

Índice general

| | |
|--|-----------|
| 1. Ad-hoc | 3 |
| 1.1. Invertir Bytes | 3 |
| 1.2. Año bisiesto | 3 |
| 1.3. Doomsday | 3 |
| 1.4. Hanoi | 3 |
| 2. Estructuras | 4 |
| 2.1. Union - Find | 4 |
| 2.2. BIT | 5 |
| 2.3. Segment Tree | 5 |
| 2.4. Segment Tree con Lazy Propagación | 7 |
| 2.5. Trie | 8 |
| 2.6. Arbol Binario de Búsqueda | 10 |
| 3. Complete Search | 11 |
| 3.1. El problema de las ocho reinas | 11 |
| 3.2. El problema de las ocho reinas acelerado con Bits | 12 |
| 4. Divide y vencerás | 13 |
| 4.1. Búsqueda Binaria (Recursivo) | 13 |
| 4.2. Búsqueda Binaria (Iterativo) | 13 |
| 4.3. MergeSort | 14 |
| 4.4. Meet in the Middle | 15 |
| 5. Programación Dinamica | 17 |
| 5.1. El problema de la mochila y Subset Sum | 17 |
| 5.2. El problema de la mochila(iterativo) | 18 |
| 5.3. Cambio de modenas | 19 |
| 5.4. LIS ($O(n^2)$) | 20 |
| 5.5. LIS ($O(n \log(n))$) | 20 |
| 5.6. Suma maxima en rango | 21 |
| 5.7. Suma maxima en rango 2D $O(n^4)$ | 21 |
| 5.8. Suma maxima en rango 2D $O(n^3)$ | 22 |
| 5.9. Maxima submatriz de ceros | 22 |
| 5.10. Submatriz de suma maxima | 23 |
| 5.11. Distancia de edición (Algoritmo de Levenshtein) | 24 |
| 5.12. Distancia de edición (Recursivo) | 25 |
| 6. Grafos | 26 |
| 6.1. BFS | 26 |
| 6.2. DFS | 26 |
| 6.3. Topological Sort | 27 |
| 6.4. Dijkstra | 27 |
| 6.5. BellmandFord | 28 |
| 6.6. Floyd Warshall | 29 |
| 6.7. Componentes Fuertemente Conexas | 30 |
| 6.8. Componentes Fuertemente Conexas (Kosaraju) | 31 |

| | |
|---|-----------|
| 6.9. Componentes Fuertemente Conexas (Tarjan $O(n + v)$) | 32 |
| 6.10. Minimum Spanning Tree (Kruskall) | 34 |
| 6.11. Second Minimum Spanning Tree | 35 |
| 6.12. Minimum Cut | 37 |
| 6.13. MaxFlow | 38 |
| 6.14. Lowest common ancestor (LCA) | 39 |
| 6.15. Maximo Emparejamiento Bipartito | 40 |
| 6.16. Puentes | 42 |
| 6.17. Puntos de Articulación | 43 |
| 6.18. Camino Euleriano | 44 |
| 6.19. Dijkstra Fast | 45 |
| 7. Matemáticas | 46 |
| 7.1. GCD | 46 |
| 7.2. LCM | 46 |
| 7.3. Algoritmo extendido de euclides | 46 |
| 7.4. Exponenciación rápida | 47 |
| 7.5. Criba de Erathostenes | 47 |
| 7.6. Criba de Erathostenes Aumentada | 48 |
| 7.7. Triangulo de Pascal | 48 |
| 7.8. Combinaciones(Para numeros muy grandes) | 48 |
| 7.9. Polinomios | 49 |
| 7.10. Fibonacci ($O(\log(n))$) | 50 |
| 7.11. Multiplicación entero por cadena | 50 |
| 7.12. Multiplicación de numeros grandes (Karatsuba) | 51 |
| 7.13. Integracion de Simpson | 51 |
| 7.14. Inverso modular | 52 |
| 7.15. El juego de Nim | 52 |
| 7.16. Fraccion | 52 |
| 7.17. Matriz | 54 |
| 7.18. Gauss Jordan | 55 |
| 7.19. Numeros romanos | 58 |
| 8. Cadenas | 59 |
| 8.1. Utilidades | 59 |
| 8.2. Boyer Moore | 59 |
| 8.3. Rabin Karp | 59 |
| 8.4. Knuth Morris Pratt | 60 |
| 8.5. Iesima permutación | 60 |
| 8.6. Algoritmo de Manacher | 61 |
| 8.7. Longest Common Subsequence (LCS) | 63 |
| 8.8. Suffix Array | 63 |
| 8.9. Suffix Array DC3 | 65 |
| 8.10. Minima Rotacion Lexicografica | 67 |
| 8.11. Trie | 68 |
| 8.12. Aplicaciones | 69 |
| 9. Geometria Computacional | 71 |
| 9.1. Intersección de rectangulos | 71 |
| 9.2. Distancia Punto - Segmento | 72 |
| 9.3. Distancia Punto - Recta | 72 |
| 10. Utilitarios | 73 |
| 10.1. Plantilla | 73 |
| 10.2. Lectura rapida | 73 |
| 10.3. Espificadores de formato para printf y scanf | 74 |
| 10.4. Contar bits en 1 en un numero | 74 |
| 10.5. Busqueda binaria | 74 |

Capítulo 1

Ad-hoc

1.1. Invertir Bytes

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

1.2. Año bisiesto

```
1 bool esBisiesto(int y) {
2     return ( ((y%4)==0 && (y%100)!=0) || (y%400)==0 );
3 }
```

1.3. Doomsday

```
1 //Entrada: Una fecha (ene = 1, ..., dic = 12) Salida: Dia de la semana
2 string vec[7] = {"DOM", "LUN", "MAR", "MIE", "JUE", "VIE", "SAB"};
3 string doomsday(int dia, int mes, int anio){
4     int a, y, m, d;
5     a = (14 - mes) / 12;
6     y = anio - a;
7     m = mes + 12 * a - 2;
8     d = (dia + y + y/4 - y/100 + y/400 + (31*m)/12) % 7;
9     return vec[d];
10 }
```

1.4. Hanoi

```
1 int hanoi(int n){
2     if(n==1) return 1;
3     else return 2 * hanoi(n-1) + 1;
4 }
```

Capítulo 2

Estructuras

2.1. Union - Find

```
1 struct UnionFind{
2     int p[MAX], r[MAX], setSize[MAX];
3     int n, numSets;
4     UnionFind(int _N){
5         reset(_N);
6     }
7     void reset(int N){
8         n = N;
9         numSets = n;
10        for(int i = 0; i <= n; i++){
11            p[i] = i;
12            r[i] = 0;
13            setSize[i] = 1;
14        }
15    }
16    int Find(int i){return (p[i] == i)?i:(p[i] = Find(p[i]));}
17    bool sameComponent(int i, int j){return Find(i) == Find(j);}
18    void Union(int i, int j){
19        if(!sameComponent(i, j)){
20            numSets--;
21            int x = Find(i), y = Find(j);
22            if(r[x] > r[y]){
23                p[y] = x;
24                setSize[x] += setSize[y];
25            } else {
26                p[x] = y;
27                setSize[y] += setSize[x];
28                if (r[x] == r[y]) r[y]++;
29            }
30        }
31    }
32    int numDisjointSets(){
33        return numSets;
34    }
35    int sizeOfSet(int i){
36        return setSize[Find(i)];
37    }
38 };
```

2.2. BIT

```

1  #define MAXN 1000000
2
3  int T[MAXN + 1];
4  int A[MAXN];
5  int N;
6  int lowbit(int i) {
7      return (i & -i);
8  }
9  int sum(int i) {
10     int value = 0;
11     for(; i > 0; i -= lowbit(i))
12         value += T[i];
13     return value;
14 }
15 int sum(int i, int j) {
16     return i > 1 ? sum(j) - sum(i-1) : sum(j);
17 }
18 void update(int i, int value) {
19     for(; i <= N ; i += lowbit(i))
20         T[i] += value;
21 }
22 void build() {
23     memset(T, 0, sizeof(T));
24     for(int i=0; i < N; i++)
25         update(i+1, A[i]);
26 }
27 int main() {
28     cin >> N;
29     for(int i = 0; i < N; ++i)
30         cin >> A[i];
31
32     build();
33     cout << sum( 1, N ) << endl;
34     return 0;
35 }

```

2.3. Segment Tree

```

1  int c[1000000];
2  int tree[4000001];
3  //si es segment tree de -, * o / solo sustituir el + en init query y update
4  void init(int node, int a, int b)
5  {
6      if(a==b)
7      {
8          tree[node]=c[a];
9          return ;
10     }
11     init(2*node+1, a, (a+b)/2);
12     init(2*node+2, (a+b)/2+1, b);
13     tree[node]=tree[2*node+1]+tree[2*node+2];
14 }
15 // consula para llamar query(0,0,n-1,desde,hasta)
16 int query(int node, int a, int b, int p, int q)

```

```

17 {
18     //la consulta se hace en el rango desde p a q, a y b son los limites del
        rango
19     if( q<a || b<p ) return 0;
20     if(p<=a && b<=q)
21     {
22         return tree[node];
23     }
24     return query(2*node+1,a,(a+b)/2,p,q)+query(2*node+2,(a+b)/2+1,b,p,q);
25 }
26 //sustituir para llamar(0,0,n-1,posicion,valor)
27 void update(int node,int a,int b,int p,int val)
28 {
29     if(p<a || b<p) return;
30     if(a==b)
31     {
32         tree[node]=val;
33         return ;
34     }
35     update(2*node+1,a,(a+b)/2,p,val);
36     update(2*node+2,(a+b)/2+1,b,p,val);
37     tree[node]=tree[2*node+1]+tree[2*node+2];
38 }
39
40 int main()
41 {
42     int n,aux;
43     for(int i=0; i<n; i++)
44     {
45         scanf("%d",&aux);
46         c[i]=aux;
47     }
48     init(0,0,n-1);
49     //ejemplo de sustitucion tree[a]=val
50     int a,b;
51     int val;
52     scanf("%d %d",&a,&val);
53     a--; //solo si los subindices del problema van de 1...n
54     update(0,0,n-1,a,val);
55     //ejemplo de consulta x=SUM(a,a+1,...,b)
56     scanf("%d %d",&a,&b);
57     a--; //solo si los subindices del problema van de 1...n
58     b--; //solo si los subindices del problema van de 1...n
59     int x=query(0,0,n-1,a,b);
60     printf("%d\n",x);
61     printf("\n");
62     return 0;
63 }

```

Otra implementacion en java:

```

1 public class SegmentTree{
2
3     static final int NEUT = 2147483647; // Cambiar segun operacion
4
5     public static void main(String[] args){
6         int[] m = {4,0,7,1,3,5,8,8,3,2,2,1};
7         int[] rmq = new int[m.length*4];
8         rmq_init(1, 0, m.length, rmq, m);

```

```

9      System.out.println(rmq_query(1, 0, m.length, rmq, 2, 5));
10  }
11  public static int LEFT(int n){ return 2*n; }
12  public static int RIGHT(int n){ return 2*n+1; }
13  public static int oper(int a, int b){ return Math.min(a, b); }
14
15  public static void rmq_init(int n, int s, int e, int[] rmq, int[] m){
16      if(s+1 == e)
17          rmq[n] = m[s];
18      else{
19          rmq_init(LEFT(n), s, (s+e)/2, rmq, m);
20          rmq_init(RIGHT(n), (s+e)/2, e, rmq, m);
21          rmq[n] = oper(rmq[LEFT(n)], rmq[RIGHT(n)]);
22      }
23  }
24  public static void rmq_update(int n, int s, int e, int[] rmq, int[] m, int p
    , int v){
25      if(s+1 == e)
26          rmq[n] = m[s] = v;
27      else{
28          if(p < (s+e)/2)
29              rmq_update(LEFT(n), s, (s+2)/2, rmq, m, p, v);
30          else
31              rmq_update(RIGHT(n), (s+e)/2, e, rmq, m, p, v);
32          rmq[n] = oper(rmq[LEFT(n)], rmq[RIGHT(n)]);
33      }
34  }
35  public static int rmq_query(int n, int s, int e, int[] rmq, int a, int b){
36      if(a >= e || b <= s)
37          return NEUT;
38      else if (s >= a && e <= b)
39          return rmq[n];
40      else{
41          int l = rmq_query(LEFT(n), s, (s+e)/2, rmq, a, b);
42          int r = rmq_query(RIGHT(n), (s+e)/2, e, rmq, a, b);
43          return oper(l, r);
44      }
45  }
46  }

```

2.4. Segment Tree con Lazy Propagación

```

1  struct Node{
2      long long sum; //suma de hijos
3      long long offset;//suma que no propaga de total nodo
4  }Tree[500000];
5
6  void Update(long long node,long long lo,long long hi,long long i,long long j,
    long long val){
7      if(lo>j || hi<i)
8          return;
9
10     if(lo>=i && hi<=j){
11         Tree[node].sum+=(hi-lo+1)*val;
12         Tree[node].offset+=val;
13     }else{
14         long long mid=(lo+hi)>>1;

```

```

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

```

2.5. Trie

```

1 struct Trie{
2     Trie* hijos[26];
3     bool esfin;
4     vector<int> idhijos;
5     Trie(){
6         esfin = false;
7         for(int i = 0; i < 26; ++i)
8             hijos[i] = NULL;
9     }
10 };
11 void insertar(Trie* v, char* c){
12     if(c[0] == '\0')
13         v->esfin = true;
14     else{
15         int k = c[0] - 'a';
16         if(v->hijos[k] == NULL){
17             v->hijos[k] = new Trie();
18             (v->idhijos).push_back(k);
19         }

```



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

```
78     printf("Case # %d:\n", caso);
79     mostrar(t, cad, cadena);
80     if(!haysol)
81         puts("No match.");
82 }
83 return 0;
84 }
```

2.6. Arbol Binario de Busqueda

```
1  struct nodo{
2      int valor;
3      nodo* izq;
4      nodo* der;
5      nodo(){
6          valor = (1<<30);
7          izq = NULL;
8          der = NULL;
9      }
10     nodo(int _v){
11         valor = _v;
12         izq = NULL;
13         der = NULL;
14     }
15 };
16
17 void insertar(int val, nodo **n){
18     if((*n) == NULL)
19         *n = new nodo(val);
20     else
21         if(val > (*n)->valor)
22             insertar(val, &((*n)->der));
23         else
24             insertar(val, &((*n)->izq));
25 }
26
27 void inorden(nodo *n){
28     if(n != NULL){
29         inorden(n->izq);
30         cout << (n->valor) << " ";
31         inorden(n->der);
32     }
33 }
34 int main(){
35     nodo *raiz;
36     raiz = NULL;
37     int nro,tmp;
38     cin >> nro;
39     while(nro--){
40         cin >> tmp;
41         insertar(tmp, &raiz);
42     }
43     cout << "IN-ORDEN" << endl;
44     inorden(raiz);
45     return 0;
46 }
```

Capítulo 3

Complete Search

3.1. El problema de las ocho reinas

```
1  int nrosol, f, c;
2  int sol[10];
3  bool sePuede(int fila, int col){
4      if(col != c && fila == f) return false;
5      for(int i = col - 1 ; i >= 1; i--)
6          if(sol[i] == fila || (abs(sol[i] - fila) == abs(i - col)))
7              return false;
8      return true;
9  }
10 void solve(int col){
11     if(col == 9){
12         if(sol[c] == f){
13             printf(" %2d      %d", nrosol++, sol[1]);
14             for(int i = 2; i <= 8; i++)
15                 printf(" %d", sol[i]);
16             printf("\n");
17         }
18         return ;
19     }
20     for(int i = 1; i <= 8; i++)
21         if(sePuede(i, col)){
22             sol[col] = i;
23             solve(col + 1);
24         }
25 }
26 int main(){
27     int T; cin >> T;
28     while(T--){
29         printf( "SOLN      COLUMN\n #      1 2 3 4 5 6 7 8\n\n" );
30         cin >> f >> c;
31         scanf("%d %d", &f, &c);
32         nrosol = 1;
33         memset(sol, 0, sizeof(sol));
34         sol[c] = f;
35         solve(1);
36     }
37     return 0;
38 }
```

3.2. El problema de las ocho reinas acelerado con Bits

```
1  class Solution {
2  public:
3      int ans, lim;
4
5      void solve(int row, int ld, int rd) {
6          if (row == lim) { // have placed row queens
7              ans++; // we have a solution
8              return;
9          }
10         int pos = lim & ~(row | ld | rd); // valid positions
11         while (pos != 0) {
12             int p = pos & (-pos); // rightmost position
13             pos -= p; // have tried this
14             solve(row + p, (ld + p) << 1, (rd + p) >> 1);
15         }
16     }
17
18     int totalNQueens(int n) {
19         ans = 0;
20         lim = (1 << n) - 1;
21         solve(0, 0, 0);
22         return ans;
23     }
24 };
```

Capítulo 4

Divide y vencerás

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

4.2. Búsqueda 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    return -1; //indica que no esta
15 }
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 ($1 \leq N \leq 34$) 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.

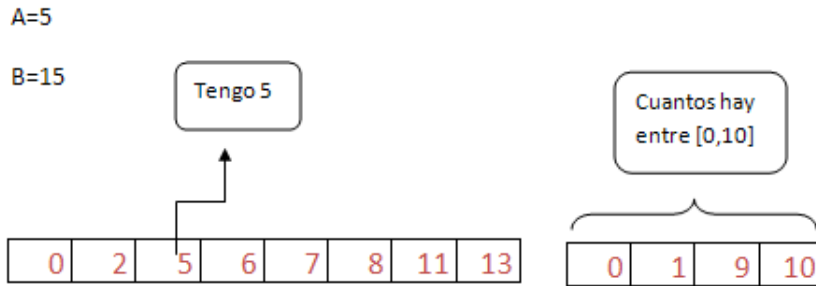
$X' = \{ 0, 2, 5, 6, 7, 8, 11, 13 \}$

$Y' = \{ 0, 1, 9, 10 \}$

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

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

```
1  /*
2  Nota:
3  Si los pesos son muy grandes o el valor de llenar la mochila es muy grande
4  se puede utilizar un map en vez de una matriz para la memoizacion
5  */
6  #include <cstdio>
7  #include <algorithm>
8  #include <cstring>
9
10 using namespace std;
11
12 long long int lib;
13
14 bool sel[35][1005], sol[35]; // para reconstruccion (*)
15 int cant ; // * cantidad de elementos tomados
16
17 long long int dp[35][1005], v[35], p[35]; // dp[OBJETOS + 1][LIBRE + 1]
18 long long int mochila(int i, int libre) { // Llamar mochila(N-1, libre)
19     if(i < 0) return 0LL;
20     long long int &ans = dp[i][libre];
21     if(ans != -1) return ans;
22     ans = mochila(i-1, libre); // no lo tomo
23     if(p[i] <= libre) { // si cabe en el espacio disponible
24         ans = max(ans, v[i] + mochila(i-1, libre - p[i])); // pruebo si lo tomara
25         if(ans == v[i] + mochila(i-1, libre - p[i])) // si se tomo el objeto (*)
26             sel[i][libre] = true; // marco como tomado (*)
27     }
28     return ans;
29 }
30
31 /* Marca en el vector sol[ELEMENTOS] los elementos que fueron tomados
32 void reconstruir(int i, int libre) {
```

```

33     if(i < 0) return;
34     if(sel[i][libre]) { //si fue seleccionado
35         sol[i] = true; //marco
36         cant++;
37         reconstruir(i-1, libre - p[i]);
38     } else
39         reconstruir(i-1, libre);
40 }
41
42 int main() {
43     int n;
44     scanf("%d %ld", &n, &lib);
45     for(int i = 0; i < n; i++)
46         scanf("%ld %ld", &p[i], &v[i]);
47     memset(dp, -1, sizeof(dp));
48     memset(sel, false, sizeof(sel)); //para la reconstruccion (*)
49     printf("%ld\n", mochila(n-1, lib));
50     // imprimir los elementos
51     memset(sol, false, sizeof(sol));
52     cant = 0;
53     reconstruir(n-1, lib);
54     printf("%d\n", cant);
55     for(int i = 0; i < n; i++)
56         if(sol[i])
57             printf("%ld %ld\n", p[i], v[i]);
58     return 0;
59 }

```

5.2. El problema de la mochila(iterativo)

```

1  /*
2  Nota:
3  -Tomar en cuenta que para calcular la fila i solo se necesita la fila i-1
   entonces,
4  si el problema utiliza demasiada memoria o no entra se puede hacer el mismo
   procedimiento
5  con solo 2 vectores llenando la fila actual con la fila ya calculada
6  y luego actual.swap(anterior), para todas la filas
7
8  - La version recursiva corre mas rapido por que no es necesario llenar toda la
   matriz dp
9  */
10 #include <cstdio>
11 #include <algorithm>
12 #include <cstring>
13
14 using namespace std;
15
16 long long int d[35], w, lib;
17 //Para reconstruccion (*)
18 bool sel[35][1005]; //seleccionados (*)
19 bool sol[35]; //el elemento fue seleccionado (*)
20 int cant ; // * cantidad de elementos tomados (*)
21
22 long long int dp[35][1005], v[35], p[35]; //dp[OBJETOS + 1][LIBRE + 1]
23
24 long long int mochila(int n, int libre) { //Llamar con mochila(N, libre)

```

```

25  memset(sel, false, sizeof(sel));
26  memset(dp, 0, sizeof(dp));
27  for(int i = 1; i <= n; i++)
28      for(int j = 1; j<= libre; j++)
29          if(p[i-1] <= j) {
30              dp[i][j] = max(dp[i-1][j-p[i-1]] + v[i-1], dp[i-1][j]);
31              if(dp[i][j] == dp[i-1][j-p[i-1]] + v[i-1])// se selecciono el elemento ?
32                  (*);
33              sel[i][j] = 1; // marcar (*)
34          } else
35              dp[i][j] = dp[i-1][j];
36
37  return dp[n][libre];
38 }
39 void reconstruir(int n, int libre) {
40     int j = libre;
41     cant = 0;
42     memset(sol, false, sizeof(sol));
43     for(int i = n; i >= 1; i--)
44         if(sel[i][j] == 1) {
45             sol[i-1] = true;
46             cant++;
47             j = j - p[i-1];
48         }
49 }
50 int main() {
51     int n;
52
53     scanf("%d %ld", &n, &lib); //N objetos y LIV espacio de la mochila
54     for(int i = 0; i < n; i++)
55         scanf("%ld %ld", &p[i], &v[i]);
56
57     memset(dp, -1, sizeof(dp));
58     memset(sel, false, sizeof(sel)); //para reconstruccion (*)
59     printf("%ld\n", mochila(n, lib));
60
61     reconstruir(n, lib);
62     printf("%d\n", cant);
63     for(int i = 0; i < n; i++)
64         if(sol[i])
65             printf("%ld %ld\n", p[i], v[i]);
66     return 0;
67 }

```

5.3. Cambio de monedas

```

1  long long int dp[30001];
2  int money[]={1,5,10,25,50};
3  void coinchange(int sum){
4      memset(dp, 0, sizeof(a));
5      dp[0]=1;
6      for(int i=0;i<5;i++)
7          for(int j=money[i];j<=sum;j++)
8              dp[j]+=dp[j-money[i]];
9  }

```

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

```

1
2 #include <cstdio>
3 #include <vector>
4 #include <algorithm>
5
6 using namespace std;
7
8 vector<long long int> v;
9 vector<long long int> ancho;
10 vector<long long int> dp;
11 int t, n;
12 long long int lis(int i) {
13     if(i == 0) return ancho[0]; // (*) si no es pesado 1
14     long long int ans = dp[i];
15     if(ans != -1LL) return ans;
16     ans = ancho[i]; // (*) 1
17     for(int j = 0; j < i; j++)
18         if(v[j] < v[i]) // si no es estrictamente creciente "<="
19             ans = max(ans, ancho[j] + lis(j)); // 1 + lis(j)
20     return dp[i] = ans;
21 }
22 int solve() {
23     (vector<long long int>(n, -1LL)).swap(dp);
24     long long int ans = 0;
25     for(int i = n-1; i >= 0; i--)
26         ans = max(ans, lis(i));
27     return ans;
28 }
29
30 int main() {
31     scanf("%d", &n);
32     (vector<long long int>(n)).swap(v);
33     (vector<long long int>(n)).swap(ancho);
34     for(int i = 0; i < n; i++)
35         scanf("%lld", &v[i]);
36     for(int i = 0; i < n; i++) // si es pesado
37         scanf("%lld", &ancho[i]); //
38     printf("lis = %d\n", solve());
39     return 0;
40 }

```

5.5. LIS ($O(n \log(n))$)

```

1 #include <algorithm>
2 #include <cstdio>
3 #include <stack>
4 using namespace std;
5
6 #define MAX_N 100000
7
8 void imprimir(int end, int a[], int p[]) {
9     int x = end;
10    stack<int> s;
11    for (; p[x] >= 0; x = p[x]) s.push(a[x]);

```

```

12 printf("%d\n", a[x]);
13 for (; !s.empty(); s.pop()) printf("%d\n", s.top());
14 }
15
16 int main() {
17     int A[MAX_N], L[MAX_N], L_id[MAX_N], P[MAX_N];
18     int n, tmp;
19     scanf("%d", &n);
20     for(int i = 0; i < n; i++)
21         scanf("%d", &A[i]);
22
23     int lis = 0, lis_end = 0;
24     for (int i = 0; i < n; i++) {
25         int pos = lower_bound(L, L + lis, A[i]) - L;
26         L[pos] = A[i];
27         L_id[pos] = i; // (*) marcamos que posicion es la que estamos guardando
28         P[i] = pos ? L_id[pos - 1] : -1; // (*) El padre del i esimo elemento
29         if (pos + 1 > lis) {
30             lis = pos + 1;
31             lis_end = i;
32         }
33     }
34     printf("%d\n", lis);
35     imprimir(lis_end, A, P);
36     return 0;
37 }

```

5.6. Suma maxima en rango

```

1 //Algoritmo de kadane
2 //Entrada: Un vector de numero enteros
3 //Salida : La suma maxima de alguno de rangos en O(n)
4 long long int maxrangesum(vector<int> &v) {
5     long long int sum = 0, ans = -(1LL<<60); //la maxima suma podria ser negativa
6     for(int i = 0; i < v.size(); i++) {
7         sum += v[i];
8         ans = max(ans, sum);
9         if(sum < 0) sum = 0;
10    }
11    return ans;
12 }

```

5.7. Suma maxima en rango 2D $O(n^4)$

```

1 #include <cstdio>
2 #include <algorithm>
3 #define MAX 103
4 using namespace std;
5 int v[MAX][MAX] = {0};
6 int main() {
7     int n, tmp;
8     scanf("%d", &n);
9     for(int i = 1; i <= n; i++)
10         for(int j = 1; j <= n; j++) {

```

```

11         scanf("%d", &v[i][j]);
12         v[i][j] += (v[i-1][j] + v[i][j-1] - v[i-1][j-1]);
13     }
14     int ans = -127*100*100;
15     for(int i = 1; i <= n; i++) for(int j = 1; j <= n; j++)
16     for(int k = i; k <= n; k++) for(int l = j; l <= n; l++)
17         ans = max(ans, v[k][l] - v[i-1][l] - v[k][j-1] + v[i-1][j-1]);
18     printf("%d\n", ans);
19     return 0;
20 }

```

5.8. Suma maxima en rango 2D $O(n^3)$

```

1  #include <algorithm>
2  #include <cstdio>
3  using namespace std;
4
5  int n, A[101][101] = {0}, maxSubRect, subRect;
6
7  int main() { // O(n^3) 1D DP + greedy (Kadane) solucion
8      scanf("%d", &n); // La dimension de la matriz
9      for (int i = 1; i <= n; i++)
10         for (int j = 1; j <= n; j++) {
11             scanf("%d", &A[i][j]);
12             A[i][j] += A[i][j - 1]; // vamos acumulando de fila en fila
13         }
14
15     maxSubRect = -127*100*100; // El valor mas bajo para la respuesta
16     for (int l = 1; l <= n; l++)
17         for (int r = l; r <= n; r++) {
18             subRect = 0;
19             for (int row = 1; row <= n; row++) {
20                 // Max 1D Range Sum en las columnas de la fila i
21                 subRect += A[row][r] - A[row][l - 1];
22
23                 // El algoritmo de Kadane en las filas
24                 if (subRect < 0) subRect = 0; // golosamente, reiniciamos si sum < 0
25                 maxSubRect = max(maxSubRect, subRect);
26             }
27         }
28     printf("%d\n", maxSubRect);
29     return 0;
30 }

```

5.9. 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));

```

```

9      for (int i=0; i<n; ++i)
10         for (int j=0; j<m; ++j)
11             cin >> a[i][j];
12
13     int ans = 0;
14     vector<int> d (m, -1);
15     vector<int> dl (m), dr (m);
16     stack<int> st;
17     for (int i=0; i<n; ++i) {
18         for (int j=0; j<m; ++j)
19             if (a[i][j] == 1)
20                 d[j] = i;
21         while (!st.empty()) st.pop();
22         for (int j=0; j<m; ++j) {
23             while (!st.empty() && d[st.top()] <= d[j]) st.pop();
24             dl[j] = st.empty() ? -1 : st.top();
25             st.push (j);
26
27         }
28         while (!st.empty()) st.pop();
29         for (int j=m-1; j>=0; --j) {
30             while (!st.empty() && d[st.top()] <= d[j]) st.pop();
31             dr[j] = st.empty() ? m : st.top();
32             st.push (j);
33         }
34         for (int j=0; j<m; ++j)
35             ans = max (ans, (i - d[j]) * (dr[j] - dl[j] - 1));
36     }
37     cout << ans;
38     return 0;
39 }

```

5.10. Submatriz de suma maxima

```

1 void subMatriz_con_sumaMaxima (int m[n][n]) {
2     for (int i = 1; i <= n; i++) {
3         vector<int> vc(n + 1, 0); //para aplicar DP magico
4         for (int j = i; j <= n; j++) {
5             mn_temp = n * n;
6             for (int k = 1; k <= n; k++)
7                 vc[k] += m[j][k];
8
9             dp[0] = 0;
10            for (int i = 1; i <= n; i++) {
11                dp[i] = max(vc[i], dp[i-1] + vc[i]);
12                if (res < dp[i])
13                    res = dp[i];
14            }
15        }
16    }
17
18    printf("%d\n", res);
19 }

```

5.11. Distancia de edición (Algoritmo de Levenshtein)

```
1  #include <string>
2  #include <vector>
3  #include <algorithm>
4  #include <iostream>
5
6  using namespace std;
7  /*Encuentra la distancia de edicion entre 2 cadenas*/
8  /*
9  El numero minimos de cambios para transformar la cadena s1 en la cadena s2, con
    los siguientes cambios:
10
11 - Elimina una letra de una de las cadenas
12 - Inserta una letra en una de las cadenas
13 - Reemplaza una letra en una de las cadenas
14 */
15 int levenshtein(const string &s1, const string &s2)
16 {
17     int N1 = s1.size();
18     int N2 = s2.size();
19     int i, j;
20     vector<int> T(N2+1);
21
22     for ( i = 0; i <= N2; i++ )
23         T[i] = i;
24
25     for ( i = 0; i < N1; i++ ) {
26         T[0] = i+1;
27         int corner = i;
28         for ( j = 0; j < N2; j++ ) {
29             int upper = T[j+1];
30             if ( s1[i] == s2[j] )
31                 T[j+1] = corner;
32             else
33                 T[j+1] = min(T[j], min(upper, corner)) + 1;
34             corner = upper;
35         }
36     }
37
38     return T[N2];
39 }
40
41 int main() {
42     int casos;
43     cin >> casos;
44     string a, b;
45     while(casos--){
46         cin >> a >> b;
47         if(a.size() > b.size())
48             swap(a,b);
49         cout << levenshtein(a,b) << endl;
50     }
51     return 0;
52 }
```


5.12. Distancia de edición (Recursivo)

```
1  #include <stdio>
2  #include <cstring>
3
4  #define MAXS 2005
5  #define max(a,b) a>b?a:b
6  #define min(a,b) a<b?a:b
7  using namespace std;
8
9  char a[MAXS], b[MAXS];
10 int ta, tb;
11
12 int casos;
13 int dp[MAXS][MAXS];
14
15 int solve(int ta, int tb){
16     if(ta < 0 && tb < 0)
17         return 0;
18     if(ta < 0)
19         return tb + 1;
20     if(tb < 0)
21         return ta + 1;
22
23     int &ans = dp[ta][tb];
24     if(ans == -1){
25         ans = 1<<30;
26
27         if(a[ta] == b[tb] )
28             ans = min(ans, solve(ta - 1, tb - 1));
29         else
30             ans = min(ans, 1 + solve(ta - 1, tb - 1)); // con un solo cambio
31             convierto en lo mismo
32             ans = min(ans, 1 + solve(ta - 1, tb      ));
33             ans = min(ans, 1 + solve(ta      , tb - 1));
34     }
35     return ans;
36 }
37
38 int main(){
39     //freopen("entrada.in", "r", stdin);
40     scanf("%d", &casos);
41     while(casos--){
42         scanf("%s %s", a, b);
43         ta = strlen(a);
44         tb = strlen(b);
45         memset(dp, -1, sizeof(dp));
46         printf("%d\n", solve(ta-1, tb - 1));
47     }
48     return 0;
49 }
```

Capítulo 6

Grafos

6.1. BFS

```
1  #include <queue>
2  #include <vector>
3  #include <cstring>
4
5  #define MAXN 1000
6
7  using namespace std;
8
9  vector<int> grafo[MAXN];
10 int d[MAXN];
11 queue<int> cola;
12
13 void BFS(int ini){
14     memset(d,-1, sizeof d);
15     cola.push(ini);
16     d[ini] = 0;
17     while(!cola.empty()){
18         int act = cola.front();cola.pop();
19         for(int i = 0; i < grafo[act].size(); i++){
20             int ady = grafo[act][i];
21             if(d[ady] == -1){
22                 d[ady] = d[act] + 1;
23                 cola.push(ady);
24             }
25         }
26     }
27 }
```

6.2. DFS

```
1  #define MAX 1001
2  #define pain 64
3  int n;
4  int mat[MAX][MAX];
5  bool visitado[MAX][MAX];
6  int di[]={1,1, 1,-1,-1,-1,0, 0};
7  int dj[]={1,0,-1, 1, 0,-1,1,-1};
8
9  void ff_dfs(int i, int j) {
```

```

10     visitado[i][j] = true;
11     for(int k=0; k<8; k++) {
12         int a = di[k] + i;
13         int b = dj[k] + j;
14         if(a >= 0 && b >= 0 && a < n && b < n && !visitado[a][b]) {
15             if(mat[a][b]==pain) ff_dfs(a,b);
16             else visitado[a][b]=true;
17         }
18     }
19 }

```

6.3. Topological Sort

Grafo Aciclico 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

```

1  using namespace std;
2  vector<int> graph[110];
3  bool visitado[110];
4  vector<int> sol;
5  int n,m;
6  void dfs(int node) {
7      visitado[node] = true;
8      for (int i = 0; i < graph[node].size(); i++) {
9          if (!visitado[graph[node][i]])
10             dfs(graph[node][i]);
11     }
12     sol.push_back(node);
13 }
14 void lim() {
15     for(int i=0; i<=110; i++)
16         graph[i].clear();
17
18     clr(visitado);
19     sol.clear();
20 }
21 int main() {
22     //llenado
23     for (int i = 1; i <= n; i++)
24         if (!visitado[i]) dfs(i);
25     reverse(sol.begin(), sol.end());
26     //printsol
27     return 0;
28 }

```

6.4. Dijkstra

```

1  for(int i = 0; i < nodos; ++i) {
2      d[i] = INF;
3      v[i] = false;
4  }
5  d[ini] = 0;
6  q.insert(make_pair(0, ini));
7  while(!q.empty()) {

```

```

8      act = (*(q.begin())).second; q.erase(q.begin());
9      if(v[act]) continue;
10     v[act] = true;
11     for(int i = 0; i < grafo[act].size(); ++i){
12         ady = grafo[act][i].first;
13         peso = grafo[act][i].second;
14         if(!v[ady] && d[act] + peso < d[ady] ){
15             d[ady] = d[act] + peso;
16             q.insert(make_pair(d[ady], ady));
17         }
18     }
19 }
20 if(d[fin] != INF) cout << d[fin] << endl;
21 else cout << "unreachable" << endl;

```

6.5. BellmandFord

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

```

1  #include <cstdio>
2  #include <algorithm>
3  #include <cstring>
4  #include <vector>
5  #include <fstream>
6
7  #define MAX 205
8  #define oo 1LL<<59
9
10 using namespace std;
11
12 int n, m;
13 long long int t[MAX];
14
15 vector<pair<int, long long int> > g[MAX];
16 bool ciclo[MAX];
17 long long int d[MAX];
18 bool v[MAX];
19 void dfs(int i) {
20     v[i] = true;
21     d[i] = -1; //NO CALCULABLE
22     for(int j = 0; j < g[i].size(); j++)
23         if(!v[g[i][j].first])
24             dfs(g[i][j].first);
25 }
26 void bellmanford(int ini) {
27     //fill(d + 1, d + 1 + n, oo);
28     //-----
29     for(int i = 1; i <= n; i++) d[i] = oo, ciclo[i] = false;
30     d[ini] = 0LL;
31     for(int i = 0; i < n-1; i++)
32         for(int u = 1; u <= n; u++)
33             if(d[u] != oo) // Nos aseguramos que ya existe un camino hasta u
34                 for(int j = 0; j < g[u].size(); j++) {
35                     pair<int, long long int> v = g[u][j];
36                     d[v.first] = min(d[v.first], d[u] + v.second); //por que INF + (
37                                     pesonegativo) es menor que INF
38                 }

```

```

38 //verificamos y marcamos todos los nodos que pertenecen a un ciclo negativo
39 for(int u = 1; u <= n; u++)
40     for(int j = 0; j < g[u].size(); j++) {
41         pair<int,int> v = g[u][j];
42         if(d[v.first] > d[u] + v.second)
43             ciclo[u] = ciclo[v.first] = true; //Ciclo negativo return -1
44     }
45 //-----
46 //Todos los nodos accesibles desde un nodo con ciclo negativo tambien tienen
    distancia NO CALCULABLE
47 memset(v, false, sizeof(v));
48 for(int i = 1; i <= n; i++)
49     if(ciclo[i] && !v[i])
50         dfs(i);
51 }
52 int main() {
53     freopen("in", "r", stdin);
54     int u, v, caso = 1;
55
56     while(scanf("%d", &n) == 1) {
57         for(int i = 1; i <= n; i++) {
58             scanf("%lld", &t[i]);
59             g[i].clear();
60         }
61         scanf("%d", &m);
62         while(m--) {
63             scanf("%d%d", &u, &v);
64             long long int tmp = t[v]-t[u];
65             g[u].push_back(make_pair(v, tmp * tmp * tmp));
66         }
67         bellmanford(1);
68         scanf("%d", &m); //querys
69         printf("Set # %d\n", caso++);
70         while(m--) {
71             scanf("%d", &v);
72             if(d[v] < 3 || d[v] == oo) //No accesible o tiene ciclo negativo
73                 puts("?");
74             else
75                 printf("%lld\n", d[v]); //Distancia correcta
76         }
77     }
78     return 0;
79 }

```

6.6. Floyd Warshall

```

1 int dp[100][100]; //grafo matriz de ady
2
3 void floyd_warshall() {
4     for(int k=1; k<=n; k++)
5         for(int i=1; i<=n; i++)
6             for(int j=1; j<=n; j++)
7                 dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j]);
8 }

```

6.7. Componentes Fuertemente Conexas

```

1  #include <cstring>
2  #include <string>
3  #include <cstdio>
4  #include <cstdlib>
5  #include <vector>
6  #include <algorithm>
7  #include <iostream>
8
9  #define clr(a)  memset(a,0,sizeof(a))
10 #define pb push_back
11 using namespace std;
12 /*
13 Finding Strongly Connected Components
14 */
15 vector<int> grafo[N];
16 vector<int> gaux;
17 bool visitado[N];
18 void dfs(int u) {
19     visitado[u]=true;
20     for(int i=0;i<grafo[u].size();i++) {
21         if(!visitado[grafo[u][i]]) dfs(grafo[u][i]);
22     }
23 }
24 int main() {
25     //r(input);
26     int t;
27     int x,y;
28     int n,m;
29     sc("%d",&t);
30     while(t--){
31         sc("%d %d",&n,&m);
32         clr(visitado);
33         for(int i=0;i<=n;i++) grafo[i].clear();
34         gaux.clear();
35         while(m--){
36             sc("%d %d",&x,&y);
37             grafo[x].pb(y);
38         }
39         int M=0;
40         for(int i=1;i<=n;i++){
41             if(!visitado[i]){
42                 dfs(i),gaux.pb(i);
43             }
44         }
45         clr(visitado);
46         M=0;
47         for(int i=gaux.size()-1;i>=0;i--){
48             if(!visitado[gaux[i]]){
49                 dfs(gaux[i]);
50                 M++;
51             }
52         }
53         cout<<M<<endl;
54     }
55 }

```

6.8. Componentes Fuertemente Conexas (Kosaraju)

```

1  #include <cstdio>
2  #include <vector>
3  #include <cstring>
4  #include <fstream>
5
6  #define MAX 2005
7
8  using namespace std;
9  vector<int> g[MAX], gt[MAX];
10 vector<int> S; //para el topsort
11 int N,M; // nodos, edges
12 int numSCC;
13 bool vis[MAX];
14 int scc[MAX]; // Para asignar scc correspondiente
15 void dfs(int u, int pass) {
16     vis[u] = true;
17     vector<int> &ady = ((pass == 1)?g[u]: gt[u]);
18     for(int j = 0; j < ady.size(); j++) {
19         int v = ady[j];
20         if(!vis[v])
21             dfs(v, pass);
22     }
23     if(pass == 1) S.push_back(u); //adicionamos al topsort
24     else scc[u] = numSCC; // asignamos a que SCC pertenece el nodo
25 }
26
27 int kosaraju() {
28     S.clear();
29     memset(vis, false, sizeof(vis));
30     for(int i = 1; i <= N; i++) //Los nodos estan numerados de 1 - N
31         if(!vis[i])
32             dfs(i, 1);
33     numSCC = 0;
34     memset(vis, false, sizeof(vis));
35     memset(scc, -1, sizeof(scc));
36     for(int i = N-1; i >= 0; i--)
37         if(!vis[S[i]]) {
38             dfs(S[i], 2);
39             numSCC++;
40         }
41     return numSCC;
42 }
43
44 int main(){
45     //freopen("in", "r", stdin);
46     int u, v, sent;
47     while(scanf("%d%d", &N, &M) and !(N and !M)){
48         for(int i = 1; i <= N; i++) g[i].clear(), gt[i].clear();
49         while(M--){
50             scanf("%d%d%d", &u, &v, &sent);
51             if(sent == 1){ //un solo sentido
52                 g[u].push_back(v);
53                 gt[v].push_back(u);
54             } else { //Doble sentido
55                 g[u].push_back(v); g[v].push_back(u);
56                 gt[v].push_back(u); gt[u].push_back(v);
57             }
58         }
59     }

```

```

57     }
58     printf("%d\n", kosaraju()==1);
59 }
60 return 0;
61 }

```

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() {
79     int nodos;
80     cin >> nodos;
81     Grafo g(nodos);
82     //leer lados
83     g.ejecutarTarjan();
84     //g.nComponentes contiene el numero de SCC
85     return 0;
86 }

```

Otra implementacion:

```

1  import java.util.*;
2
3  public class SCCTarjan {
4
5      int time;
6      List<Integer>[] graph;
7      int[] lowlink;
8      boolean[] used;
9      List<Integer> stack;
10     List<List<Integer>> components;
11
12     public List<List<Integer>> scc(List<Integer>[] graph) {
13         int n = graph.length;
14         this.graph = graph;
15         lowlink = new int[n];
16         used = new boolean[n];
17         stack = new ArrayList<>();
18         components = new ArrayList<>();
19

```

```

20     for (int u = 0; u < n; u++)
21         if (!used[u])
22             dfs(u);
23
24     return components;
25 }
26
27 void dfs(int u) {
28     lowlink[u] = time++;
29     used[u] = true;
30     stack.add(u);
31     boolean isComponentRoot = true;
32
33     for (int v : graph[u]) {
34         if (!used[v])
35             dfs(v);
36         if (lowlink[u] > lowlink[v]) {
37             lowlink[u] = lowlink[v];
38             isComponentRoot = false;
39         }
40     }
41
42     if (isComponentRoot) {
43         List<Integer> component = new ArrayList<>();
44         while (true) {
45             int k = stack.remove(stack.size() - 1);
46             component.add(k);
47             lowlink[k] = Integer.MAX_VALUE;
48             if (k == u)
49                 break;
50         }
51         components.add(component);
52     }
53 }
54
55 public static void main(String[] args) {
56     List<Integer>[] g = new List[1000];
57     for (int i = 0; i < g.length; i++) {
58         g[i] = new ArrayList<>();
59     }
60
61     List<List<Integer>> components = new SCCTarjan().scc(g);
62     System.out.println(components);
63 }
64 }

```

6.10. Minimum Spanning Tree (Kruskall)

```

1  #define lim 100000
2  using namespace std;
3  typedef vector<pair<int, pair<int, int>>> > V;
4  int N, lider[lim];
5  V v;
6  void init() {
7      Sort(v);
8      for(int i=0; i<N; i++) lider[i]=i;
9  }

```

```

10 int find(int n) {
11     if (n==lider[n]) return n;
12     else return lider[n]=find(lider[n]);
13 }
14 int kruskall() {
15     int a,b,sum=0;
16     init();
17     for(int i=0; (int) i<v.sz; i++) {
18         a=find(v[i].s.f);
19         b=find(v[i].s.s);
20         if (a!=b) {
21             lider[b]=a;
22             sum+=v[i].f;
23         }
24     }
25     return sum;
26 }
27 //v.pb(mp(c,mp(a,b))); peso(c),arista(a,b)

```

6.11. Second Minimum Spanning Tree

Segundo Mejor Arbol de Expansion Complejidad: $O(V \cdot E)$, (V = nodos, E =Arcos)

- Armamos el primer MST
- Para cada arco en el MST el Segundo Mejor Arbol de Expansion es el minimo de:
 - Ignoramos ese arco
 - Hallamos el MST sin ese arco
- No es necesario volver a ordenar por que ya se ordeno al calcular el primer MST

```

1  #include <cstdio>
2  #include <fstream>
3  #include <algorithm>
4  #include <vector>
5  #include <cmath>
6  #include <climits>
7
8  #define MAX 102
9
10 using namespace std;
11 struct UnionFind {
12     int p[MAX];
13     int r[MAX];
14     int n;
15     UnionFind(int _n) {
16         reset(_n);
17     }
18     void reset(int N) {
19         n = N;
20         for(int i = 0 ; i <= n ; ++i )
21             p[i] = i, r[i] = 0;
22     }
23     int Find(int x) {
24         return (p[x]==x?x:p[x]=Find(p[x]));
25     }
26     bool sameComponent(int x, int y) {

```

```

27     return Find(x) == Find(y);
28 }
29 void Union(int x, int y) {
30     int xRoot = Find(x);
31     int yRoot = Find(y);
32     if(r[xRoot] > r[yRoot])
33         p[yRoot] = xRoot;
34     else {
35         p[xRoot] = yRoot;
36         if(r[xRoot] == r[yRoot])
37             r[yRoot]++;
38     }
39 }
40 };
41 vector<pair<long long int, pair<int, int> > > g;
42 vector<int> mst;
43 UnionFind uf(MAX - 1);
44 int t, n, m, u, v;
45 long long int p;
46 long long int solve(int ign, bool guardar) {
47     long long int ans = 0;
48     uf.reset(n);
49     int comp = n;
50     for(int i = 0; i < g.size(); i++) {
51         if(i == ign) continue;
52         int a = g[i].second.first, b = g[i].second.second;
53         long long int p = g[i].first;
54         if(!uf.sameComponent(a, b)) {
55             uf.Union(a, b);
56             ans += p;
57             if(guardar) mst.push_back(i);
58             comp--;
59             if(comp == 1) break;
60         }
61     }
62     if(comp != 1) return LLONG_MAX;
63     return ans;
64 }
65 int main() {
66     freopen("in", "r", stdin);
67     scanf("%d", &t);
68     while(t--) {
69         scanf("%d%d", &n, &m);
70         g.clear();
71         while(m--) {
72             scanf("%d%d%lld", &u, &v, &p);
73             g.push_back(make_pair(p, make_pair(u,v)));
74         }
75         sort(g.begin(), g.end());
76         mst.clear();
77         long long int first = solve(-1, true), second = LLONG_MAX;
78         for(int i = 0; i < mst.size(); i++)
79             second = min(second, solve(mst[i], false));
80         printf("%lld %lld\n", first, second);
81     }
82     return 0;
83 }

```

6.12. Minimum Cut

```

1  #include <iostream>
2  #include <limits.h>
3  #include <string.h>
4  #include <queue>
5  using namespace std;
6
7  #define V 100
8
9  int bfs(int rGraph[V][V], int s, int t, int parent[])
10 {
11     bool visited[V];
12     memset(visited, 0, sizeof(visited));
13     queue<int> q;
14     q.push(s);
15     visited[s] = true;
16     parent[s] = -1;
17     while (!q.empty())
18     {
19         int u = q.front();
20         q.pop();
21         for (int v=0; v<V; v++)
22         {
23             if (visited[v]==false && rGraph[u][v] > 0)
24             {
25                 q.push(v);
26                 parent[v] = u;
27                 visited[v] = true;
28             }
29         }
30     }
31     return (visited[t] == true);
32 }
33 void dfs(int rGraph[V][V], int s, bool visited[])
34 {
35     visited[s] = true;
36     for (int i = 0; i < V; i++)
37         if (rGraph[s][i] && !visited[i])
38             dfs(rGraph, i, visited);
39 }
40
41 void minCut(int graph[V][V], int s, int t)
42 {
43     int u, v;
44     int rGraph[V][V];
45     for (u = 0; u < V; u++)
46         for (v = 0; v < V; v++)
47             rGraph[u][v] = graph[u][v];
48
49     int parent[V];
50     while (bfs(rGraph, s, t, parent))
51     {
52         int path_flow = INT_MAX;
53         for (v=t; v!=s; v=parent[v])
54         {
55             u = parent[v];
56             path_flow = min(path_flow, rGraph[u][v]);

```

```

57     }
58
59     for (v=t; v != s; v=parent[v])
60     {
61         u = parent[v];
62         rGraph[u][v] -= path_flow;
63         rGraph[v][u] += path_flow;
64     }
65 }
66 bool visited[V];
67 memset(visited, false, sizeof(visited));
68 dfs(rGraph, s, visited);
69 int costoCorte=0;
70 for (int i = 0; i < V; i++)
71     for (int j = 0; j < V; j++)
72         if (visited[i] && !visited[j] && graph[i][j]){
73             //cout << i << " - " << j << endl;
74             costoCorte+=graph[i][j];
75         }
76     cout<<costoCorte<<endl;
77     return;
78 }
79
80 int main()
81 {
82     int graph[V][V];
83     minCut(graph, 0, 5);
84     return 0;
85 }

```

6.13. MaxFlow

```

1  #include <algorithm>
2  #include <cstring>
3  #include <cstdio>
4  #include <limits>
5  #include <queue>
6
7  using namespace std;
8
9  #define N 1000 //maximo de nodos
10 //halla el flujo maximo desde el nodo s
11 //hasta el nodo t
12 //cap[][] almacena flujos parciales
13 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     printf("el flujo maximo entre s y t es %d\n", maxFlow());
64
65     return 0;
66 }
67

```

6.14. Lowest common ancestor (LCA)

```

1  #include <cctype>
2  #include <cstdio>
3  #include <iostream>
4  #define NN 100048
5  using namespace std;
6
7  // LCA con pesos
8  // halla camino corto desde a hasta b con numero de consultas q
9
10 int P[NN][17], L[NN];
11 long long W[NN];
12

```

```

13 int query(int p, int q)
14 {
15     int log, i;
16     if (L[p] < L[q]) p ^= q ^= p ^= q;
17     for (log = 1; 1 << log <= L[p]; log++);
18     log--;
19     for (i = log; i >= 0; i--)
20         if (L[p] - (1 << i) >= L[q])
21             p = P[p][i];
22     if (p == q) return p;
23     for (i = log; i >= 0; i--)
24         if (P[p][i] != -1 && P[p][i] != P[q][i])
25             p = P[p][i], q = P[q][i];
26
27     return P[p][0];
28 }
29 int main(void)
30 {
31     int a, b, N, q;
32     cin>>N; //nodos
33     long long w;
34     P[0][0] = -1;
35     L[0] = W[0] = 0;
36     int lados;
37     cin>>lados;
38     int ii; // (ii<=>desde), (a<=> hasta), (w<=>peso)
39     for(int i = 0; i < lados; ++i)
40     {
41         cin>>ii>>a>>w;
42         for(int j = 0; (1 << j) < N; ++j) P[ii][j] = -1;
43         P[ii][0] = a;
44         L[ii] = L[a] + 1;
45         W[ii] = W[a] + w;
46     }
47     for(int j = 1; (1 << j) < N; ++j)
48         for(int i = 0; i < N; ++i)
49             if(P[i][j - 1] != -1)
50                 P[i][j] = P[P[i][j - 1]][j - 1];
51     cin>>q;
52     while(q--)
53     {
54         cin>>a>>b;
55         printf("%lld\n", W[a] + W[b] - (W[query(a, b)] << 1));
56     }
57     return 0;
58 }

```

6.15. Maximo Emparejamiento Bipartito

```

1 #include <cstdio>
2 #include <list>
3 #include <queue>
4
5 //Emparejamiento Bipartito hasta 50000 nodos y 150000 caminos :) crazy...
6
7 using namespace std;
8 //MAX_N y SINK son mayores al numero de nodos

```



```
9  const int SINK = 50001;
10 const int MAX_N = 50005;
11 const int INFINITO = 87654321;
12
13 int N, M, P;
14 int grafo1[MAX_N], grafo2[MAX_N], dist[MAX_N];
15 list< int > adj[MAX_N];
16
17 bool bfs() {
18     queue< int > q; // cola de G1 nodes
19
20     for (int v = 0; v < N; ++v) {
21         if (grafo1[v] == SINK) {
22             dist[v] = 0;
23             q.push(v);
24         }
25         else {
26             dist[v] = INFINITO;
27         }
28     }
29     dist[SINK] = INFINITO;
30
31     while (!q.empty()) {
32         int v = q.front();
33         q.pop();
34
35         if (dist[v] < dist[SINK]) {
36             for (list< int >::iterator it = adj[v].begin(); it != adj[v].end();
37                  ++it) {
38                 int u = *it;
39
40                 if (dist[grafo2[u]] == INFINITO) {
41                     dist[grafo2[u]] = dist[v] + 1;
42                     q.push(grafo2[u]);
43                 }
44             }
45         }
46         return dist[SINK] != INFINITO;
47     }
48
49 bool dfs(int u) {
50     // DFS comienza en el nodo u de G1
51
52     if (u == SINK) {
53         return true;
54     }
55
56     for (list< int >::iterator it = adj[u].begin(); it != adj[u].end(); ++it) {
57         int v = *it;
58
59         if (dist[grafo2[v]] == dist[u] + 1) {
60             if (dfs(grafo2[v])) {
61                 grafo1[u] = v;
62                 grafo2[v] = u;
63                 return true;
64             }
65         }
```

```

66     }
67     dist[u] = INFINITO;
68     return false;
69 }
70
71 int hopcroft_karp() {
72     int matching = 0;
73
74     for (int i = 0; i < N; ++i) {
75         grafo1[i] = SINK;
76     }
77     for (int j = 0; j < M; ++j) {
78         grafo2[j] = SINK;
79     }
80
81     while (bfs()) {
82         for (int i = 0; i < N; ++i) {
83             if (grafo1[i] == SINK) {
84                 matching += dfs(i);
85             }
86         }
87     }
88     return matching;
89 }
90
91 int main() {
92     // inside in main() ggg al estilo halim
93     //N primer conjunto
94     //M segundo conjunto
95     //P numero de relaciones (arcos)
96     scanf("%i %i %i", &N, &M, &P);
97     //clear
98     for (int i = 0; i < N; ++i) {
99         adj[i] = list< int >();
100     }
101
102     for (int i = 0; i < P; ++i) {
103         int u, v;
104
105         scanf("%i %i", &u, &v);
106         --u; --v;
107         adj[u].push_back(v);
108     }
109
110     printf("%i\n", hopcroft_karp());
111
112     return 0;
113 }

```

6.16. Puentes

```

1  #define forn(i, n) for (int i = 0; i < (n); i++)
2  #define MAXN maximonodos
3  vector<int> g[MAXN];
4  vector<pair<int,int> > bridges;
5  int n; //Nodos
6  bool vis[MAXN];

```

```

7  int tin[MAXN];
8  int fup[MAXN];
9  int tiempo;
10 void dfs(int u, int padre = -1) {
11     vis[u] = true;
12     tin[u] = fup[u] = tiempo++;
13     for (int to: g[u]) {
14         if (to == padre) continue;
15         if (vis[to])
16             fup[u] = min(fup[u], tin[to]);
17         else {
18             dfs(to, u);
19             fup[u] = min(fup[u], fup[to]);
20             if (fup[to] > tin[u])
21                 bridges.pb(mp(u, to));
22         }
23     }
24 }
25 void puentes() {
26     bridges.clear();
27     fill(vis, vis + n, false);
28     tiempo = 0;
29     forn (i, n)
30         if (!vis[i])
31             dfs(i);
32     for (auto x: bridges)
33         printf("%d - %d\n", x.first, x.second);
34 }
35 void addEdge(int u, int to) {
36     g[u].pb(to);
37     g[to].pb(u);
38 }
39 int main() {
40     n = 5;
41     addEdge(1, 0);
42     addEdge(0, 2);
43     addEdge(2, 1);
44     addEdge(0, 3);
45     addEdge(3, 4);
46     puentes();
47     return 0;
48 }

```

6.17. Puntos de Articulación

```

1  import java.util.*;
2
3  public class ArticulationPoints {
4
5      static int time;
6      static List<Integer>[] grafo;
7      static boolean[] vis;
8      static int[] tin;
9      static int[] dfsLow;
10     static List<Integer> puntosArticulacion;
11
12     static void dfs(int u, int p) {

```

```

13     vis[u] = true;
14     dfsLow[u] = tin[u] = time++;
15     int child = 0;
16     for (int v : grafo[u]) {
17         if (v == p) {
18             continue;
19         }
20         if (vis[v]) {
21             dfsLow[u] = Math.min(dfsLow[u], tin[v]);
22         } else {
23             dfs(v, u);
24             dfsLow[u] = Math.min(dfsLow[u], dfsLow[v]);
25             if (dfsLow[v] >= tin[u] && p != -1) {
26                 puntosArticulacion.add(u);
27             }
28             ++child;
29         }
30     }
31     if (p == -1 && child > 1) {
32         puntosArticulacion.add(u);
33     }
34 }
35
36 public static void main(String[] args) {
37     time = 0;
38     int n = 5;
39     grafo = new List[n];
40     for (int i = 0; i < n; i++) {
41         grafo[i] = new ArrayList<>();
42     }
43
44     vis = new boolean[n];
45     tin = new int[n];
46     dfsLow = new int[n];
47     puntosArticulacion = new ArrayList<>();
48
49     dfs(0, -1);
50
51     //System.out.println(puntosArticulacion); si puntosArticulacion == null
52     //entonces es un grafo biconexo
53 }

```

6.18. Camino Euleriano

```

1 list<int> cyc;
2 void EulerTour(list<int>::iterator i, int u) {
3     for(int j = 0; j < grafo[u].size(); j++) {
4         pair<int, int> v = grafo[u][j];
5         if(v.second) {
6             v.second = 0;
7             for(int k = 0; k < grafo[v.first].size(); k++) {
8                 pair<int, int> uu = grafo[v.first][k];
9                 if(uu.first == u && uu.second) {
10                     uu.second = 0;
11                     break;
12                 }

```

```

13     }
14     EulerTour(cyc.insert(i, u), v.first);
15 }
16 }
17 }
18 int main() {
19     cyc.clear();
20     EulerTour(cyc.begin(), ini);
21     for(list<int>::iterator it = cyc.begin(); it != cyc.end(); it++)
22         cout << (*it) << endl;
23     return 0;
24 }

```

6.19. Dijkstra Fast

```

1  vector<int> dist(n, INF); dist[S] = 0;
2  queue<int> q; q.push(S);
3  vector<int> in_queue(n, 0); in_queue[S] = 1;
4  while (!q.empty()) {
5      int u = q.front(); q.pop(); in_queue[u] = 0;
6      for (j = 0; j < (int)AdjList[u].size(); j++) {
7          int v = AdjList[u][j].first, weight_u_v = AdjList[u][j].second;
8          if (dist[u] + weight_u_v < dist[v]) {
9              dist[v] = dist[u] + weight_u_v;
10             if (!in_queue[v]) { //Agregamos a la cola solo si no esta en la cola
11                 q.push(v);
12                 in_queue[v] = 1;
13             }
14         }
15     }
16 }
17 if (dist[T] != INF) printf("%d\n", dist[T]);
18 else printf("unreachable\n");

```

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. Algoritmo extendido de euclides

```
1 /*Permite encontrar los coeficientes (x, y) mas pequenos de:
2     ax + by = gcd(a, b);
3 */
4 int x, y; //Al terminar la sol estara aqui
5 void solve(int a, int b) {
```

```

6   if(b == 0){x = 1; y = 0; d = a; return;}
7   solve(b, a % b);
8   int x1 = y;
9   int y1 = x - (a / b) * y;
10  x = x1;
11  y = y1;
12 }

```

7.4. Exponenciación rápida

```

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 }

```

7.5. 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 }

```

7.6. Criba de Erathostenes Aumentada

```

1     scanf("%llu %llu", &a, &b);
2     if(a <= 1) a = 2; //Causa error con en 1 (No es primo)
3     fill(p2, p2 + MAX, true);
4     int ptr = 0;
5     while(primos[ptr] * primos[ptr] <= b) {
6         unsigned long long int ini = a;
7         if(a % primos[ptr] != 0) ini += (primos[ptr] - (a % primos[ptr])); //el
            siguiente multiplo de primos[ptr]
8         if(ini <= primos[ptr]) ini += primos[ptr]; //Si es primo
9         while(ini <= b) {
10             p2[ini - a] = false;
11             ini += primos[ptr];
12         }
13         ptr++;
14     }
15     unsigned long long int ans = 0, primos = 0;
16     for(unsigned long long int i = a; i <= b; i++)
17         if(p2[i - a])
18             primos++;

```

7.7. Triangulo de Pascal

```

1     #define MAXN 100 // largest n or m
2
3     int n,m;
4     long binomial_coefficient(n,m) {
5         int i, j;
6         long bc[MAXN][MAXN];
7         for (i=0; i<=n; i++) bc[i][0] = bc[i][i] = 1;
8         for (i=1; i<=n; i++)
9             for (j=1; j<i; j++)
10                 bc[i][j] = bc[i-1][j-1] + bc[i-1][j];
11         return bc[n][m];
12     }
13 }

```

7.8. Combinaciones(Para numeros muy grandes)

```

1     unsigned long long C(int n, int k) {
2         k = min(k,n-k);
3         unsigned long long res = 1;

```



```
4     for(int i = 0; i < k; i++)
5     {
6         res *= (n-i);
7         res /= (i+1);
8     }
9     return res;
10 }
```

7.9. Polinomios

```
1 class Polinomio {
2     double[] coef;
3
4     // a0 + a1x + a2x^2 + ...
5     public Polinomio(double[] coef) {
6         this.coef = coef;
7     }
8
9     public double integrar(double a, double b) {
10         Polinomio integ = integral();
11         return integ.evaluar(b) - integ.evaluar(a);
12     }
13
14     public double evaluar(double x) {
15         double result = 0;
16         for (int i = coef.length - 1; i >= 0; i--)
17             result = coef[i] + x*result;
18         return result;
19     }
20
21     public Polinomio integral() {
22         double[] newCoef = new double[coef.length + 1];
23         newCoef[0] = 0;
24         for (int i = 1; i < newCoef.length; i++)
25             newCoef[i] = coef[i-1]/i;
26         return new Polinomio(newCoef);
27     }
28
29     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 }
```

7.10. 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 }

```

7.11. 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 }

```

7.12. Multiplicación de numeros grandes (Karatsuba)

```

1 //Entrada: 2 BigInteger's x e y
2 //Salida: La multiplicacion 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 }

```

7.13. 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 }

```

7.14. 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:

$a/b(mod p) = (a * modpow(b, p-2, p))$

7.15. El juego de Nim

```

1  #include <stdio>
2
3  int main() {
4      int N;
5      while (scanf("%d", &N) && N) {
6          int b = 0;
7          while (N-->0) {
8              int v;
9              scanf("%d", &v);
10             b ^= v;
11         }
12         puts(b ? "Yes" : "No");
13     }
14     return 0;
15 }

```

7.16. Fraccion

```

1
2  //////////////////////////////////////////////////
3  //Fraction.
4  //////////////////////////////////////////////////
5  using namespace std;
6
7  typedef long long Num;
8
9  Num gcd(Num a, Num b) {
10     Num temp;
11     if (a < b) {
12         temp = a;
13         a = b;
14         b = temp;

```

```

15     }
16     do {
17         temp = a % b;
18         a = b;
19         b = temp;
20     } while (b != 0);
21     return a;
22 }
23
24 struct Fraction {
25     Num up;
26     Num down;
27
28     Fraction(Num _up = 1, Num _down = 1) {
29         up = _up;
30         down = _down;
31     }
32
33     void reduce() {
34         if (up != 0) {
35             Num div = up > 0? gcd(up, down) : gcd(-up, down);
36             up /= div;
37             down /= div;
38         }
39         else {
40             down = 1;
41         }
42     }
43
44     Fraction operator + (const Fraction& add) const {
45         Fraction result(up * add.down + down * add.up, down * add.down);
46         result.reduce();
47         return result;
48     }
49
50     Fraction operator - (const Fraction& sub) const {
51         Fraction result(up * sub.down - down * sub.up, down * sub.down);
52         result.reduce();
53         return result;
54     }
55
56     Fraction operator * (const Fraction& mul) const {
57         Fraction result(up * mul.up, down * mul.down);
58         result.reduce();
59         return result;
60     }
61
62     Fraction operator / (const Fraction& div) const {
63         Fraction result(up * div.down, down * div.up);
64         result.reduce();
65         return result;
66     }
67
68     bool operator == (Num i) const {
69         return up % down == 0 && up / down == i;
70     }
71
72     bool operator != (Num i) const {

```

```

73         return !(*this == i);
74     }
75
76     void operator = (Num i) {
77         up = i;
78         down = 1;
79     }
80 };
81
82 //Test suite.
83 #include <iostream>
84
85 ostream& operator << (ostream& os, const Fraction& f) {
86     if (f.down != 1) {
87         os << f.up << "/" << f.down;
88     }
89     else {
90         os << f.up;
91     }
92     return os;
93 }
94
95 int main () {
96     typedef Fraction F;
97     Fraction f[] = {F(2, 3), F(5, 7), F(9, 6), F(3, 4), F(125, 63)};
98     Fraction result = (f[0] + f[1] * f[2]) / f[3];
99     cout << result << " ";
100    result = result - f[4];
101    cout << result << " " << (result == 1) << " " << (result != 0) << endl;
102    //Correct output: 146/63 1/3 0 1;
103    return 0;
104 }

```

7.17. Matriz

```

1  #define MOD 1000000007
2  #define MAXN 16
3
4  int size;
5
6  struct Matrix {
7      int X[MAXN][MAXN];
8      Matrix () {}
9      Matrix (int k) {
10         memset(X, 0, sizeof(X));
11
12         for(int i=0; i<size; i++)
13             X[i][i] = k;
14     }
15 };
16
17 Matrix operator *(Matrix &A, Matrix &B) {
18     Matrix M;
19
20     for(int i=0; i<size; i++) {
21         for(int j=0; j<size; j++) {
22             long long tmp = 0;

```

```

23     for(int k=0; k<size; k++)
24         tmp += (long long)A.X[i][k] * B.X[k][j];
25     M.X[i][j] = tmp % MOD;
26 }
27 }
28 return M;
29 }
30 Matrix pow(Matrix x, long long n) {
31     Matrix P(1);
32     while(n) {
33         if(n & 1) P = P * x;
34
35         n >>= 1;
36         x = x * x;
37     }
38     return P;
39 }
40 long long modpow(long long x, long long n) {
41     long long P = 1;
42
43     while(n) {
44         if(n & 1) P = P * x % MOD;
45         n >>= 1;
46         x = x * x % MOD;
47     }
48     return P;
49 }

```

7.18. Gauss Jordan

```

1  class fraction {public:
2      L n, d;
3      fraction(L n,L d) {
4          this->n=n;
5          this->d=d;
6          simplify();
7      }
8      void simplify(){
9          L g=__gcd(abs(n),abs(d));
10         n/=g;
11         d/=g;
12         if(n==0) d=1;
13         if(n<1 && d<1) {
14             n = abs(n);
15             d = abs(d);
16         } else if (n<1 || d<1) {
17             n=abs(n);
18             d=abs(d);
19             n =-n;
20         }
21     }
22     fraction inverse(){
23         return fraction(d,n);
24     }
25     fraction() {
26         n=0;
27         d=1;

```

```

28     }
29     fraction operator/(const fraction& f) {
30         fraction ret(n*f.d,d*f.n);
31         return ret;
32     }
33     fraction operator*(const fraction& f) {
34         fraction ret(n*f.n,d*f.d);
35         return ret;
36     }
37     fraction operator+(const fraction& f) {
38         fraction ret((n*f.d)+(f.n*d),d*f.d);
39         return ret;
40     }
41     fraction operator*(L x) {
42         fraction ret(x*n,d);
43         return ret;
44     }
45     fraction operator-() {
46         fraction ret(-n,d);
47         return ret;
48     }
49     fraction operator-(const fraction& f) {
50         fraction ret((n*f.d)-(f.n*d),d*f.d);
51         return ret;
52     }
53     bool operator>(const fraction& f) {
54         return abs(n*f.d)>abs(f.n*d);
55     }
56
57     fraction read() {
58         char c;
59         scanf("%lld",&n);
60         d = 1;
61         scanf("%c",&c);
62         if(c=='/')
63             scanf("%lld",&d);
64     }
65     void print() {
66         if(d==1) {
67             printf("%lld\n",n);
68         } else {
69             printf("%lld/ %lld\n",n,d);
70         }
71     }
72 };
73
74 bool hasSol;
75 int rankA,rankAC;
76
77 const int MX = 59;
78 vector<fraction> mat[MX];
79 fraction x[MX];
80 int N,Y,X,nunk;
81
82 void gauss() {
83     for(int p=1;p<=nunk;p++) {
84         int maxR=p;
85         for(int y=p+1;y<=Y;y++)

```



```

86         if(mat[y][p] > mat[maxR][p])
87             maxR=y;
88         swap(mat[p],mat[maxR]);
89
90         if(mat[p][p].n == 0)
91             continue;
92
93         fraction t = mat[p][p].inverse();
94         for(int i=1;i <= X;i++)
95             mat[p][i] = mat[p][i] * t;
96         for(int y=p+1;y<=Y;y++){
97             t = -mat[y][p];
98             for(int j=1;j <= X;j++) {
99                 mat[y][j] = mat[y][j] + (t*mat[p][j]);
100             }
101         }
102     }
103 }
104
105 void rank() {
106     rankA=rankAC=Y;
107     bool allZeroes;
108     for(int y=1;y<=Y;y++) {
109         allZeroes=true;
110         for(int j=1;allZeroes and j <= nunk;j++)
111             if(mat[y][j].n != 0)
112                 allZeroes=false;
113
114         if(allZeroes) {
115             rankA--;
116             if(mat[y][nunk+1].n == 0)
117                 rankAC--;
118         }
119     }
120 }
121
122 void resuelva() {
123     L num,den;
124     char c;
125
126     scanf("%d %d",&nunk,&Y);
127     X = nunk + 1;
128     for(int y=1;y <= Y;y++)
129         for(int x=1;x <= X;x++)
130             mat[y][x].read();
131
132     hasSol=true;
133     gauss();
134     rank();
135
136     if(rankAC != rankA){
137         printf("No Solution.\n");
138     } else {
139         if(rankAC<nunk)
140             printf("Infinitely many solutions containing %d arbitrary constants.\n",
141                 ,nunk-rankAC);
142         else{
143             for(int p=nunk;p>=1;--p) {

```

```

143         fraction s(0,1);
144         for(int k=p+1;k<=nunk;++k)
145             s = s + (x[k]*mat[p][k]);
146         x[p] = (mat[p][X]-s) / mat[p][p];
147     }
148     for(int y=1;y<=nunk;y++)
149         printf("x[%d] = ",y),x[y].print();
150     }
151 }
152 }
153 int main() {
154     int nc;
155     bool first=true;
156
157     for(int i=0;i<MX;i++)
158         mat[i].resize(MX);
159
160     while(scanf("%d",&N) and N>0) {
161         if(first == false) printf("\n");first = false;
162         printf("Solution for Matrix System # %d\n",N);
163         resuelva();
164     }
165 }

```

7.19. Numeros romanos

```

1 string AtoR(int A){
2     map<int, string> cvt;
3     cvt[1000] = "M"; cvt[900] = "CM"; cvt[500] = "D"; cvt[400] = "CD";
4     cvt[100] = "C"; cvt[90] = "XC"; cvt[50] = "L"; cvt[40] = "XL";
5     cvt[10] = "X"; cvt[9] = "IX"; cvt[5] = "V"; cvt[4] = "IV";
6     cvt[1] = "I";
7     string ans = "";
8     for(map<int, string>::reverse_iterator i = cvt.rbegin(); i != cvt.rend(); i++)
9         while(A >= (i->first)){
10             ans = ans + (i->second);
11             A -= i->first;
12         }
13     return ans;
14 }
15 int RtoA(char R[]){
16     map<char, int> RtoA;
17     RtoA['I'] = 1; RtoA['V'] = 5; RtoA['X'] = 10; RtoA['L'] = 50;
18     RtoA['C'] = 100; RtoA['D'] = 500; RtoA['M'] = 1000;
19     int value = 0;
20     for(int i = 0; R[i]; i++)
21         if(R[i+1] && RtoA[R[i]] < RtoA[R[i+1]]){
22             value += RtoA[R[i+1]] - RtoA[R[i]];
23             i++;
24         }else
25             value += RtoA[R[i]];
26     return value;
27 }

```

Capítulo 8

Cadenas

8.1. Utilidades

```
1 bool is_palindrome(string &ss){
2     return equal(ss.begin(), ss.end(), ss.rbegin());
3 }
```

8.2. Boyer Moore

Encuentra todos los match de un patron en el texto.

```
1 //Mejor caso(O(N/M)): Si el alfabeto y el patron son grandes
2 //Peor caso(O(NM)): Como fuerza bruta, no es recomendable si no cumple el mejor
   caso
3 void BoyerMooreHorspool(const string &T, const string &P) {
4     int N = T.size();
5     int M = P.size();
6     int sig[257];
7     fill(sig, sig + 257, M);
8     forn (i, M) sig[P[i]] = M - 1 - i;
9     int i = M - 1;
10    int j = M - 1;
11    while (i < N && j >= 0) {
12        if (T[i] == P[j])
13            i--, j--;
14        else {
15            i += sig[T[i]];
16            j = M - 1;
17        }
18        if (j < 0) {
19            cout << "Match : " << (i + 1) << endl;
20            i += M + 1;
21            j = M - 1;
22        }
23    }
24 }
```

8.3. Rabin Karp

```
1 void RabinKarp(string t, string s) {
2     const int p = 31; //si tambien hay mayusculas usar 53
```

```

3  vector<long long> p_pow (max (s.length(), t.length()));
4  p_pow[0] = 1;
5  for (int i = 1; i < p_pow.size(); ++i)
6      p_pow[i] = p_pow[i-1] * p;
7
8  vector<long long> h (t.length());
9  for (int i = 0; i < t.length(); ++i) {
10     h[i] = (t[i] - 'a' + 1) * p_pow[i]; //Cuidado con Mayusculas
11     if (i) h[i] += h[i-1];
12 }
13
14 long long h_s = 0;
15 for (int i = 0; i < s.length(); ++i)
16     h_s += (s[i] - 'a' + 1) * p_pow[i]; //Cuidado con Mayusculas
17
18 for (int i = 0; i + s.length() - 1 < t.length(); ++i) {
19     long long cur_h = h[i+s.length()-1];
20     if (i) cur_h -= h[i-1];
21
22     if (cur_h == h_s * p_pow[i])
23         cout << "Match : " << i << endl;
24 }
25 }

```

8.4. 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++;
19         sp++;
20         if(kp == K.size()) matches.push_back(sp - K.size());
21     }
22     /* En el vector matches se guardan los índices a cada una de las
23        ocurrencias, por tanto el tamaño de dicho vector nos dirá cuántas
24        ocurrencias se han encontrado. */
25     return matches.size();
26 }

```

8.5. Iesima permutación

```

1 //Entrada: Una cadena cad(std::string), un long th
2 //Salida : La th-esima permutacion lexicografica de cad
3 string ipermutacion(string cad, long long int th){
4     sort(cad.begin(), cad.end());
5     string sol = "";
6     int pos;
7     for(int c = cad.size() - 1; c >= 0; c--){
8         pos = th / fact[c];
9         th %= fact[c];
10        sol += cad[pos];
11        cad.erase(cad.begin() + pos);
12    }
13    return sol;
14 }

```

8.6. Algoritmo de Manacher

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

```

1 #define MAXN 20100
2
3 int N,i,j,izq,der,posi,dist;
4 string cad,tmp;
5 char v[MAXN];
6 int pos[MAXN];
7
8 void todoslosPalindromos(vector<int> &LP, int par){
9     izq = der = -1;
10    for(posi = 0; posi < N; posi++){
11        dist = 0;
12        if(posi <= der)
13            dist = min(der - posi + par, LP[izq+der-(posi+par)]) + 1;
14        while(posi-dist>=0 && posi+dist-par<N
15            && v[posi-dist]==v[posi+dist-par])
16            dist++;
17        LP[posi] = --dist;
18        if(posi+dist-par > der){
19            der = posi + dist - 1;
20            izq = posi - dist ;
21        }
22    }
23 }
24
25 int main() {
26     getline(fin,cad);
27     while(getline(fin,tmp))
28         cad +='\n'+tmp;
29     N = 0;
30     for(i = 0; i < cad.length(); i++){
31         if(isalpha(cad[i])){
32             v[N] = tolower(cad[i]);
33             pos[N] = i;
34             N++;
35         }
36     }
37     v[N] = '\0';
38     vector<int> LP(N,0), LI(N,0);

```

```

38     todoslosPalindromos(LP,1);
39     todoslosPalindromos(LI,0);
40     int SOL1 = 0,SOL2 = 0;
41     //primero en los de tamaño impar
42     for(int i = 0; i < N; i++)
43         if(LI[i]*2+1 > LI[SOL1]*2+1)
44             SOL1 = i;
45     //luego en los de tamaño par
46     for(int i = 0; i < N; i++)
47         if(LP[i]*2 > LP[SOL2]*2)
48             SOL2 = i;
49
50     if(LI[SOL1]*2+1 > LP[SOL2]*2) {
51         izq = SOL1 - LI[SOL1];
52         der = SOL1 + LI[SOL1];
53     }else{
54         izq = SOL2 - LP[SOL2];
55         der = SOL2 + LP[SOL2]-1;
56     }
57
58     fout << der - izq + 1 << endl;
59
60     izq = pos[izq];
61     der = pos[der];
62     fout << cad.substr(izq,der-izq+1)<<endl;
63     return 0;
64 }

```

Otra implementacion:

```

1  #define MAXN 600005
2  char str[MAXN];
3  int rad[MAXN];
4  int n;
5  /*
6  En rad se guarda el tamaño del palindromo
7  str[0]='$'; // si o si
8  scanf("%s", str+1); // puntero para iniciar desde 1
9  n=strlen(str+1);
10 */
11 void Manacher() {
12     int i, j, k;
13     i = 1;
14     j = 0;
15     while (i <= n) {
16         while (str[i - j] == str[i + j + 1] && i - j > 0 && i + j + 1 <= n) j++;
17         rad[i] = j;
18         k = 1;
19         while (k <= rad[i] && rad[i - k] != rad[i] - k) {
20             rad[i + k] = min(rad[i - k], rad[i] - k);
21             k++;
22         }
23         i = i + k;
24         j = max(j - k, 0);
25     }
26 }

```

8.7. Longest Common Subsequence (LCS)

```

1  int longestCommonSubsequence(const string& a, const string& b) {
2      int A = a.size(), B = b.size();
3      int L[2][B + 1];
4      for (int i = 0; i <= 1; ++i) L[i][0] = 0;
5      for (int i = 0; i <= B; ++i) L[0][i] = 0;
6      for (int i = 1; i <= A; ++i) {
7          int this_i = i % 2, pre_i = this_i ? 0 : 1;
8          for (int j = 1; j <= B; ++j) {
9              if (a[i - 1] == b[j - 1]) L[this_i][j] = 1 + L[pre_i][j - 1];
10             else L[this_i][j] = max(L[pre_i][j], L[this_i][j - 1]);
11         }
12     }
13     return max(L[0][B], L[1][B]);
14 }
15
16 int main() {
17     string a, b;
18     cin >> a >> b;
19     cout << longestCommonSubsequence(a, b);
20 }

```

8.8. Suffix Array

```

1  #define MAXN 1000005
2  int n,t; //n es el tamaño de la cadena
3  int m,tam1;
4  set<string> orde;
5  int p[MAXN],r[MAXN],h[MAXN];
6  //r índices del sa para mostrar índice s.substr(r[i])
7  //p es el inverso del suffix array, no usa índices del suffix array ordenado
8  //h el tamaño del lcp entre el i-esimo y el i+1-esimo elemento de suffix
   array ordenado
9  char s[250000];
10 char s2[250000];
11 void fix_index(int *b, int *e) {
12     int pkml, pk, np, i, d, m;
13     pkml = p[*b + t];
14     m = e - b; d = 0;
15     np = b - r;
16     for(i = 0; i < m; i++) {
17         if (((pk = p[*b+t]) != pkml) && !(np <= pkml && pk < np+m)) {
18             pkml = pk;
19             d = i;
20         }
21         p[*b++] = np + d;
22     }
23 }
24
25 bool comp(int i, int j) {
26     return p[i + t] < p[j + t];
27 }
28 void suff_arr() {
29     int i, j, bc[256];
30     t = 1;

```

```

31     for(i = 0; i < 256; i++) bc[i] = 0;    //alfabeto
32     for(i = 0; i < n; i++) ++bc[int(s[i])]; //counting sort inicial del alfabeto
33     for(i = 1; i < 256; i++) bc[i] += bc[i - 1];
34     for(i = 0; i < n; i++) r[--bc[int(s[i])]] = i;
35     for(i = n - 1; i >= 0; i--) p[i] = bc[int(s[i])];
36     for(t = 1; t < n; t *= 2) {
37         for(i = 0, j = 1; i < n; i = j++) {
38             while(j < n && p[r[j]] == p[r[i]]) ++j;
39             if (j - i > 1) {
40                 sort(r + i, r + j, comp);
41                 fix_index(r + i, r + j);
42             }
43         }
44     }
45 }
46
47 void lcp() {
48     int tam = 0, i, j;
49     for(i = 0; i < n; i++) if (p[i] > 0) {
50         j = r[p[i] - 1];
51         while(s[i + tam] == s[j + tam]) ++tam;
52         h[p[i] - 1] = tam;
53         if (tam > 0) --tam;
54     }
55     h[n - 1] = 0;
56 }
57 int owner(int idx) {
58     return (idx < n-m-1) ? 1 : 2;
59 }
60 int main() {
61     freopen("760.in", "r", stdin);
62     bool sw=false;
63     while(scanf("%s", &s)==1) {
64         scanf("%s", &s2);
65         strcat(s, "$");
66         tam1=strlen(s);
67         strcat(s, s2);
68         strcat(s, "#");
69         n =strlen(s);
70         m=strlen(s)-tam1;
71         suff_arr();
72         lcp();
73         int dev=0;
74         string dev2;
75         orde.clear();
76         for(int i=0; i<n; i++) {
77             if(owner(r[i]) != owner(r[i-1])){    //por si quiere mostrat $ o #
78                 if(h[i-1] >= dev) {
79                     dev=h[i-1];
80                     //cout<<r[i]<<" --- "<<h[i-1]<<endl;
81                     string aux="", aux2="";
82                     for(int j=0; j<dev; j++) {
83                         aux2=s[r[i]+j];
84                         aux+=aux2;
85                     }
86                     orde.insert(aux);
87                 }
88             }

```



```

89     }
90
91     if(sw)
92         printf("\n");
93     sw = true;
94     if(dev == 0)
95         printf("No common sequence.\n");
96     else{
97         set<string>::iterator it=orde.begin();
98         for (;it!=orde.end(); it++){
99             if((*it).size()==dev)cout <<*it<< endl;
100         }
101     }
102 }
103 return 0;
104 }
105
106 //rotacion menor lexicografica
107 /*
108 int tam1=strlen(s);
109 for(int i=tam1;i<2*tam1;i++)s[i]=s[i-tam1];
110 n=2*tam1;
111 suff_arr();
112 char dev[tam1];
113 for(int i=0;i<n;i++)
114     if(r[i]<tam1){
115         for(int j=r[i];j<r[i]+tam1;j++)
116             dev[j-r[i]]=s[j];
117         break;
118     }
119 for(int i=0;i<tam1;i++)
120     printf("%c",dev[i]);
121
122 */
123 /*
124 ACM 2009 File Recover Solucion
125 Problema: Contar los substrings q se repiten al menos una vez.
126 Analisis: Notamos que si el lcp(i,i+1) con lcp(i+1,i+2) aumenta
127 quiere decir que encontramos h[i+1]-h[i] palabras nuevas (prefijos)
128 SUM(max(h[i+1]-h[i],0))
129 */

```

8.9. Suffix Array DC3

```

1 //from http://blog.csdn.net/acdreamers/article/details/10746023
2 const int N=250005;
3
4 struct State
5 {
6     State *pre,*go[26];
7     int step;
8     void clear()
9     {
10         pre=0;
11         step=0;
12         memset(go,0,sizeof(go));
13     }

```

```

14 }*root,*last;
15
16 State statePool[N*2],*cur;
17
18 void init()
19 {
20     cur=statePool;
21     root=last=cur++;
22     root->clear();
23 }
24
25 void Insert(int w)
26 {
27     State *p=last;
28     State *np=cur++;
29     np->clear();
30     np->step=p->step+1;
31     while(p&&!p->go[w])
32         p->go[w]=np,p=p->pre;
33     if(p==0)
34         np->pre=root;
35     else
36     {
37         State *q=p->go[w];
38         if(p->step+1==q->step)
39             np->pre=q;
40         else
41         {
42             State *nq=cur++;
43             nq->clear();
44             memcpy(nq->go,q->go,sizeof(q->go));
45             nq->step=p->step+1;
46             nq->pre=q->pre;
47             q->pre=nq;
48             np->pre=nq;
49             while(p&&p->go[w]==q)
50                 p->go[w]=nq, p=p->pre;
51         }
52     }
53     last=np;
54 }
55
56 char A[N],B[N];
57
58 int main()
59 {
60     int n,m;
61     scanf("%s%s",A,B);
62     n=strlen(A);
63     m=strlen(B);
64     init();
65     for(int i=0; i<n; i++)
66         Insert(A[i]-'a');
67     int ans=0,len=0;
68     State *p=root;
69     for(int i=0; i<m; i++)
70     {
71         int x=B[i]-'a';

```

```

72         if(p->go[x])
73         {
74             len++;
75             p=p->go[x];
76         }
77         else
78         {
79             while(p&&!p->go[x]) p=p->pre;
80             if(!p) p=root, len=0;
81             else len=p->step+1, p=p->go[x];
82         }
83         ans=max(ans, len);
84     }
85     printf("%d\n", ans);
86     return 0;
87 }

```

8.10. Minima Rotacion Lexicografica

```

1  /*
2  Rotacion Lexicografica minima MinRotLex(cadena,tamano)
3  para cambiar inicio de la cadena char s[300]; int h; s+h;
4  retorna inicio de la rotacion minima :D
5  */
6  int MinRotLex(const char *s, const int slen) {
7      //for(int j=0; j<slen; j++){
8      //    cout<<s[j];
9      // }
10     int i = 0, j = 1, k = 0, x, y, tmp;
11     while(i < slen && j < slen && k < slen) {
12         x = i + k;
13         y = j + k;
14         if(x >= slen) x -= slen;
15         if(y >= slen) y -= slen;
16         if(s[x] == s[y]) {
17             k++;
18         } else if(s[x] > s[y]) {
19             i = j+1 > i+k+1 ? j+1 : i+k+1;
20             k = 0;
21             tmp = i, i = j, j = tmp;
22         } else {
23             j = i+1 > j+k+1 ? i+1 : j+k+1;
24             k = 0;
25         }
26     }
27     return i;
28 }
29 int main(){
30     int n;
31     scanf("%d", &n); getchar();
32     while(n--){
33         char str[10009];
34         gets(str);
35         printf("%d\n", Minlex(str, strlen(str))+1);
36     }
37 }

```

8.11. Trie

```

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

```

```

57
58 bool Trie::searchWord(string s)
59 {
60     Node* current = root;
61
62     while ( current != NULL )
63     {
64         for ( int i = 0; i < s.length(); i++ )
65         {
66             Node* tmp = current->findChild(s[i]);
67             if ( tmp == NULL )
68                 return false;
69             current = tmp;
70         }
71
72         if ( current->wordMarker )
73             return true;
74         else
75             return false;
76     }
77
78     return false;
79 }
80
81
82 int main() {
83     Trie* trie = new Trie();
84     trie->addWord("algo");
85     trie->addWord("ala");
86     trie->addWord("abeja");
87     trie->addWord("abel");
88     trie->addWord("trie");
89
90     if ( trie->searchWord("abel") )
91         cout << "Encontrado" << endl;
92     else cout << "No encontrado" << endl;
93
94
95     delete trie;
96 }

```

8.12. Aplicaciones

```

1  ////////////////////////////////////////////////// Suffix Array //////////////////////////////////
2  Algunas Aplicaciones
3  ////////////////////////////////////////////////// Rotacion lexicografica menor //////////////////////////////////
4      scanf("%s",&s);
5      tam1=strlen(s);
6      for(int i=tam1;i<2*tam1;i++){
7          s[i]=s[i-tam1];
8      }
9      n=2*tam1;
10     s[n++]="'~'"; //cual quiera de estos { | } ~
11     suff_arr();
12     char cam[tam1];
13     for(int i=0;i<n;i++){
14         if(r[i]<tam1){

```

```

15         printf("%d\n", r[i]+1); //inicio lexicografica
16         for(int j=r[i]; j<r[i]+tam1; j++)
17             cam[j-r[i]]=s[j]; //llenar camino
18         break;
19     }
20 }
21 for(int i=0; i<tam1; i++){
22     printf("%c", cam[i]); //print camino
23 }
24 //nota es un poco caro
25
26 ////////////////////////////////////////////////// palindromo mas grande //////////////////////////////////////////////////
27 usar Manachae
28 ////////////////////////////////////////////////// mayor substring 2 cadenas //////////////////////////////////////////////////
29 n,m tamaño respectivo de las cadenas 1,2 respectivamente
30 int owner(int idx){
31     return (idx < n-m-1) ? 1 : 2;
32 }
33 //main
34 for(int i=0; i<n; i++){
35     if(owner(r[i]) != owner(r[i-1])){ //por si quiere mostrar $ o #
36         ans=max(ans, h[i-1]); // ans subcadena maxima
37     }
38 }
39 //end main
40 ////////////////////////////////////////////////// subcadenas distintas //////////////////////////////////////////////////
41 int dev=s.size()*(s.size()+1)/2;
42 s += '$';
43 suff_arr(); lcp();
44 for(int i=0; i<n; i++) dev-=h[i];
45 ////////////////////////////////////////////////// FIN SUFFIX ARRAY //////////////////////////////////////////////////

```

Capítulo 9

Geometria Computacional

9.1. Intersección de rectangulos

```
1 struct Rect{
2     int x1,y1, x2,y2;
3     int area;
4     Rect(int _x1, int _y1, int _x2, int _y2){
5         x1 = _x1;
6         y1 = _y1;
7         x2 = _x2;
8         y2 = _y2;
9         getArea();
10    }
11    int getArea(){
12        if(x1>=x2 || y1>=y2) return area = 0;
13        return area = (x2-x1)*(y2-y1);
14    }
15 };
16
17 Rect interseccion(Rect t, Rect r){
18     int x1,y1,x2,y2;
19     x1 = max(t.x1,r.x1);
20     y1 = max(t.y1,r.y1);
21     x2 = min(t.x2,r.x2);
22     y2 = min(t.y2,r.y2);
23     Rect res(x1,y1,x2,y2);
24     return res;
25 }
26 /*
27 -----
28 |      |   arr   |      |   <=== sobre
29 |      |_____|      |
30 |      |_____|      |
31 |izq| dentro | der|
32 |      |_____|      |
33 |      |      |      |
34 |      |   aba   |      |
35 -----
36 */
37 //Si "dentro" cubre todo "sobre" no olvidar borrar "sobre"
38 //o mantenerlo pero no adicionar dentro
39 vector<Rect> obtenerbordes(Rect sobre, Rect dentro) {
40     Rect inter = interseccion(sobre, dentro);
41     Rect arr(inter.x1,inter.y2,inter.x2, sobre.y2);
```

```

42 Rect aba(inter.x1, sobre.y1, inter.x2, inter.y1);
43 Rect izq(sobre.x1, sobre.y1, inter.x1, sobre.y2);
44 Rect der(inter.x2, sobre.y1, sobre.x2, sobre.y2);
45 vector<Rect> ans;
46 if (arr.getArea()) ans.pb(arr);
47 if (aba.getArea()) ans.pb(aba);
48 if (izq.getArea()) ans.pb(izq);
49 if (der.getArea()) ans.pb(der);
50 return ans;
51 }

```

9.2. Distancia Punto - Segmento

```

1 struct point{
2     double x,y;
3 };
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

<https://github.com/BranimirE/Library/archive/master.zip>

10.1. Plantilla

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 #define DBG(X) cerr << #X << " = " << X << endl;
6 #define pb push_back
7 #define mp make_pair
8 #define sz(x) ((int)(x).size())
9 #define all(c) (c).begin(), (c).end()
10 #define forn(i, n) for (int i = 0; i < (n); i++)
11
12 int main() {
13
14     return 0;
15 }
```

10.2. Lectura rapida

```
1 #include <cstdio>
2
3 using namespace std;
4
5 char line[4000000]; //OJO maximo mas 3 para el '\n' y el '\0'
6 int now = 0;
7 inline int getInt() {
8     int n;
9     while(1)
10         if(line[now] != 0) {
11             if(line[now] < '0' || line[now] > '9') {
12                 now++;
13                 continue;
14             }
15             n = 0;
16             while(line[now] >= '0' && line[now] <= '9') {
17                 n = n*10 + line[now] - '0';
18                 now++;
19             }
20             return n;
21         }
22 }
```

```

19         }
20         return n;
21     }
22     else{
23         gets(line);
24         now = 0;
25     }
26     return n;
27 }

```

10.3. Especificadores 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++ :

```

1 __builtin_popcount(unsigned int)
2 __builtin_popcountll(unsigned long long)

```

Java :

```

1 Integer.bitCount(int)

```

10.5. Búsqueda 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
```