First to Penalty

Contents

1 Template

```
#include "bits/stdc++.h"
//assert(x>0) si falla da RTE
using namespace std;
#define endl '\n'
#define DBG(x) cerr<<#x<< "=" << (x) << endl;
#define RAYA cerr<<"====================="<<endl;
#define RAYAS cerr<<"....................."<<endl;
//#define DBG(x) ;
//#define RAYA ;
//#define RAYAS ;

//-------------------SOLBEGIN--------------------------------
int main() {
  ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
  int tC;

  cin >> tC;
  while (tC--) {

  }

}
//-------------------EOSOLUTION--------------------------------
```

2 Data structures

2.1 Simplified DSU (Stolen from GGDem)

```
int uf[MAXN];
void uf_init(){memset(uf,-1,sizeof(uf));}
int uf_find(int x){return uf[x]<0?x:uf[x]=uf_find(uf[x]);}
bool uf_join(int x, int y){
  x=uf_find(x);y=uf_find(y);
  if(x==y)return false;
  if(uf[x]>uf[y])swap(x,y);
  uf[x]+=uf[y];uf[y]=x;
  return true;
}
```

2.2 Disjoint Set Union

```
class disjSet {
  int* sz;
  int* par;
public:
  int len;
  disjSet(int tam){
        sz = new int[tam + 4]();
        par = new int[tam + 4]();
        len = 0;
        for(int i = 0; i<=tam; i++){
            par[i] = i;
            sz[i] = 1;
            len++;
        }
    }
  int finds(int el){
        if (el == par[el]) return el;
        return par[el] = finds(par[el]);
    }
  void unions(int a, int b){
        a = finds(a);
      b = finds(b);
        if (a == b) return;
        len--;
        //se hace que el gde sea padre del pequeno
        if (sz[a] > sz[b]) swap(a,b);
        par[a] = b;
        sz[b] += sz[a];
    }
  ~disjSet(){
        delete[] size;
        size = nullptr;
        delete[] parent;
        parent = nullptr;
    }
};
```

2.3 Segment tree

```
//MAXN = 2^k, n = tam arreglo inicial
int stsize; long long int neut;int n;
long long int* st = new long long int[2*MAXN-1]();
long long int fst(long long int a, long long int b);
```

```
long long int build(int sti,int csize){
    if(csize == 1) return st[sti];
    return st[sti] = fst(build(sti*2+1,csize/2),build(sti*2+2,csize/2));
}
void innit(){
    for(int i = 0; i<stsize; i++) st[i] = neut;
    /*int d = 0;
    for(int i = stsize-n; i<stsize && d<n; i++){
        st[i] = arr[d];d++;
    }*/
    build(0,n);
}
void upd(int ind, long long int val){
    ind = stsize-n+ind;
    st[ind] = val;ind--;ind/=2;
    while(true){
        st[ind] = fst(st[ind*2+1],st[ind*2+2]);
        ind--;
        if(ind<0) break;
        ind/=2;
    }
}
long long int rqu(int l, int r,int sti, int ls, int rs){
    if(l<=ls && rs<= r) return st[sti];
    if(r<ls || l>rs) return neut;
    int m = (rs+ls)/2;
    return fst(rqu(l,r,sti*2+1,ls,m),rqu(l,r,sti*2+2,m+1,rs));
}
long long int query(int l, int r){
    return rqu(l,r,0,0,n-1);
}
//uso, inicializa neut, determina n (asegurate que sea una potencia de 2), define fst para determinar
//la opracion del segment tree
```

2.4 Segment tree Lazy

```
//MAXN = 2^k, n = tam arreglo inicial
#define MAXN 524288
vector<int> arr;
int stsize; long long int neut;int n;
long long int* st = new long long int[2*MAXN-1]();
long long int* pendientes = new long long int[2*MAXN-1]();
long long int fst(long long int a, long long int b){return a+b;}
long long int build(int sti,int csize){
    if(csize == 1) return st[sti];
    return st[sti] = fst(build(sti*2+1,csize/2),build(sti*2+2,csize/2));
}
bool hasChildren(int sti){sti*=2;sti++;sti++;return sti<stsize;}
void innit(){
    for(int i = 0; i<stsize; i++) st[i] = neut;
    int d = 0;
    for(int i = stsize-n-1; i<stsize && d<n; i++) {st[i] = arr[d];d++;}
    build(0,n);
}
void updrec(int l,int r, int sl, int sr,int sti, long long int val){
    if(sr<l || r< sl) return;
    if(l<= sl && sr <=r){
        st[sti] += val*(sr-sl+1);
        if(hasChildren(sti)){pendientes[sti*2+1]+=val;pendientes[sti*2+2]+=val;}
        return;
    }

    int sm = (sl+sr)/2;
    updrec(l,r,sl,sm,sti*2+1,val);
    updrec(l,r,sm+1,sr,sti*2+2,val);
    st[sti] = fst(st[sti*2+1],st[sti*2+2]);
}
void upd(int l, int r, long long int val){updrec(l,r,0,n-1,0,val);}

long long int rqu(int l, int r,int sti, int ls, int rs){
    if(r<ls || l>rs) return neut;
    if(l<=ls && rs<= r){
        return st[sti]+pendientes[sti]*(rs-ls+1);
    }

    st[sti] += pendientes[sti]*(rs-ls+1);
    if(hasChildren(sti)){pendientes[sti*2+1]+=pendientes[sti];pendientes[sti*2+2]+=pendientes[sti];}
    pendientes[sti] = 0;

    int m = (rs+ls)/2;
    return fst(rqu(l,r,sti*2+1,ls,m),rqu(l,r,sti*2+2,m+1,rs));
}
long long int query(int l, int r){
```

```
    return rqu(l,r,0,0,n-1);
}
//uso, inicializa neut, lee n y arr, iguala n a la potencia de dos mas cercana y mayor
//determina stsize = 2*n (asegurate que sea una potencia de 2), define fst para determinar
//la opracion del segment tree
```

3 Graphs

3.1 Graph Transversal

3.1.1 BFS

```
#define GS 400040
vector<int> graph[GS];
bitset <GS> vis;
//anchura O(V+E)
void dfs(int curr) {
  queue<int> fringe;
  fringe.push(curr);
  while (fringe.size()) {
    curr = fringe.front(); fringe.pop();
    if (!vis[curr]) {
      vis[curr] = 1;
      for (int h : graph[curr]) fringe.push(h);
    }
  }
}
```

3.1.2 DFS

```
#define GS 400040
vector<int> graph[GS];
bitset <GS> vis;
//profundidad O(V+E)
void dfs(int curr) {
  stack<int> fringe;
  fringe.push(curr);
  while (fringe.size()){
    curr = fringe.top(); fringe.pop();
    if (!vis[curr]) {
      vis[curr] = 1;
      for (int h : graph[curr]) fringe.push(h);
    }
  }
}
```

3.2 Topological Sort

```
#define GS 400040
vector<int> graph[GS];
bitset <GS> vis;
vector<int> topsort;
int e,n;
//profundidad
//O(N+E)
//Solo funciona con DAG's, no existe un top sort de un grafo Non-DAG
void todfs(int pa) {
  vis[pa]=1;
  for(int h: graph[pa]){if(!vis[h]){todfs(h);}}
  topsort.push_back(pa);
}
void topologicalSort(){
  vis.reset();
  topsort.clear();
  for(int i = 0; i<n; i++){if(!vis[i]){dfs(i);}}
  reverse(topsort.begin(),topsort.end());
}
```

3.3 APSP: Floyd Warshall

```
#define GS 1000
#define INF 100000000
//destino, costo
int graph[GS][GS];
//All Pairs Dist
int dist[GS][GS];
//Toma en cuenta nodos [0-tam] inclusivo, modificar de acuerdo a las necesidades
//Ten cuidado con el valor que le pones a INF, puede provocar overflows o puede no ser lo suficientemente grande.
void Floyd_Warshall(int tam){
    for(int i = 0; i<=tam; i++)
        for(int f = 0; f<=tam; f++)
            dist[i][f] = INF;

    for(int i = 0; i<=tam; i++)
```

```
        for(int f = 0; f<=tam; f++)
            dist[i][f] = graph[i][f];

    //para reconstruir el camino solo basta con guardar intermedio como
        el padre de ini si el cambio se hizo, -1 otherwise
    for(int intermedio = 0; intermedio<=tam; intermedio++)
        for(int ini = 0;  ini<=tam; ini++)
            for(int fin = 0; fin<=tam; fin++)
                dist[ini][fin] = min(dist[ini][fin],dist[ini][intermedio
                    ]+dist[intermedio][fin]);
}
```

3.4 SSSP

3.4.1 Lazy Dijkstra

```
#define GS 1000
#define INF 100000000
//destino, costo
vector<pair<int,int>> graph[GS];
int dist[GS];
void dijkstra(int origen,int tam){
    for(int i = 0; i<=tam; i++){
        dist[i] = INF;
    }
    priority_queue<pair<int,int>,vector<pair<int,int>>, greater<pair<int
        ,int>>> pq;
    pair<int,int> curr;

    pq.push(make_pair(0,origen));

    while(pq.size()){
        curr = pq.top();pq.pop();
        if(curr.first >= dist[curr.second]) continue;

        dist[curr.second] = curr.first;
        for(pair<int,int> h: graph[curr.second]){
            if((h.second+curr.first)<dist[h.first]) pq.push({h.second+
                curr.first,h.first});
        }
    }
}
//Esta es la implementacion huevona
//Resuelve Single Source Shortest Paths con aristas positivas
//Como es la lazy implementation, si funciona con edges negativos
    siempre y cuando no hayan ciclos negativos
//Si hay ciclos negativos se va atascar en un ciclo infinito
//Si no los hay puede que funcione en O((V+E)log(V)) o puede que se
    exponencial, si no jala prueba BellmanFord
```

3.4.2 Bellman-Ford

```
//esta es la implementacion huevona
#define GS 1000
//cuidado con overflows!!
#define INF 100000000
#define NINF -100000000
//destino, costo
vector<pair<int,int>> graph[GS];
int dist[GS];
struct edge{
    int from,to,cost;
};
//Corre en O(VE)
void bellmanFord(int origen,int tam){
    for(int i = 0; i<=tam; i++){
        dist[i] = INF;
    }
    dist[origen] = 0;
    edge aux;
    vector<edge> aristas;
    bool optimal;

    for(int i = 0; i<=tam; i++){
        for(pair<int,int> h: graph[i]){
            aux.from = i; aux.to  = h.first;aux.cost = h.second;
            aristas.push_back(aux);
        }
    }

    //Si se relajan todos las aristas V-1 veces en un orden arbitrario
    //Se asegura que la distancia optima para cada vertice sera
        alcanzada
    for(int i = 0; i<tam && !optimal; i++){
        optimal = true;
        for(edge elem: aristas){
            if(dist[elem.from] + elem.cost < dist[elem.to]){
```

```
                dist[elem.to] = dist[elem.from] + elem.cost;
                //si algun vertice fue actualizado significa que puede que
                //las distancias aun no sean optimas
                optimal = false;
            }
        }
    }

    //Se corre de nuevo para asegurar encontrar todos los ciclos negativos
    for(int i = 0; i<tam && !optimal; i++){
        optimal = true;
        for(edge elem: aristas){
            if(dist[elem.from] + elem.cost < dist[elem.to]){
                //Si aun despues de correr V-1 veces se puede actualizar
                //Significa que esta en un ciclo negativo
                dist[elem.to] = NINF;
                //si algun vertice fue actualizado significa que puede que
                //las distancias aun no sean optimas
                optimal = false;
            }
        }
    }

}
```

3.5 Strongly Connected Components: Kosaraju

```
#define GS 2010
vector<int> graph[GS];
vector<int> graphI[GS];
vector<int> orden;
bitset<GS> vis;

void invertirGrafo(int n){
    for(int p = 1;p<= n; p++)
        for(int h: graph[p])graphI[h].push_back(p);
}
void obtOrd(int p,int n){
    vis[p] = 1;
    for(int h: graph[p]){
        if(!vis[h] && h<=n) obtOrd(h,n);
    }
    orden.push_back(p);
}
int findSCC(int n){
    int res = 0;
    invertirGrafo(n);
    orden.clear();
    for(int i = 1; i<=n; i++) vis[i] =0;
    for(int i = 1; i<=n; i++) if(!vis[i]) obtOrd(i,n);
    reverse(orden.begin(),orden.end());
    //cuenta los connected components
    //vector<int> lscc;
    stack<int> fringe;
    int curr;
    for(int i = 1; i<=n; i++) vis[i] =0;
    for(int i: orden){
        //lscc.clear();
        if(!vis[i]){
            fringe.push(i);
            while (fringe.size()){
                curr = fringe.top();fringe.pop();
                //lscc.push_back(curr);
                if (!vis[curr]) {
                    vis[curr] = 1;
                    for (int h : graphI[curr]) fringe.push(h);
                }
            }
            res++;
        }
        //hacer lo que sea con lcss
    }
    return res;
}

//OJO esto solo jala con directed graphs
//por definicion todas las undirected graphs tienen un solo SCC
//NOTAR QUE LOS GRAFOS QUE USA CUMPLEN CON: 0<=VERTICE<=n
```

3.6 Articulation Points and Bridges: ModTarjan

```
#define GS 50
vector<int> graph[GS];
```

```
bitset<GS> vis, isArtic;
vector<int> padre;
//id por tiempo, menor id accesible
//ya sea por descendientes o por back edges
vector<int> tId,lId;
//cantidad de hijos que tiene en el bfs spanning tree
int rootChildren;
int cnt;
int dfsRoot;
void findAP_B(int p){
    cnt++;vis[p] = 1;tId[p] = cnt;lId[p] = tId[p];

    for(int hijo: graph[p]){
        if(!vis[hijo]){
            padre[hijo] = p;
            if(p == dfsRoot) rootChildren++;

            findAP_B(hijo);

            //esto significa que ni por un back edge el hijo accede al padre
            //por lo que si el padre fuese eliminado el hijo quedaria aislado
            if(lId[hijo] >= tId[p]) isArtic[p] = 1;
            if(lId[hijo] > tId[p]){
                //esto significa que si se eliminase el camino de padre ->hijo
                //se lograria desconectar el grafo, aka bridge
            }
            lId[p] = min(lId[p],lId[hijo]);
        }else{
            //si hay un ciclo indirecto, actualiza el valor para el padre
            if(hijo != padre[p]) lId[p] = min(lId[p],tId[hijo]);
        }
    }
}
//OJO esto solo jala con Undirected graphs
/*
    MAIN
    for(int i = 0; i<n; i++){
        if(!vis[i]){
            rootChildren = 0;
            dfsRoot = i;
            findAP_B(i);
            //el algoritmo no puede detectar si el nodo que lo origino
            //es un articulation point, por lo que queda checar si
            //en el spanning tree que genero tiene mas de un solo hijo
            isArtic[i] = (rootChildren>1?1:0);
        }
    }
*/
```

4 Math

4.1 Identities

$$C_n = \frac{2(2n-1)}{n+1} C_{n-1}$$
$$C_n = \frac{1}{n+1}\binom{2n}{n}$$
$$C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$$
$$\sigma(n) = O(\log(\log(n))) \text{ (number of divisors of } n\text{)}$$
$$F_{2n+1} = F_n^2 + F_{n+1}^2$$
$$F_{2n} = F_{n+1}^2 - F_{n-1}^2$$
$$\sum_{i=1}^{n} F_i = F_{n+2} - 1$$
$$F_{n+i}F_{n+j} - F_nF_{n+i+j} = (-1)^n F_iF_j$$

(Möbius Inv. Formula) Let $g(n) = \sum_{d|n} f(d)$, then $f(n) = \sum d \mid ng(d)\mu\left(\frac{n}{d}\right)$.

4.2 Binary Exponentiation and modArith

```
long long int inf = 10000000007;
//suma (a+b)%m
//resta ((a-b)%m+m)%m
//mult (a*b)%m
long long binpow(long long b, long long e) {
    long long res = 1; b%=inf;
    while (e > 0) {
        if (e & 1) res = (res * b)%inf;
        b = (b * b)%inf;
        e >>= 1;
    }
    return res;
}
```

4.3 Modular Inverse (dividir mod)

```
long long int inf = 10000000007;
```

```
long long int gcd(long long int a, long long int b, long long int& x,
    long long int& y) {
    x = 1, y = 0;
    long long int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        long long int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
    return a1;
}
long long int modinverse(long long int b, long long int m){
    long long int x,y;
    long long int d = extEuclid(b,inf,x,y);
    if(d!=1) return -1;
    return ((x%inf)+inf)%inf;
}
```

4.4 Modular Binomial Coeficient and Permutations

```
long long int inf = 10000000007;
//cat[n] = bincoef(2*n,n)/(n+1), cat[0] = 1
class binCoef{
    long long int lim;
    long long int* fact;
public:
    binCoef(long long int l){
        lim = l; fact = new long long int[l+1];fact[0]= 1;
        for(long long int i = 1; i<=l; i++) fact[i] = (fact[i-1]*i)%inf;
    }
    //perm = (fact[n] * modinverse(fac[n-k],inf)%inf;
    long long int query(long long int n, long long int k){
        if(n<k) return 0;
        return (fact[n] * modinverse((fac[n-k]*fact[k])%inf,inf))%inf;
    }
};
```

4.5 Non-Mod Binomial Coeficient and Permutations

```
//Solo usar con n<=20
//cat[n] = bincoef(2*n,n)/(n+1), cat[0] = 1
unsigned long long int bincoef(unsigned long long int n, unsigned long
    long int k){
    if(n<k) return 0;
    unsigned long long int num  = 1, den= 1;
    for(unsigned long long int i = (n-k)+1; i<=n; i++) num*=i;
    for(unsigned long long int i = 2; i<=k; i++) den*=i;
    //perm = return num;
    return num/den;
}
```

4.6 Modular Catalan Numbers

```
long long int inf = 10000000007;
class catalan{
    long long int* cat; long long int lim
public:
    catalan(long long int l){
        lim = l; cat = new long long int[l+10];cat[0] = 1;
        for(long long int i = 0;i<=l; i++) cat[i+1] = (((((4LL*i+2)%inf)
             *cat[i])%inf) *modinverse(n+2))%inf;
    }
    long long int query(long long int n){ return cat[n];}
};
```

4.7 Ceil Fraccionario

```
long long int techo(long long int num, long long int den){ return (num+
    den-1)/den;}
```

4.8 Numeros de Fibonacci

```
//en caso de ser usados mod un m pequeno
//recordar que los numeros de fibonacci se repiten por lo menos cada m^2
//O(n)
unsigned long long int fib(int n){
  unsigned long long int a = 1,b = 1,aux;
  if(n<=2){
    return 1;
  }
  for(int i = 3; i<=n; i++){
    aux = a+b;
    a = b;
    b = aux;
  }
  return b;
}
```

```
const long long int inf = 1000000007;
unordered_map<long long int,long long int> Fib;
//O(log n) :DD
long long int fib(long long int n)
{
    if(n<2) return 1;
    if(Fib.find(n) != Fib.end()) return Fib[n];
    Fib[n] = (fib((n+1) / 2)*fib(n/2) + fib((n-1) / 2)*fib((n-2) / 2)) %
         inf;
    return Fib[n];
}
```

4.9 Sieve Of Eratosthenes

```
#define MAXN 10e6
class soe{
public:
    bitset<MAXN> isPrime;
    soe(){
        for(int i = 3; i<MAXN; i++) isPrime[i] = (i%2);
        isPrime[2] = 1;
        for(int i = 3; i*i<MAXN; i+=2)
            if(isPrime[i])
                for(int j = i*i; j<MAXN; j+=i)
                    isPrime[j] = 0;
    }
};
```

4.10 Sieve-based Factorization

```
#define MAXN 10e6
class soe{
public:
    int smolf[MAXN];
    soe(){
        for(int i = 2; i<MAXN; i++) smolf[i] = (i%2==0?2:i);

        for(int i = 3; i*i<MAXN; i+=2)
            if(smolf[i]==i)
                for(int j = i*i; j<MAXN; j+=i)
                    smolf[j] = min(smolf[j],smolf[i]);
    }
};
```

4.11 Berlekamp Massey

```
typedef long long int ll;
//Obtiene recurrencia lineal dados los primeros elementos en O(n^2)
vector<ll> berlekampMassey(const vector<ll> &s) {
    vector<ll> c;
    vector<ll> oldC;
    int f = -1;
    for (int i=0; i<(int)s.size(); i++) {
        ll delta = s[i];
        for (int j=1; j<=(int)c.size(); j++) delta -= c[j-1] * s[i-j];
        if (delta == 0) continue;
        if (f == -1) {
            c.resize(i + 1);
            mt19937 rng(chrono::steady_clock::now().time_since_epoch().
                count());
            for (ll &x : c) x = rng();
            f = i;
        } else {
            vector<ll> d = oldC;
            for (ll &x : d) x = -x;
            d.insert(d.begin(), 1);
            ll df1 = 0;
            for (int j=1; j<=(int)d.size(); j++) df1 += d[j-1] * s[f+1-j
                ];
            assert(df1 != 0);
            ll coef = delta / df1;
            for (ll &x : d) x *= coef;
            vector<ll> zeros(i - f - 1);
            zeros.insert(zeros.end(), d.begin(), d.end());
            d = zeros;
            vector<ll> temp = c;
            c.resize(max(c.size(), d.size()));
            for (int j=0; j<(int)d.size(); j++) c[j] += d[j];
            if (i - (int) temp.size() > f - (int) oldC.size()) {oldC =
                temp;f = i;}
        }
    }
    return c;
}
```

4.12 Modular Berlekamp Massey

```
typedef long long int ll;
long long int inf = 1000000007;
vector<ll> bermas(vector<ll> x){
    vector<ll> ls,cur;
    int lf,ld;
    for(int i = 0; i<x.size(); i++){
        long long int t = 0;
        for(int j = 0; j<cur.size(); j++) t=(t+x[i-j-1]*(long long int)cur[j])%inf;
        if((t-x[i])%inf==0)continue;
        if(cur.size()==0){cur.resize(i+1);lf=i;ld=(t-x[i])%inf;continue;}
        long long int k = (x[i]-t)*powermod(ld,inf-2)%inf;
        vector<ll>c(i-lf-1);c.push_back(k);
        for(int j = 0; j<ls.size(); j++) c.push_back(-ls[j]*k%inf);
        if(c.size()<cur.size()) c.resize(cur.size());
        for(int j = 0; j<cur.size();j++) c[j]=(c[j]+cur[j])%inf;
        if(i-lf+ls.size()>=cur.size())ls=cur,lf=i,ld=(t-x[i])%inf;
            cur=c;
  }
    for(int i =0; i<cur.size(); i++) cur[i]=(cur[i]%inf+inf)%inf;
  return cur;
}
```

4.13 Matrix exponentiation

```
typedef vector<vector<long long int>> Matrix;
long long int inf = 1000000007;
Matrix ones(int n) {
  Matrix r(n,vector<long long int>(n));
  for(int i= 0; i<n; i++){
        r[i][i]=1;
    }
  return r;
}
Matrix operator*(Matrix &a, Matrix &b) {
  int n=a.size(),m=b[0].size(),z=a[0].size();
  Matrix r(n,vector<long long int>(m));
  for(int i=0; i<n; i++){
        for(int j=0; j<m; j++){
            for(int k=0;k<z; k++){
                r[i][j]+=((a[i][k]%inf)*(b[k][j]%inf))%inf;
                r[i][j]%=inf;}}}
  return r;
}
Matrix be(Matrix b, long long int e) {
  Matrix r=ones(b.size());
  while(e){if(e&1LL)r=r*b;b=b*b;e/=2;}
  return r;
}

//Matrix mat(n,vector<long long int>(n));
```

5 Geometry

6 Strings

6.1 Explode by token

```
//#include <sstream>

vector<string> explode(string const& s, char delim) {
  vector<string> result;
  istringstream iss(s);
  for (string token; getline(iss, token, delim); )
  {
    result.push_back(move(token));
  }
  return result;
}
```

6.2 Multiple Hashings DS

```
struct multhash{
    unsigned long long int h1,h2;
    unsigned long long int alf[257];
    bool operator < (multhash b) const {
    if (h1 != b.h1) return h1 < b.h1;
    return h2 < b.h2;
  }
  bool operator == (multhash b) const { return (h1== b.h1 && h2== b.h2);}
  bool operator != (multhash b) const { return !(h1== b.h1 && h2== b.h2);}
public:
    string s;
```

```
    multhash(){
        h1 = 0; h2 = 0;s = "";
        for(char l = 'a'; l<='z'; l++) alf[l] = l-'a'+1;
    }
    void innit(){
        unsigned long long int inf,p,op;

        inf = 999727999;
        p = 325255434;op = 325255434;
        for(char l: s){
            h1+=(p*alf[l])%inf;
            p*=op;
            p%=inf;
        }

        inf = 1070777777;
        p = 10018302;op = 10018302;
        for(char l: s){
            h2+=(p*alf[l])%inf;
            p*=op;
            p%=inf;
        }
    }
};
//VALORES ALTERNATIVOS DE INF, LOG 17
//666666555557777777
//986143414027351997
//974383618913296759
//973006384792642181
//953947941937929919
//909090909090909091
//VALORES PARA P, USAR PRIMOS MAYORES A |Alfabeto|
//31,47,53,61,79
```

6.3 Permute chars of string

```
void permute(string str){
  // Sort the string in lexicographically
  // ascennding order
  sort(str.begin(), str.end());

  // Keep printing next permutation while there
  // is next permutation
  do {
    cout<<str<<endl;
  } while (next_permutation(str.begin(), str.end()));
}
```

6.4 Longest common subsequence

```
//O(|te|*|pa|)
//cambiar score para otros problemas, str all match = +2, miss/ins/del =
    -1
//usar char que no este en el alfabeto para denotar del/ins
string te,pa;
long long int ninf = -10e13;
long long int score(char a, char b){
    if(a=='*' || b=='*') return 0;
    if(a==b) return 1;
    return ninf;
}
long long int lcs(){
    long long int** dp;te = "*"+te; pa = "*"+pa;
    long long int res = 0;

    dp = new long long int*[te.size()];
    for(int i = 0; i<te.size(); i++) dp[i] = new long long int[pa.size()
        ]();

    for(int r = 1; r<te.size(); r++){
        for(int c = 1; c<pa.size(); c++){
            dp[r][c] = dp[r-1][c-1]+score(te[r],pa[c]);
            dp[r][c] = max(dp[r][c-1]+score(te[r],'*'),dp[r][c]);
            dp[r][c] = max(dp[r-1][c]+score('*',pa[c]),dp[r][c]);
        }
    }

    return dp[te.size()-1][pa.size()-1];
}
```

6.5 KMP

```
string T,P;
int bt[MAXN];
//O(|Text|+|Pattern|)
void KMPpre(){
    int i = 0,j = 0; bt[0] = -1;
```

```
    while(i<P.size()){
        while(j>=0 && P[i]!=P[(j>=0?j:0)]) j = bt[j];
        i++;j++; bt[i] = j;
    }
}
int kmp(){
    int res =0, i = 0, j = 0;
    while(i<T.size()){
        while(j>=0 && T[i] != P[(j>=0?j:0)])  j = bt[j];
        i++; j++;
        if(j==P.size()){//match, do anything
            res++;j = bt[j];
        }
    }
    return res;
}
```

7 Flow

8 Miscellaneous

8.1 Bit Manipulation

```
#include "bits/stdc++.h"
using namespace std;
#define endl '\n'


int main() {
  ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
  //Se representan bitmasks de 30 a 62 bits
  //usando signed int y signed long long int
  //para evitar problemas con el complemento de dos
  signed int a, b;
  //para multiplicar un numero por dos solo es necesario aplicar un
  // shifteo de sus bits a la izquierda
  a = 1;
  a= a << 3;
  cout << a << endl;
  //para dividir un numero entre dos es necesario aplicar un
  //shifteo a la derecha
  a = 32;
  a = a >> 3;
  cout << a << endl;
  //para encender el bit n de a, solo hay que igualar a = a | pow(2,n-1)
  //prende el tercer bit
  a = 1;
  b = 1 << 2;
  a = a | b;
  cout << a << endl;
  //para apagar el bit n de a, solo hay que  a &= ~pow(2,n-1)
  //prende el tercer bit
  a = 5;
  b = 1 << 2;
  a &= ~b;
  cout << a << endl;
  //para revisar si el bit n de a esta encendido
  //revisa si el tercer bit esta encendido
  a = 5;
  b = 1 << 2;
  a = a & b;
  cout << (a?"SI":"NO") << endl;
  //para volter el bit n de a, solo hay que igualar a = a ^ pow(2,n-1)
  //apaga el tercer bit
  a = 5;
  b = 1 << 2;
  a = a ^ b;
  cout << a << endl;
  //para obtener el bit menos significativo que esta encendido a& -a
  a = 12;
  cout << log2(a & ((-1) * a))+1 << endl;
  //para prender todos los bits hasta n
  a = (1<<4)-1;
  cout << a << endl;
}
//-------------------EOSOLUTION-------------------------------
```

```
#include "bits/stdc++.h"
using namespace std;
#define endl '\n'
#pragma GCC optimize("O3")
#pragma GCC target("popcnt")

//no usar con visual c++
//solo con g++ like compilers
int main() {
```

```
    ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
    signed long long int a, b, n;
    //Obtain the remainder (modulo) of a when it is divided by n (n is a
        power of 2)
    a = 15; n = 8-1;
    a &= n;
    cout << "a%n, a = 15, n = 2^3" << endl;
    cout << a << endl;
    //Apaga el bit menos significativo de a
    a = 14;
    b = (a & ((-1) * a));
    a &= ~b;
    cout << a << endl;
    //enciende el ultimo cero de a
    a = 9;
    b = ~a;
    b = (b & ((-1) * b));
    a = a | b;
    cout << a<<endl;
    //contar bits encendidos en a
    cout << __builtin_popcount(a)<<endl;
    //checar la paridad de a
    cout << (__builtin_parity(a) ? "IMPAR" : "PAR") << endl;
    //contar leading zeroes en a
    cout << __builtin_clz(a)<<endl;
    //contar 9,trailling zeroes en a
    cout << __builtin_ctz(a)<<endl;
}
//--------------------EOSOLUTION--------------------------------
```

9 Testing