid = len / 2;
6      if (array[mid] == what) return mid;
7      if (array[mid] < what) {
8          int result = binsearch(array+mid+1, len-(mid+1), what);
9          if (result == -1) return -1;
10         else return result + mid+1;
11
12     }
13     if (array[mid] > what)
14         return binsearch(array, mid, what);
15
16 }
17
18 #include <iostream>
19 int main()
20 {
21     int array[] = {2, 3, 5, 6, 8};
22     int result1 = binsearch(array, sizeof(array)/sizeof(int), 4),
23         result2 = binsearch(array, sizeof(array)/sizeof(int), 8);
24     if (result1 == -1) std::cout << "4 not found!" << std::endl;
25     else std::cout << "4 found at " << result1 << std::endl;
26     if (result2 == -1) std::cout << "8 not found!" << std::endl;
27     else std::cout << "8 found at " << result2 << std::endl;
28
29     return 0;
30
31 }
```

4.2. Busqueda Binaria (Iterativo)

```
1  template <class T>
2  int binSearch(const T arr[], int len, T what) {
3      int low = 0;
4      int high = len - 1;
5      while (low <= high) {
```

```
6           int mid = (low + high) / 2;
7           if (arr[mid] > what)
8               high = mid - 1;
9           else if (arr[mid] < what)
10              low = mid + 1;
11          else
12              return mid;
13
14      }
15      return -1; //indica que no esta
16  }
```

4.3. MergeSort

Primera implementacion:

```
1   //Entrada: p = indice del primer elemento a ordenar
2   //         r = indice del ultimo elemento a ordenar
3   //Salida: El vector "a" ordenado
4   int a[LIM], L[LIM/2 + 4], R[LIM/2 + 4];
5   void fusionar(long p,long q,long r) {
6       long i,j,k,ind1,ind2;
7       ind1 = ind2 = 1;
8       for(i = p; i <= q; i++)
9           L[ind1++] = a[i];
10      L[ind1] = inf;
11      for(i = q+1; i <= r; i++)
12          R[ind2++] = a[i];
13      R[ind2] = inf;
14      i = j = 1;
15      for(k = p; k <= r; k++)
16          if(L[i] > R[j]) {
17              a[k] = R[j];
18              j++;
19          } else {
20              a[k] = L[i];
21              i++;
22          }
23  }
24  void mergeSort(long p,long r) {
25      if(p < r) {
26          long q = (p+r)/2;
27          mergeSort(p,q);
28          mergeSort(q+1,r);
29          fusionar(p,q,r);
30      }
31  }
```

Segunda implementación:

```
1   //Entrada: Un array
2   //Salida: El vector "array" ordenado
3   //        La cantidad optima de intercambios
4   int array[MAXN];
5   int temp[MAXN];
6   int intercambios = 0;
7   int d = 0;
8   void mergesort(int inicio, int length) {
```

```
9       if(length<=1)
10          return;
11      mergesort(inicio, length/2);
12      mergesort(inicio+length/2, length-length/2);
13      int i = inicio;
14      int j = inicio+length/2;
15      int n = 0;
16      while(i<inicio+length/2 && j<inicio+length)
17          if(array[i]<array[j]) {
18              temp[n++] = array[i++];
19              intercambios+=d; //para contar los intercambios
20          } else {
21              temp[n++] = array[j++];
22              d++;
23          }
24      while(i<inicio+length/2) {
25          temp[n++] = array[i++];
26          intercambios+=d;
27      }
28      while(j<inicio+length)
29          temp[n++] = array[j++];
30      memcpy(array+inicio, temp, length*sizeof(int));
31  }
```

4.4. Meet in the Middle

Si generamos los subconjuntos de una lista de N elementos; sabemos que existen 2N subconjuntos, pero esta complejidad es demasiado cuando N es grande, digamos (N>20).

Problema.- Dado una secuencia de N (1<= N <=34) números, determinar cuántos subconjuntos de esta secuencia tiene una suma entre A y B.

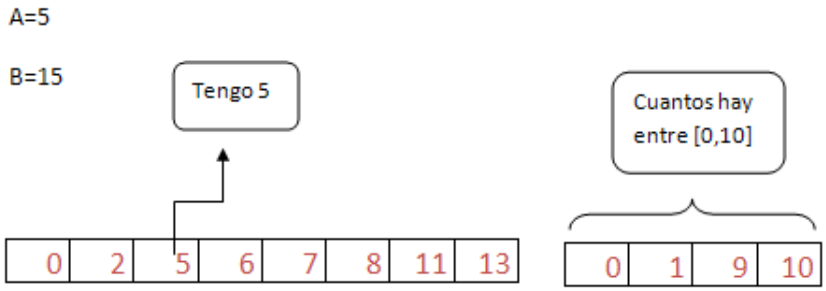
Esta técnica consiste en *dividir en dos listas por la mitad*, Luego generamos las sumatorias de los subconjuntos de cada lista y *los ordenamos*. Luego combinar ambos resultados.

Considerare la sumatoria de conjunto vacio como 0.

Ejemplo:

L= { 2 , 5 , 6 , 1, 9};
A=5, B=15
Los dividimos en los listas por la mitad.
X= { 2 , 5, 6}
Y= { 1 , 9}
Generamos los subconjuntos de cada lista.
X' = { 0, 2, 5 , {2,5}, 6,{2,6}, {5,6} ,{2,5,6} }
Y'= { 0, 1, 9, {1,9} }
Sumatorias de cada subconjunto de manera ordenada.
X'= {0,2,5,6,7,8,11,13}
Y'= {0,1,9,10}
Combinar resultados
Tenemos que recorrer la lista X' y ver cuánto nos falta para llegar al intervalo [A, B].

- Si el primer elemento de X' tenemos 0 entonces en la lista Y' tenemos que buscar cuantos elementos existen entre 5 y 15.
- Si el segundo elemento de X' tenemos 2 entonces en la lista Y' buscamos cuantos existen entre 3 y 13.



x

Si bien se dan cuenta es muy importante tener ambas listas de una manera ordenada para poder buscar cuantos elementos existen en la segunda matriz entre ese intervalo.

El algoritmo sigue buscando hasta terminar con el último elemento de la primera lista.

```
1  #include <vector>
2  #include <algorithm>
3  #include <iostream>
4
5  #define all(v) (v).begin(),(v).end()
6  #define rall(v) (v).rbegin(),(v).rend()
7  #define sz size()
8
9  using namespace std;
10 vector<int>A,B;
11 void subsets(vector<int> v, vector<int>&ans){
12     ans.clear();
13     int n=v.sz;
14     for(int mask=0;mask<(1<<n);mask++){
15         int sum=0;
16         for(int j=0;j<v.sz;j++)if(mask&(1<<j))sum+=v[j];
17         ans.push_back(sum);
18     }
19     sort(all(ans));
20 }
21
22 int main(){
23
24     int N,X,Y,n,nx;
25     cin>>N>>X>>Y;
26     vector<int>v(N);
27     for(int i=0;i<N;i++)cin>>v[i];
28     n=N/2;
29     subsets(vector<int>(v.begin(),v.begin()+n),A);
30     subsets(vector<int>(v.begin()+n,v.end()),B);
31     long long ans=0;
32     for(int i=0;i<A.sz;i++){
33         int val=A[i];
34         int frs=X-val;
35         int scd=Y-val;
36         int x1=lower_bound(all(B),frs)-B.begin();
37         int y1=upper_bound(all(B),scd)-B.begin();
38         if(y1>=x1)ans+=(y1-x1);
39     }
40     cout<<ans<<endl;
41     return 0;
42 }
```

Capítulo 5

Programación Dinamica

5.1. El problema de la mochila y Subset Sum

Subset Sum:

$$OPT(j,W) = max \begin{cases} OPT(j-1,W) & (w_j > W) \\ w_j + OPT(j-1,W-w_j) & (w_j \leq W) \end{cases}$$

0-1 Mochila

$$OPT(j,W) = max \begin{cases} OPT(j-1,W) & (w_j > W) \\ \mathbf{v_j} + OPT(j-1,W-w_j) & (w_j \leq W) \end{cases}$$

```cpp
#include <cstdio>
#include <cstring>

const int MAXITEMS = 2005;
const int MAXCAP   = 2005;

using namespace std;

int espaciolibre, elementos;
int peso[MAXITEMS], ganancia[MAXITEMS];

int dp[MAXITEMS][MAXCAP];
int mochila(int elem, int libre){
    if(elem == -1)
        return 0;
    int &ans = dp[elem][libre];

    if(ans == -1 )
        if(peso[elem] <= libre) // Â¿ el elemento elem cabe en el espacio libre en la mochila ?
            ans =  max(mochila(elem - 1, libre), ganancia[elem] + mochila(elem - 1, libre - peso[elem]));/*si cabe, asi que probamos cuando no tomamos el elemento y probamos si si metieramos el elemento*/
        else
            ans =  mochila(elem - 1, libre); /* no cabe, asi que lo ignoramos e intetamos meter los demas elementos */
    return ans;
}

int main(){
    scanf("%d%d", &espaciolibre, &elementos);
    for(int i = 0; i < elementos; i++)
```

```
29          scanf("%d%d", &peso[i], &ganancia[i]);
30      memset(dp, -1, sizeof dp);
31      printf("%d\n", mochila( elementos -1, espaciolibre ));
32      return 0;
33  }
```

5.2. Cambio de modenas

```
1   #define clr(a)  memset(a,0,sizeof(a))
2   using namespace std;
3   long long int dp[30001];
4   int money[]={1,5,10,25,50};
5   void coinchange(int sum){
6       clr(dp);
7       dp[0]=1;
8       for(int i=0;i<5;i++){
9           for(int j=money[i];j<=sum;j++){
10              dp[j]+=dp[j-money[i]];
11          }
12      }
13  }
```

5.3. LIS ($O(n^2)$)

```
1   #include <stdio.h>
2   #define SIZE 200000
3   #define MAX(x,y) ((x)>(y)?(x):(y))
4
5   int best[SIZE];         // best[] holds values of the optimal sub-sequence
6
7   int main (void) {
8       int i, n, k, x, sol = -1;
9       scanf ("%d", &n);   // N = how many integers to read in
10      for (i = 0; i < n; i++) {
11          best[i] = -1;
12          scanf ("%d", &x);
13          for (k = 0; best[k] > x; k++)
14              ;
15          best[k] = x;
16          sol = MAX (sol, k + 1);
17      }
18      printf ("best is %d\n", sol);
19      return 0;
20  }
```

5.4. LIS ($O(n\,log(n))$)

```
1   #include <stdio.h>
2   #define SIZE 200000
3   #define MAX(x,y) ((x)>(y)?(x):(y))
4
5   int best[SIZE];         // best[] holds values of the optimal sub-sequence
```

```
6
7  int main (void) {
8      int i, n, k, x, sol;
9      int low, high;
10
11     scanf ("%d", &n);   // N = how many integers to read in
12     // read in the first integer
13     scanf ("%d", &best[0]);
14     sol = 1;
15     for (i = 1; i < n; i++) {
16         best[i] = -1;
17         scanf ("%d", &x);
18
19         if(x >= best[0]) {
20             k = 0;
21             best[0] = x;
22         }
23         else {
24             // use binary search instead
25             low = 0;
26             high = sol-1;
27             for(;;) {
28                 k = (int) (low + high) / 2;
29                 // go lower in the array
30                 if(x > best[k] && x > best[k-1]) {
31                     high = k - 1;
32                     continue;
33                 }
34                 // go higher in the array
35                 if(x < best[k] && x < best[k+1]) {
36                     low = k + 1;
37                     continue;
38                 }
39                 // check if right spot
40                 if(x > best[k] && x < best[k-1])
41                     best[k] = x;
42                 if(x < best[k] && x > best[k+1])
43                     best[++k] = x;
44                 break;
45             }
46         }
47         sol = MAX (sol, k + 1);
48     }
49     printf ("best is %d\n", sol);
50     return 0;
51 }
```

5.5. Maxima submatriz de ceros

```
1  #include <vector>
2  #include <iostream>
3  #include <stack>
4  using namespace std;
5  int main(){
6      int n, m;
7      cin >> n >> m;
8      vector < vector<char> > a (n, vector<char> (m));
```

```
    for (int i=0; i<n; ++i)
        for (int j=0; j<m; ++j)
            cin >> a[i][j];

    int ans = 0;
    vector<int> d (m, -1);
    vector<int> dl (m), dr (m);
    stack<int> st;
    for (int i=0; i<n; ++i) {
        for (int j=0; j<m; ++j)
            if (a[i][j] == 1)
                d[j] = i;
        while (!st.empty()) st.pop();
        for (int j=0; j<m; ++j) {
            while (!st.empty() && d[st.top()] <= d[j]) st.pop();
            dl[j] = st.empty() ? -1 : st.top();
            st.push (j);

        }
        while (!st.empty()) st.pop();
        for (int j=m-1; j>=0; --j) {
            while (!st.empty() && d[st.top()] <= d[j]) st.pop();
            dr[j] = st.empty() ? m : st.top();
            st.push (j);
        }
        for (int j=0; j<m; ++j)
            ans = max (ans, (i - d[j]) * (dr[j] - dl[j] - 1));
    }
    cout << ans;
    return 0;
}
```

5.6. Submatriz de suma maxima

```
void subMatriz_con_sumaMaxima(int m[n][n]){
    for(int i = 1; i <= n; i++){
        vector<int> vc(n + 1,0);//para aplicar DP magico
        for(int j = i; j <= n; j++){
            mn_temp = n * n;
            for(int k = 1; k <= n; k++)
                vc[k] += m[j][k];

            dp[0] = 0;
            for(int i = 1; i <= n;i++){
                dp[i] = max(vc[i], dp[i-1] + vc[i]);
                if(res < dp[i])
                    res = dp[i];
            }
        }
    }

    printf("%d\n",res);
}
```

5.7. Distancia de edición (Algoritmo de Levenshtein)

```cpp
#include <string>
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;
/*Encuentra la distancia de edicion entre 2 cadenas*/
/*
El numero minimos de cambios para transformar la cadena s1 en la cadena s2, con los siguientes cambios:

- Elimina una letra de una de las cadenas
- Inserta una letra en una de las cadenas
- Reemplaza una letra en una de las cadenas
*/
int levenshtein(const string &s1, const string &s2)
{
    int N1 = s1.size();
    int N2 = s2.size();
    int i, j;
    vector<int> T(N2+1);

    for ( i = 0; i <= N2; i++  )
        T[i] = i;

    for ( i = 0; i < N1; i++  ) {
        T[0] = i+1;
        int corner = i;
        for ( j = 0; j < N2; j++  ) {
            int upper = T[j+1];
            if ( s1[i] == s2[j]  )
                T[j+1] = corner;
            else
                T[j+1] = min(T[j], min(upper, corner)) + 1;
            corner = upper;

        }

    }
    return T[N2];
}
int main(){
    int casos;
    cin >> casos;
    string a, b;
    while(casos--){
        cin >> a >> b;
        if(a.size() > b.size())
            swap(a,b);
        cout << levenshtein(a,b) << endl;
    }
    return 0;
}
```

Capítulo 6

Grafos

6.1. BFS

```
#include <queue>
#include <vector>
#include <cstring>

#define MAXN 1000

using namespace std;

vector<int> grafo[MAXN];
int d[MAXN];
queue<int> cola;

void BFS(int ini){
    memset(d,-1, sizeof d);
    cola.push(ini);
    d[ini] = 0;
    while(!cola.empty()){
        int act = cola.front();cola.pop();
        for(int i = 0; i < grafo[act].size(); i++){
            int ady = grafo[act][i];
            if(d[ady] == -1){
                d[ady] = d[act] + 1;
                cola.push(ady);
            }
        }
    }
}
```

6.2. DFS

```
#define MAX 1001
#define pain 64
int n;
int mat[MAX][MAX];
bool visitado[MAX][MAX];
int di[]={1,1, 1,-1,-1,-1,0, 0};
int dj[]={1,0,-1, 1, 0,-1,1,-1};

void ff_dfs(int i, int j) {
```

```
    visitado[i][j] = true;
    for(int k=0; k<8; k++) {
        int a = di[k] + i;
        int b = dj[k] + j;
        if(a >= 0 && b >= 0 && a < n && b < n &&!visitado[a][b]) {
            if(mat[a][b]==pain) ff_dfs(a,b);
            else visitado[a][b]=true;
        }
    }
}
```

6.3. Topological Sort

G asiclico dirigido (DAG)

Las tareas no son independientes y la ejecución de una tarea sólo es posible si otras tareas que ya se han ejecutado. Solo hay un orden

```
using namespace std;
vector<int>graph[110];
bool visitado[110];
vector<int> sol;
int n,m;
void dfs(int node) {
    visitado[node] = true;
    for (int i = 0; i < graph[node].size(); i++) {
        if (!visitado[graph[node][i]])
            dfs(graph[node][i]);
    }
    sol.push_back(node);
}
void lim(){
    for(int i=0;i<=110;i++)
        graph[i].clear();

    clr(visitado);
    sol.clear();
}
int main() {
    //llenado
        for (int i = 1; i <= n; i++)
            if(!visitado[i])dfs(i);
        reverse(sol.begin(), sol.end());
        //printsol
    return 0;
}
```

6.4. Dijkstra

```
#define f first
#define s second
#define pb push_back
#define mp make_pair
using namespace std;

typedef int V;          //tipo coste
```

```
typedef pair<V,int> P;  //(coste,nodo)
typedef set<P> S;
int N;
vector<P>A[10001];      //COSTE NODO
V dijstra(int s, int t){
    S m;
    vector<V>z(N,1000000000);
    z[s]=0;
    m.insert(mp(0,s));
    while(m.size() > 0){
        P p=*m.begin();
        m.erase(m.begin());
        if(p.s==t)return p.f;
        for(int i=0 ;i<A[p.s].size();i++){
            P q(p.f + A[p.s][i].f, A[p.s][i].s);
            if(q.f < z[q.s]){
                m.erase(mp(z[q.s],q.s));
                m.insert(q);
                z[q.s]=q.f;
            }
        }
    }
    return -1;
}
int main(){
    N=6;
    //llenado
    //A[a].pb(mp(c,b)); arista a,b coste c
    A[0].pb(mp(2,1));//arista(0,1) coste 2
    //solucion
    printf("%d\n",dijstra(4,5));
    return 0;
}
```

6.5. BellmandFord

Si un grafo contiene un ciclo de coste total negativo entonces este grafo no tiene solución

```
#include <cstdio>
#include <cstdlib>
#include <iostream>
#include <vector>
#define f first
#define s second
#define pb push_back
#define mp make_pair
using namespace std;

#define M 1000000000
typedef pair<pair<int,int>,int> P;
int N;
vector <P> v;
int bellmandford(int a, int b){
    vector<int>d(N,M);
    d[a]=0;
    for(int i=0;i<N;i++){
        for(int j=0;j<v.size();j++){
```

```
            if(d[v[j].f.f]<M && d[v[j].f.f]+v[j].s < d[v[j].f.s]){
                d[v[j].f.s] = d[v[j].f.f]+v[j].s;
            }
        }
    }
    for(int j=0;j<v.size();j++){
        if(d[v[j].f.f] < M && d[v[j].f.f]+v[j].s < d[v[j].f.s]){
            return -M; /// ciclo negativo
        }
    }
    return d[b];
}
int main(){
    N=5;
    v.pb(mp(mp(0,1),-1));
    v.pb(mp(mp(0,2),0));
    v.pb(mp(mp(1,3),3));
    v.pb(mp(mp(2,1),2));
    v.pb(mp(mp(3,2),-6));
    v.pb(mp(mp(3,4),-3));
    printf("%d\n",bellmandford(0,4));
    return 0;
}
```

6.6. Floyd Warshall

```
int dp[100][100]; //grafo matriz de ady

void floyd_warshall(){
   for(int k=1;k<=n;k++)
      for(int i=1;i<=n;i++)
         for(int j=1;j<=n;j++)
            dp[i][j] = min(dp[i][j],dp[i][k] + dp[k][j]);
}
```

6.7. Componentes Fuertemente Conexas

```
#include <cstring>
#include <string>
#include <cstdio>
#include <cstdlib>
#include <vector>
#include <algorithm>
#include <iostream>

#define clr(a)  memset(a,0,sizeof(a))
#define pb push_back
using namespace std;
vector<int>grafo[N];
vector<int>gaux;
bool visitado[N];
void dfs(int u){
    visitado[u]=true;
    for(int i=0;i<grafo[u].size();i++){
```

```
        if(!visitado[grafo[u][i]]) dfs(grafo[u][i]);
    }
}
int main(){
    //r(input);
    int t;
    int x,y;
    int n,m;
    sc("%d",&t);
    while(t--){
        sc("%d %d",&n,&m);
        clr(visitado);
        for(int i=0;i<=n;i++)   grafo[i].clear();
            gaux.clear();
        while(m--){
            sc("%d %d",&x,&y);
            grafo[x].pb(y);
        }
        int M=0;
        for(int i=1;i<=n;i++){
            if(!visitado[i]){
                dfs(i),gaux.pb(i);
            }
        }
        clr(visitado);
        M=0;
        for(int i=gaux.size()-1;i>=0;i--){
            if(!visitado[gaux[i]]){
                dfs(gaux[i]);
                M++;
            }
        }
        cout<<M<<endl;
    }
}
```

6.8. Componentes Fuertemente Conexas (Kosaraju)

```
#include <stack>
#include <queue>
#include <vector>
#include <cstring>

#include <map>
#include <cstdio>
#include <iostream>

using namespace std;

/*
Algoritmo de Kosaraju para
Componentes Fuertemente Conexas
*/

#define MAX_V 1000

int V,num_scc;
```

```
20 vector< vector<int> > G;
21 vector< vector<int> > GT;
22 bool visited[MAX_V];
23 stack<int> S;
24 queue<int> Q;
25
26 void dfs(int v){
27     visited[v] = true;
28
29     for(int i=G[v].size()-1;i>=0;--i)
30         if(!visited[G[v][i]])
31             dfs(G[v][i]);
32
33     S.push(v);
34 }
35
36 void bfs(int v){
37     Q.push(v);
38     visited[v] = true;
39
40     int aux;
41
42     while(!Q.empty()){
43         aux = Q.front();
44         Q.pop();
45
46         for(int i=GT[aux].size()-1;i>=0;i--){
47             if(!visited[GT[aux][i]]){
48                 Q.push(GT[aux][i]);
49                 visited[GT[aux][i]] = true;
50             }
51         }
52     }
53 }
54 void SCC(){
55     memset(visited,false,sizeof(visited));
56
57     for(int i=0;i<V;++i) if(!visited[i]) dfs(i);
58
59     num_scc = 0;
60     int aux;
61
62     memset(visited,false,sizeof(visited));
63
64     for(int i=0;i<V;++i){
65         aux = S.top();
66         S.pop();
67
68         if(!visited[aux]){
69             bfs(aux);
70             ++num_scc;
71         }
72     }
73 }
74
75 int main(){
76     int E,u,v;
77     string s;
```

```
78      map<string, int> num;
79
80      while(true){
81          scanf("%d %d",&V,&E);
82          if(V==0) break;
83
84          getline(cin,s);
85          num.clear();
86
87          for(int i=0;i<V;++i){
88              getline(cin,s);
89              num[s] = i;
90          }
91
92          G.clear(); G.resize(V);
93          GT.clear(); GT.resize(V);
94
95          for(int i=0;i<E;++i){
96              getline(cin,s);
97              u = num[s];
98              getline(cin,s);
99              v = num[s];
100
101             G[u].push_back(v);
102             GT[v].push_back(u);
103         }
104
105         SCC();
106
107         printf("%d\n",num_scc);
108     }
109
110     return 0;
111 }
```

6.9. Componentes Fuertemente Conexas (Tarjan $O(n+v)$)

```
1  #include<iostream>
2  #include<vector>
3  #include<cstdio>
4
5  using namespace std;
6
7  typedef long long lli;
8  typedef pair<int, int> pii;
9  #define LENGTH(a) ((int)a.length())
10 #define SIZE(a) ((int)a.size())
11 int const MAX = 100;
12 int const INF = 1000000;
13
14 class Grafo
15 {
16 public:
17     vector<vector<int> > L;
18     vector<int> num, low;
19     vector<bool> visitado;
20     vector<int> componente;
```

```
21      int nComponentes, numCnt;
22
23      Grafo(int n)
24      {
25          init(n);
26      }
27
28      void init(int n)
29      {
30          L = vector<vector<int> >(n);
31          num = vector<int>(n, -1);
32          low = vector<int>(n, 0);
33          visitado = vector<bool>(n, false);
34          componente.clear();
35          nComponentes = numCnt = 0;
36      }
37
38      void add(int a, int b)
39      {
40          L[a].push_back(b);
41      }
42
43      void ejecutarTarjan()
44      {
45          for (int i = 0; i < SIZE(L); ++i)
46              if (num[i] == -1)
47                  tarjan(i);
48      }
49
50      void tarjan(int u)
51      {
52          low[u] = num[u] = numCnt++;
53          visitado[u] = true;
54          componente.push_back(u);
55          for (int i = 0; i < SIZE(L[u]); ++i)
56          {
57              int v = L[u][i];
58              if (num[v] == -1)
59                  tarjan(v);
60              if (visitado[v])
61                  low[u] = min(low[u], low[v]);
62          }
63          if (num[u] == low[u])
64          {
65              nComponentes++;
66              int v;
67              do
68              {
69                  v = componente[SIZE(componente) - 1];
70                  visitado[v] = false;
71                  componente.erase(--componente.end());
72              }
73              while (v != u);
74          }
75      }
76  };
77
78  int main(){
```

```
int nodos;
cin>>nodos;
Grafo g(nodos);
//leer lados
g.ejecutarTarjan();
//g.nComponentes contiene el numero de SCC
return 0;
}
```

6.10. Minimum Spanning Tree (Kruskall)

```
#define lim 100000
using namespace std;
typedef vector<pair<int,pair<int,int> > > V;
int N,lider[lim];
V v;
void init(){
    Sort(v);
    for(int i=0;i<N;i++)lider[i]=i;
}
int find(int n){
    if(n==lider[n])return n;
    else return lider[n]=find(lider[n]);
}
int kruskall(){
    int a,b,sum=0;
    init();
    for(int i=0;(int)i<v.sz;i++){
        a=find(v[i].s.f);
        b=find(v[i].s.s);
        if(a!=b){
            lider[b]=a;
            sum+=v[i].f;
        }
    }
    return sum;
}
//v.pb(mp(c,mp(a,b))); peso(c),arista(a,b)
```

6.11. MaxFlow

```
#include <algorithm>
#include <cstring>
#include <cstdio>
#include <limits>
#include <queue>

using namespace std;

#define N 1000 //maximo de nodos
//halla el flujo maximo desde el nodo s
//hasta el nodo t
//cap[][] almacena flujos parciales
int cap[N][N], padre[N], n, s, t;
```

```
14
15 bool bfs() {
16     queue<int> q;
17     q.push(s);
18     memset(padre, -1, sizeof padre);
19     padre[s] = s;
20     while (q.size()) {
21         int u = q.front();
22         q.pop();
23         if (u == t)
24             return true;
25         for (int v = 0; v < n; ++v)
26             if (padre[v] == -1 && cap[u][v])
27                 padre[v] = u, q.push(v);
28     }
29     return false;
30 }
31
32 int maxFlow() {
33     int mf = 0, f, v;
34     while (bfs()) {
35
36         v = t;
37         f = numeric_limits<int>::max();
38         while (padre[v] != v)
39             f = min(f, cap[padre[v]][v]), v = padre[v];
40
41         v = t;
42         mf += f;
43         while (padre[v] != v)
44             cap[padre[v]][v] -= f, cap[v][padre[v]] += f, v = padre[v];
45     }
46     return mf;
47 }
48
49
50 #define capacidad(i, j, c) cap[i][j] += c, cap[j][i] += c
51 int main() {
52     int c;
53         printf("ingrese el numero de nodos\n");
54         scanf("%d", &n);
55         printf("ingrese el inicio destino y nro nodos");
56         memset(cap, 0, sizeof cap);
57         scanf("%d %d %d", &s, &t, &c);
58         --s, --t;
59         printf("ingrese los lados x y & capacidad");
60         for (int i = 0, x, y, z; i < c; ++i)
61             scanf("%d %d %d", &x, &y, &z), capacidad(x - 1, y - 1, z);
62
63
64         printf("el flujo maximo entre s y t es %d\n", maxFlow());
65
66     return 0;
67 }
```

Capítulo 7

Matemáticas

7.1. GCD

```
1  //Entrada: Dos enteros a y b
2  //Salida: El maximo comun divisor entre a y b
3
4  #include <algorithm>
5  int gcd (int a, int b){
6      __gcd(a,b);
7  }
8
9  int gcd (int a, int b){
10     return b==0?a:gcd(b, a  % b);
11 }
12
13 int gcd (int a, int b){
14     int c = a  % b;
15     while(c != 0){
16         a = b;
17         b = c;
18         c = a  % b;
19     }
20     return b;
21 }
```

7.2. LCM

```
1  //Entrada: Dos enteros a y b
2  //Salida: El minimo comun multiplo entre a y b
3  int lcm (int a, int b) {
4      return a / gcd (a, b) * b;
5  }
```

7.3. Exponenciación rapida

```
1  long long int pow(int b, int e){
2      if(!e) return 1;
3
4      long long int ans = pow(b, e>>1);
5      ans *= ans;
```

```
6
7       if(e&1)
8           ans *= b;
9       return ans;
10  }
11
12  unsigned long long int modpow(long long int b, int e, int m){
13      if(!e) return 1;
14
15      unsigned long long int ans = modpow(b, e>>1, m);
16      ans = (ans * ans) % m;// cuidado con desborde en la multiplicacion
17
18      if(e&1)
19          ans = (ans * b) % m;
20      return ans;
21  }
22  //podemos utilizar una idea similar para evitar el desborde en la multiplicacion
23  int multi(long long a, long long b, long long mod){
24      if(b == 0) return 0;
25      long long int ans = (multi( a, b / 2, mod ) * 2) % mod;
26      if( b % 2 == 1 ) ans = (ans + a) % mod;
27      return ans;
28  }
```

7.4. Criba de Erathostenes

```
1   #include <vector>
2   #include <bitset>
3
4   #define MAXPRIME 3162300 // raiz de maximo numero que manejaremos
5
6   using namespace std;
7
8   bitset<MAXPRIME> criba;
9   vector<long long int> primos;
10
11  void llena_criba()
12  {
13      criba.set();// marcamos todo como false
14      for (int i = 2; i < MAXPRIME; i++)
15          if (criba.test( i ))
16          {
17              primos.push_back( i );
18              for (int j= i + i; j<MAXPRIME; j += i)
19                  criba.reset( j );
20          }
21  }
```

7.5. Triangulo de Pascal

```
1   #define MAXN 100 // largest n or m
2
3   int n,m;
4   long binomial_coefficient(n,m){
```

```
    int i,j;
    long bc[MAXN][MAXN];
    for (i=0; i<=n; i++) bc[i][0] = bc[i][i]  = 1;
    for (i=1; i<=n; i++)
        for (j=1; j<i; j++)
            bc[i][j] = bc[i-1][j-1] + bc[i-1][j];
    return bc[n][m];

}
```

7.6. Combinaciones(Para numeros muy grandes)

```
unsigned long long C(int n, int k) {
    k = min(k,n-k);
    unsigned long long res = 1;
    for(int i = 0; i < k; i++)
    {
        res *= (n-i);
        res /= (i+1);
    }
    return res;
}
```

7.7. Polinomios

```
class Polinomio {
    double[] coef;

    // a0 + a1x + a2x^2 + ...
    public Polinomio(double[] coef) {
        this.coef = coef;
    }

    public double integrar(double a, double b) {
        Polinomio integ = integral();
        return integ.evaluar(b) - integ.evaluar(a);
    }

    public double evaluar(double x) {
        double result = 0;
        for (int i = coef.length - 1; i >= 0; i--)
            result = coef[i] + x*result;
        return result;
    }

    public Polinomio integral() {
        double[] newCoef = new double[coef.length + 1];
        newCoef[0] = 0;
        for (int i = 1; i < newCoef.length; i++)
            newCoef[i] = coef[i-1]/i;
        return new Polinomio(newCoef);
    }

    public Polinomio cuadrado() {
```

```
30          double[] newCoef = new double[coef.length * 2 - 1];
31          for (int i = 0; i < newCoef.length; i++)
32              for (int j = 0; j <= i; j++)
33                  if (j < coef.length && i - j < coef.length)
34                      newCoef[i] += coef[j] * coef[i-j];
35
36          return new Polinomio(newCoef);
37      }
38
39      public Polinomio multiplicar(double c) {
40          double[] newCoef = new double[coef.length];
41          for (int i = 0; i < newCoef.length; i++)
42              newCoef[i] = coef[i] * c;
43          return new Polinomio(newCoef);
44      }
45  }
```

7.8. Fibonacci ($O(log(n))$)

```
1  //Entrada: Un entero n
2  //Salida: El n-esimo fibonacci en tiempo O(log(n))
3  long long int fibo(int n) {
4      long long int a,b,x,y,tmp;
5      a = x = tmp = 0;
6      b = y = 1;
7      while(n!=0)
8          if(n%2 == 0) {
9              tmp = a*a + b*b;
10             b = b*a + a*b + b*b;
11             a = tmp;
12             n = n/2;
13         } else {
14             tmp = a*x + b*y;
15             y = b*x + a*y + b*y;
16             x = tmp;
17             n = n-1;
18         }
19     // x = fibo(n-1)
20     return y;// y = fibo(n)
21 }
```

7.9. Multiplicación entero por cadena

```
1  //Entrada: Una cadena y un numero entero
2  //Salida: La multiplicacion de la cadena con el numero
3  #include <iostream>
4  #include <string.h>
5
6  using namespace std;
7
8  unsigned int c,i,n;
9  char res[1000];
10
11 void multi(int n){
```

```
12      for(c = i = 0; res[i]; i++){
13          c = (res[ i ] - '0') * n + c;
14          res[ i ] = c % 10 + '0';
15          c /= 10;
16      }
17      while( c ){
18          res[ i++ ] = '0' + c % 10;
19          c /= 10;
20      }
21      res[i] = 0;
22  }
23
24  int main(){
25      string cad;
26      int m;
27
28      cin >> cad;
29      cad = string(cad.rbegin(), cad.rend());
30      strcpy(res, cad.c_str());
31
32      multi(m);
33      string r(res);
34      string sal(r.rbegin(), r.rend());
35
36      cout << sal << endl;
37      return 0;
38  }
```

7.10. Multiplicación de numeros grandes (Karatsuba)

```
1  //Entrada: 2 BigInteger's x e y
2  //Salida: La multplicacion de x e y
3  import java.math.BigInteger;
4  class Karatsuba {
5      private final static BigInteger ZERO = new BigInteger("0");
6      public static BigInteger karatsuba(BigInteger x, BigInteger y) {
7          int N = Math.max(x.bitLength(), y.bitLength());
8          if (N <= 2000) return x.multiply(y);
9          N = (N / 2) + (N % 2);
10         BigInteger b = x.shiftRight(N);
11         BigInteger a = x.subtract(b.shiftLeft(N));
12         BigInteger d = y.shiftRight(N);
13         BigInteger c = y.subtract(d.shiftLeft(N));
14         BigInteger ac = karatsuba(a, c);
15         BigInteger bd = karatsuba(b, d);
16         BigInteger abcd = karatsuba(a.add(b), c.add(d));
17         return  ac.add(abcd.subtract(ac).subtract(bd).shiftLeft(N)).add(bd.
               shiftLeft(2*N));
18     }
19 }
```

7.11. Integracion de Simpson

```
1  double a[12];
```

```
2
3  int n;
4
5  double f(double x){
6      double ans = 0;
7      for(int i = 0; i <= n; i++)
8          ans += (a[i] * pow(x,i));
9      return ans*ans*PI;
10 }
11 double S(double a, double b, double c, double h){
12     return (h / 3.0) * ( f(a) + 4.0 * f(c) + f(b));
13 }
14 double SIMP(double a, double b, double E ){
15     double h = (b-a)/2.0,
16            c = (b+a)/2.0;
17
18     double a1 = a,  b1 = c,
19            a2 = c,  b2 = b;
20
21     double c1 = (b1+a1)/2.0,
22            c2 = (b2+a2)/2.0;
23
24     double L = (S(a1,b1, c1,h) + S(a2,b2,c2,h)) / 2.0;
25     if(0.1*abs(L-S(a,b,c,h)) < E)
26         return L;
27     else
28         return SIMP(a1,b1, E/2.0) + SIMP(a2,b2, E/2.0);
29 }
```

7.12. Inverso modular

Si queremos hallar $a/b(mod\, p)$ donde p es $primo$, es lo mismo que:
$(a * b^{-1})(mod\, p)$
donde b^{-1} es el inverso modular de b .
Para hallar el inverso modular de un numero $x$modulo p:
$x^{-1}(mod\, p) = modpow(x, p-2, p)$
Entonces, volviendo al problema:
$a/b(mod\, p) = (a * modpow(b, p-2, p)$

Capítulo 8

Cadenas

8.1. Utilidades

```
#include <iostream>
#include <algorithm>

using namespace std;

bool is_palindrome(string &ss){
    return equal(ss.begin(), ss.end(), ss.rbegin());
}

int main(int argc, char const *argv[])
{

    return 0;
}
```

8.2. Boyer Moore

Encuentra todos los match de un patron en el texto.

```
#include <iostream>
#include <cstring>

using namespace std;

int M, N;

//M tamanio del patron
//N tamanio del texto

void preBM(char P[], int bmNext[])
{
    for(int i = 0; i <= 255; i++)
        bmNext[i] = M;
    for(int i = 0; i < M; i++)
        bmNext[P[i]] = M - 1 - i ;
}

void Boyer_Moore(char T[], char P[])
{

```

```
22      int i = M - 1;
23      int j = M - 1;
24      int bmNext[255];
25      preBM(P,bmNext);
26      while((i < N) && (j >= 0))
27      {
28          if(T[i] == P[j])
29          {
30              i--;
31              j--;
32          }
33          else
34          {
35              i += bmNext[T[i]];
36              j = M - 1;
37          }
38          if(j < 0)
39          {
40              cout<<"Match en: "<<(i + 1)<<endl;
41              i += M + 1;
42              j = M - 1;
43          }
44      }
45  }
46
47  int main()
48  {
49      char texto[100];
50      char patron[100];
51      cin>>texto;
52      cin>>patron;
53      M=strlen(patron);
54      N=strlen(texto);
55      Boyer_Moore(texto,patron);
56      return 0;
57  }
```

8.3. Knuth Morris Pratt

```
1   //Entrada: Una cadena S y el patron k
2   //Salida: El numero de ocurrencia del patron k en S
3   int KMP(string S, string K)
4   {
5       vector<int> T(K.size() + 1, -1);
6       for(int i = 1; i <= K.size(); i++)
7       {
8           int pos = T[i - 1];
9           while(pos != -1 && K[pos] != K[i - 1]) pos = T[pos];
10          T[i] = pos + 1;
11      }
12      vector<int> matches;
13      int sp = 0;
14      int kp = 0;
15      while(sp < S.size())
16      {
17          while(kp != -1 && (kp == K.size() || K[kp] != S[sp])) kp = T[kp];
18          kp++;
```

```
        sp++;
        if(kp == K.size()) matches.push_back(sp - K.size());
    }
    /* En el vector matches se guardan los Ãndices a cada una de las
       ocurrencias, por tanto el tamanio de dicho vector nos dirÃ¡ cuÃ¡ntas
       ocurrencias se han encontrado. */
    return matches.size();
}
```

8.4. Iesima permutación

```
//Entrada: Una cadena cad(std::string), un long th
//Salida : La th-esima permutacion lexicografica de cad
string ipermutacion(string cad, long long int th){
    sort(cad.begin(), cad.end());
    string sol = "";
    int pos;
    for(int c = cad.size() - 1; c >= 0; c--){
        pos = th / fact[c];
        th %= fact[c];
        sol += cad[pos];
        cad.erase(cad.begin() + pos);
    }
    return sol;
}
```

8.5. Algoritmo de Manacher

El algoritmo guarda el tamaño del palindrome mas grande en LP[pos] que tiene como centro las posición *pos* en la cadena original

```
#include <iostream>
#include <vector>
#include <fstream>
#include <string.h>
#define MAXN 20100

using namespace std;
int N,i,j,izq,der,posi,dist;
string cad,tmp;
char v[MAXN];
int pos[MAXN];

void todoslosPalindromos(vector<int> &LP, int par){
    izq = der = -1;
    for(posi = 0; posi < N; posi++){
        dist = 0;
        if(posi <= der)
            dist = min(der - posi + par, LP[izq+der-(posi+par)]) + 1;
        while(posi-dist>=0 && posi+dist-par<N
                && v[posi-dist]==v[posi+dist-par])
            dist++;
        LP[posi] = --dist;
        if(posi+dist-par > der){
            der = posi + dist - 1;
```

```
            izq = posi - dist ;
        }
    }

}
int main() {
    getline(fin,cad);
    while(getline(fin,tmp))
        cad +='\n'+tmp;
    N = 0;
    for(i = 0; i < cad.length(); i++)
        if(isalpha(cad[i])){
            v[N] = tolower(cad[i]);
            pos[N] = i;
            N++;
        }
    v[N] = '\0';
    vector<int> LP(N,0),LI(N,0);
    todoslosPalindromos(LP,1);
    todoslosPalindromos(LI,0);
    int SOL1 = 0,SOL2 = 0;
    //primero en los de tamanio impar
    for(int i = 0; i < N; i++)
        if(LI[i]*2+1 > LI[SOL1]*2+1)
            SOL1 = i;
    //luego en los de tamanio par
    for(int i = 0; i < N; i++)
        if(LP[i]*2 > LP[SOL2]*2)
            SOL2 = i;

    if(LI[SOL1]*2+1 > LP[SOL2]*2){
        izq = SOL1 - LI[SOL1];
        der = SOL1 + LI[SOL1];
    }else{
        izq = SOL2 - LP[SOL2];
        der = SOL2 + LP[SOL2]-1;
    }

    fout << der - izq + 1 <<   endl;

    izq = pos[izq];
    der = pos[der];
    fout << cad.substr(izq,der-izq+1)<<endl;
    return 0;
}
```

8.6. Trie

```
#include <iostream>
#include <vector>
using namespace std;
struct Node{
    char content;
    bool wordMarker;
    vector<Node*> children;
    Node() { content = ' '; wordMarker = false; }
    void setContent(char c) { content = c; }
```

```
10      void setWordMarker() { wordMarker = true; }
11      Node* findChild(char c);
12      void appendChild(Node* child) { children.push_back(child); }
13  };
14
15
16  struct Trie {
17      Node* root;
18      Trie(){root=new Node();}
19      void addWord(string s);
20      bool searchWord(string s);
21      void deleteWord(string s);
22  };
23
24  Node* Node::findChild(char c){
25      for ( int i = 0; i < children.size(); i++ ){
26          Node* tmp = children.at(i);
27          if ( tmp->content == c )
28          {
29              return tmp;
30          }
31      }
32
33      return NULL;
34  }
35
36
37  void Trie::addWord(string s){
38      Node* current = root;
39
40      if ( s.length() == 0 ){
41          current->setWordMarker();
42          return;
43      }
44
45      for ( int i = 0; i < s.length(); i++ ){
46          Node* child = current->findChild(s[i]);
47          if ( child != NULL ){
48              current = child;
49          }else{
50              Node* tmp = new Node();
51              tmp->setContent(s[i]);
52              current->appendChild(tmp);
53              current = tmp;
54          }
55          if ( i == s.length() - 1 )
56              current->setWordMarker();
57      }
58  }
59
60
61  bool Trie::searchWord(string s)
62  {
63      Node* current = root;
64
65      while ( current != NULL )
66      {
67          for ( int i = 0; i < s.length(); i++ )
```

```
68          {
69              Node* tmp = current->findChild(s[i]);
70              if ( tmp == NULL )
71                  return false;
72              current = tmp;
73          }
74
75          if ( current->wordMarker )
76              return true;
77          else
78              return false;
79      }
80
81      return false;
82  }
83
84
85  int main(){
86      Trie* trie = new Trie();
87      trie->addWord("algo");
88      trie->addWord("ala");
89      trie->addWord("abeja");
90      trie->addWord("abel");
91      trie->addWord("trie");
92
93      if ( trie->searchWord("abel") )
94          cout << "Encontrado" << endl;
95      else cout << "No encontrado" << endl;
96
97
98      delete trie;
99  }
```

Capítulo 9

Geometria Computacional

9.1. Intersección de rectangulos

```
#include <iostream>
#include <vector>
#include <fstream>

using namespace std;
#define MAXC 2501

struct Rect{
    int x1,y1, x2,y2;
    int color;
    int area;
    Rect(int _x1, int _y1, int _x2, int _y2){
        x1 = _x1;
        y1 = _y1;
        x2 = _x2;
        y2 = _y2;
        getArea();
    }
    int getArea(){
        if(x1>=x2 || y1>=y2)return area = 0;
        return area = (x2-x1)*(y2-y1);
    }
};

Rect interseccion(Rect t, Rect r){
    int x1,y1,x2,y2;
    x1 = max(t.x1,r.x1);
    y1 = max(t.y1,r.y1);
    x2 = min(t.x2,r.x2);
    y2 = min(t.y2,r.y2);
    Rect res(x1,y1,x2,y2);
    return res;
}
```

9.2. Distancia Punto - Segmento

```
struct point{
    double x,y;
};
```

```
4  inline double dist(const point &a, const point &b){
5      return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
6  }
7  inline double distsqr(const point &a, const point &b){
8      return (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y);
9  }
10 double distance_point_to_segment(const point &a, const point &b, const point &
       pnt){
11     double u = ((pnt.x - a.x)*(b.x - a.x) + (pnt.y - a.y)*(b.y - a.y)) / distsqr
           (a, b);
12     point intersection;
13     intersection.x = a.x + u*(b.x - a.x);
14     intersection.y = a.y + u*(b.y - a.y);
15
16     if (u < 0.0 || u > 1.0)
17         return min(dist(a, pnt), dist(b, pnt));
18
19     return dist(pnt, intersection);
20 }
```

9.3. Distancia Punto - Recta

```
1  double distance_point_to_line(const point &a, const point &b, const point &pnt){
2      double u = ((pnt.x - a.x)*(b.x - a.x) + (pnt.y - a.y)*(b.y - a.y)) / distsqr
           (a, b);
3      point intersection;
4      intersection.x = a.x + u*(b.x - a.x);
5      intersection.y = a.y + u*(b.y - a.y);
6      return dist(pnt, intersection);
7  }
```

Capítulo 10

Utilitarios

10.1. Plantilla

```
#include <algorithm>
#include <iostream>
#include <iterator>
#include <cassert>
#include <sstream>
#include <fstream>
#include <cstdlib>
#include <cstring>
#include <utility>
#include <complex>
#include <string>
#include <cctype>
#include <cstdio>
#include <vector>
#include <bitset>
#include <stack>
#include <queue>
#include <cmath>
#include <deque>
#include <list>
#include <set>
#include <map>

#define ll long long
#define lld long long int
#define sc scanf
#define pf printf
#define pi 2*acos(0.0)
#define f first
#define s second
#define sz size()
#define mp make_pair
#define r(input) freopen("2966.in","r",stdin)
#define w(output) freopen("output.txt","w",stdout)
#define maxall(v) *max_element(v.begin(),v.end())
#define minall(v) *min_element(v.begin(),v.end())
#define Sort(v) sort(v.begin(),v.end())
#define un(v) Sort(v), v.erase(unique(v.begin(),v.end()),v.end())
#define clr(a)  memset(a,0,sizeof(a))
#define pb push_back
#define lim 100000
```

10.2. Lectura rapida

```cpp
#include <cstdio>

using namespace std;

char line[400000];//OJO maximo mas 3 para el '\n' y el '\0'
int now = 0;
inline int getInt(){
    int n;
    while(1)
        if(line[now]!=0){
            if(line[now]<'0' || line[now]>'9'){
                now++;
                continue;
            }
            n = 0;
            while(line[now]>='0' && line[now]<='9') {
                n = n*10 + line[now] - '0';
                now++;
            }
            return n;
        }
        else{
            gets(line);
            now = 0;
        }
    return n;
}
```

10.3. Espicificadores de formato para printf y scanf

Especificador	Tipo
%c	char
%s	cadena
%d, %i	int
%u	unsigned int
%ld	long int
%lu	unsigned long int
%lld	long long int
%llu	unsigned long long int
%e, %f, %g	float
%lf	double
%o	número octal
%x	número hexadecimal
%patron	patron

10.4. Contar bits en 1 en un numero

C++ :

```cpp
__builtin_popcount(unsigned int)
__builtin_popcountll(unsigned long long)
```

Java :

```
1 Integer.bitCount(int)
```

10.5. Busqueda binaria

C++'s Standard Template Library has four functions for binary search, depending on what information you want to get. They all need

```
#include <algorithm>
```

The **lower_bound()** function returns an iterator to the first position where a value could be inserted without violating the order; i.e. the first element equal to the element you want, or the place where it would be inserted.

```
int *ptr = std::lower_bound(array, array+len, what);
// a custom comparator can be given as fourth arg
```

The **upper_bound()** function returns an iterator to the last position where a value could be inserted without violating the order; i.e. one past the last element equal to the element you want, or the place where it would be inserted.

```
int *ptr = std::upper_bound(array, array+len, what);
// a custom comparator can be given as fourth arg
```

The **equal_range()** function returns a pair of the results of lower_bound() and upper_bound().

```
std::pair<int *, int *> bounds = std::equal_range(array, array+len, what);
// a custom comparator can be given as fourth arg
```

Note that the difference between the bounds is the number of elements equal to the element you want.

The **binary_search()** function returns true or false for whether an element equal to the one you want exists in the array. It does not give you any information as to where it is.

```
bool found = std::binary_search(array, array+len, what);
// a custom comparator can be given as fourth arg
```