First to Penalty

-12

Contents

1   Template

```
1  #include "bits/stdc++.h"
2  //assert(x>0) si falla da RTE
3  using namespace std;
4  #define endl '\n'
5  #define DBG(x) cerr<<#x<< "=" << (x) << endl;
6  #define RAYA cerr<<"===================="<<endl;
7  #define RAYAS cerr<<"...................."<<endl;
8  //#define DBG(x) ;
9  //#define RAYA ;
10 //#define RAYAS ;
11
12 //-------------------SOLBEGIN----------------------------
13 int main() {
14   ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
15   int tC;
16
17   cin >> tC;
18   while (tC--) {
19
20   }
21
22 }
23 //-------------------EOSOLUTION--------------------------
```

2   Data structures

2.1   Simplified DSU (Stolen from GGDem)

```
1  int uf[MAXN];
2  void uf_init(){memset(uf,-1,sizeof(uf));}
3  int uf_find(int x){return uf[x]<0?x:uf[x]=uf_find(uf[x]);}
4  bool uf_join(int x, int y){
5    x=uf_find(x);y=uf_find(y);
6    if(x==y)return false;
7    if(uf[x]>uf[y])swap(x,y);
8    uf[x]+=uf[y];uf[y]=x;
9    return true;
10 }
```

2.2   Disjoint Set Union

```
1  class disjSet {
2    int* sz;
3    int* par;
4  public:
5    int len;
6    disjSet(int tam){
7        sz = new int[tam + 4]();
8        par = new int[tam + 4]();
9        len = 0;
10       for(int i = 0; i<=tam; i++){
11           par[i] = i;
12           sz[i] = 1;
13           len++;
14       }
15   }
16   int finds(int el){
17       if (el == par[el]) return el;
18       return par[el] = finds(par[el]);
19   }
20   void unions(int a, int b){
21       a = finds(a);
22     b = finds(b);
23       if (a == b) return;
24       len--;
25       //se hace que el gde sea padre del pequeno
26       if (sz[a] > sz[b]) swap(a,b);
27       par[a] = b;
28       sz[b] += sz[a];
29   }
30   ~disjSet(){
31       delete[] size;
32       size = nullptr;
33       delete[] parent;
34       parent = nullptr;
35   }
36 };
```

2.3   Segment tree

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

```
5   long long int build(int sti,int csize){
6       if(csize == 1) return st[sti];
7       return st[sti] = fst(build(sti*2+1,csize/2),build(sti*2+2,csize/2));
8   }
9   void innit(){
10      for(int i = 0; i<stsize; i++) st[i] = neut;
11      /*int d = 0;
12      for(int i = stsize-n; i<stsize && d<n; i++){
13          st[i] = arr[d];d++;
14      }*/
15      build(0,n);
16  }
17  void upd(int ind, long long int val){
18      ind = stsize-n+ind;
19      st[ind] = val;ind--;ind/=2;
20      while(true){
21          st[ind] = fst(st[ind*2+1],st[ind*2+2]);
22          ind--;
23          if(ind<0) break;
24          ind/=2;
25      }
26  }
27  long long int rqu(int l, int r,int sti, int ls, int rs){
28      if(l<=ls && rs<= r) return st[sti];
29      if(r<ls || l>rs) return neut;
30      int m = (rs+ls)/2;
31      return fst(rqu(l,r,sti*2+1,ls,m),rqu(l,r,sti*2+2,m+1,rs));
32  }
33  long long int query(int l, int r){
34      return rqu(l,r,0,0,n-1);
35  }
36  //uso, inicializa neut, determina n (asegurate que sea una potencia de
        2), define fst para determinar
37  //la opracion del segment tree
```

2.4   Segment tree Lazy

```
1   //MAXN = 2^k, n = tam arreglo inicial
2   #define MAXN 524288
3   vector<int> arr;
4   int stsize; long long int neut;int n;
5   long long int* st = new long long int[2*MAXN-1]();
6   long long int* pendientes = new long long int[2*MAXN-1]();
```

```
7   long long int fst(long long int a, long long int b){return a+b;}
8   long long int build(int sti,int csize){
9       if(csize == 1) return st[sti];
10      return st[sti] = fst(build(sti*2+1,csize/2),build(sti*2+2,csize/2));
11  }
12  bool hasChildren(int sti){sti*=2;sti++;sti++;return sti<stsize;}
13  void innit(){
14      for(int i = 0; i<stsize; i++) st[i] = neut;
15      int d = 0;
16      for(int i = stsize-n-1; i<stsize && d<n; i++) {st[i] = arr[d];d++;}
17      build(0,n);
18  }
19  void updrec(int l,int r, int sl, int sr,int sti, long long int val){
20      if(sr<l || r< sl) return;
21      if(l<= sl && sr <=r){
22          st[sti] += val*(sr-sl+1);
23          if(hasChildren(sti)){pendientes[sti*2+1]+=val;pendientes[sti
                *2+2]+=val;}
24          return;
25      }
26
27      int sm = (sl+sr)/2;
28      updrec(l,r,sl,sm,sti*2+1,val);
29      updrec(l,r,sm+1,sr,sti*2+2,val);
30      st[sti] = fst(st[sti*2+1],st[sti*2+2]);
31  }
32  void upd(int l, int r, long long int val){updrec(l,r,0,n-1,0,val);}
33
34  long long int rqu(int l, int r,int sti, int ls, int rs){
35      if(r<ls || l>rs) return neut;
36      if(l<=ls && rs<= r){
37          return st[sti]+pendientes[sti]*(rs-ls+1);
38      }
39
40      st[sti] += pendientes[sti]*(rs-ls+1);
41      if(hasChildren(sti)){pendientes[sti*2+1]+=pendientes[sti];pendientes
            [sti*2+2]+=pendientes[sti];}
42      pendientes[sti] = 0;
43
44      int m = (rs+ls)/2;
45      return fst(rqu(l,r,sti*2+1,ls,m),rqu(l,r,sti*2+2,m+1,rs));
46  }
47  long long int query(int l, int r){
```

```
48      return rqu(l,r,0,0,n-1);
49  }
50  //uso, inicializa neut, lee n y arr, iguala n a la potencia de dos mas
        cercana y mayor
51  //determina stsize = 2*n (asegurate que sea una potencia de 2), define
        fst para determinar
52  //la opracion del segment tree
```

2.5   Trie

```
1   struct triver {
2       char alphabet;
3       bool ter;
4       vector<triver*> child;
5       triver(char a): alphabet(a) { child.assign(26, NULL); ter = false; }
6   };
7   class trie{
8   private:
9       triver* root;
10  public:
11      trie() { root = new triver('!');}
12      void insert(string s){
13          triver* curr = root;
14          for(char l: s){
15              if(curr->child[l-'A'] == NULL) curr->child[l-'A'] =  new
                    triver(l);
16              curr = curr->child[l-'A'];
17          }
18          curr->ter = true;
19      }
20      bool search(string s){
21          triver* curr = root;
22          for(char l: s){
23              if(curr == NULL) break;
24              curr = curr->child[l-'A'];
25          }
26          return !(curr == NULL);
27      }
28  };
```

3   Graphs

3.1   Graph Transversal

3.1.1   BFS

```
1   #define GS 400040
2   vector<int> graph[GS];
3   bitset <GS> vis;
4   //anchura O(V+E)
5   void dfs(int curr) {
6     queue<int> fringe;
7     fringe.push(curr);
8     while (fringe.size()) {
9       curr = fringe.front(); fringe.pop();
10      if (!vis[curr]) {
11        vis[curr] = 1;
12        for (int h : graph[curr]) fringe.push(h);
13      }
14    }
15  }
```

3.1.2   DFS

```
1   #define GS 400040
2   vector<int> graph[GS];
3   bitset <GS> vis;
4   //profundidad O(V+E)
5   void dfs(int curr) {
6     stack<int> fringe;
7     fringe.push(curr);
8     while (fringe.size()){
9       curr = fringe.top(); fringe.pop();
10      if (!vis[curr]) {
11        vis[curr] = 1;
12        for (int h : graph[curr]) fringe.push(h);
13      }
14    }
15  }
```

3.2   Topological Sort

```
1   #define GS 400040
2   vector<int> graph[GS];
```

```
3   bitset <GS> vis;
4   vector<int> topsort;
5   int e,n;
6   //profundidad
7   //O(N+E)
8   //Solo funciona con DAG's, no existe un top sort de un grafo Non-DAG
9   void todfs(int pa) {
10    vis[pa]=1;
11    for(int h: graph[pa]){if(!vis[h]){todfs(h);}}
12    topsort.push_back(pa);
13  }
14  void topologicalSort(){
15    vis.reset();
16    topsort.clear();
17    for(int i = 0; i<n; i++){if(!vis[i]){dfs(i);}}
18    reverse(topsort.begin(),topsort.end());
19  }
```

3.3   APSP: Floyd Warshall

```
1   #define GS 1000
2   #define INF 100000000
3   //destino, costo
4   int graph[GS][GS];
5   //All Pairs Dist
6   int dist[GS][GS];
7   //Toma en cuenta nodos [0-tam] inclusivo, modificar de acuerdo a las
        necesidades
8   //Ten cuidado con el valor que le pones a INF, puede provocar overflows
        o puede no ser lo suficientemente grande.
9   void Floyd_Warshall(int tam){
10    for(int i = 0; i<=tam; i++)
11      for(int f = 0; f<=tam; f++)
12        dist[i][f] = INF;
13
14    for(int i = 0; i<=tam; i++)
15      for(int f = 0; f<=tam; f++)
16        dist[i][f] = graph[i][f];
17
18    //para reconstruir el camino solo basta con guardar intermedio como
          el padre de ini si el cambio se hizo, -1 otherwise
19    for(int intermedio = 0; intermedio<=tam; intermedio++)
20      for(int ini = 0;  ini<=tam; ini++)
21        for(int fin = 0; fin<=tam; fin++)
22          dist[ini][fin] = min(dist[ini][fin],dist[ini][intermedio
              ]+dist[intermedio][fin]);
23  }
```

3.4   SSSP

3.4.1   Lazy Dijkstra

```
1   #define GS 1000
2   #define INF 100000000
3   //destino, costo
4   vector<pair<int,int>> graph[GS];
5   int dist[GS];
6   void dijkstra(int origen,int tam){
7     for(int i = 0; i<=tam; i++){
8       dist[i] = INF;
9     }
10    priority_queue<pair<int,int>,vector<pair<int,int>>, greater<pair<int
          ,int>>> pq;
11    pair<int,int> curr;
12
13    pq.push(make_pair(0,origen));
14
15    while(pq.size()){
16      curr = pq.top();pq.pop();
17      if(curr.first >= dist[curr.second]) continue;
18
19      dist[curr.second] = curr.first;
20      for(pair<int,int> h: graph[curr.second]){
21        if((h.second+curr.first)<dist[h.first]) pq.push({h.second+
              curr.first,h.first});
22      }
23    }
24  }
25  //Esta es la implementacion huevona
26  //Resuelve Single Source Shortest Paths con aristas positivas
27  //Como es la lazy implementation, si funciona con edges negativos
        siempre y cuando no hayan ciclos negativos
28  //Si hay ciclos negativos se va atascar en un ciclo infinito
29  //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

```
1   //esta es la implementacion huevona
2   #define GS 1000
3   //cuidado con overflows!!
4   #define INF 100000000
5   #define NINF -100000000
6   //destino, costo
7   vector<pair<int,int>> graph[GS];
8   int dist[GS];
9   struct edge{
10      int from,to,cost;
11  };
12  //Corre en O(VE)
13  void bellmanFord(int origen,int tam){
14      for(int i = 0; i<=tam; i++){
15          dist[i] = INF;
16      }
17      dist[origen] = 0;
18      edge aux;
19      vector<edge> aristas;
20      bool optimal;
21
22      for(int i = 0; i<=tam; i++){
23          for(pair<int,int> h: graph[i]){
24              aux.from = i; aux.to  = h.first;aux.cost = h.second;
25              aristas.push_back(aux);
26          }
27      }
28
29      //Si se relajan todos las aristas V-1 veces en un orden arbitrario
30      //Se asegura que la distancia optima para cada vertice sera
            alcanzada
31      for(int i = 0; i<tam && !optimal; i++){
32          optimal = true;
33          for(edge elem: aristas){
34              if(dist[elem.from] + elem.cost < dist[elem.to]){
35                  dist[elem.to] = dist[elem.from] + elem.cost;
36                  //si algun vertice fue actualizado significa que puede
                        que
37                  //las distancias aun no sean optimas
38                  optimal = false;
39              }
40          }
41      }
42
43      //Se corre de nuevo para asegurar encontrar todos los ciclos
            negativos
44      for(int i = 0; i<tam && !optimal; i++){
45          optimal = true;
46          for(edge elem: aristas){
47              if(dist[elem.from] + elem.cost < dist[elem.to]){
48                  //Si aun despues de correr V-1 veces se puede actualizar
49                  //Significa que esta en un ciclo negativo
50                  dist[elem.to] = NINF;
51                  //si algun vertice fue actualizado significa que puede
                        que
52                  //las distancias aun no sean optimas
53                  optimal = false;
54              }
55          }
56      }
57
58  }
```

3.5   Strongly Connected Components: Kosaraju

```
1   #define GS 2010
2   vector<int> graph[GS];
3   vector<int> graphI[GS];
4   vector<int> orden;
5   bitset<GS> vis;
6
7   void invertirGrafo(int n){
8       for(int p = 1;p<= n; p++)
9           for(int h: graph[p])graphI[h].push_back(p);
10  }
11  void obtOrd(int p,int n){
12      vis[p] = 1;
13      for(int h: graph[p]){
14          if(!vis[h] && h<=n) obtOrd(h,n);
15      }
16      orden.push_back(p);
17  }
18  int findSCC(int n){
19      int res = 0;
20      invertirGrafo(n);
21      orden.clear();
```

```
22      for(int i = 1; i<=n; i++) vis[i] =0;
23      for(int i = 1; i<=n; i++) if(!vis[i]) obtOrd(i,n);
24      reverse(orden.begin(),orden.end());
25      //cuenta los connected components
26      //vector<int> lscc;
27      stack<int> fringe;
28      int curr;
29      for(int i = 1; i<=n; i++) vis[i] =0;
30      for(int i: orden){
31          //lscc.clear();
32          if(!vis[i]){
33              fringe.push(i);
34              while (fringe.size()){
35                  curr = fringe.top();fringe.pop();
36                  //lscc.push_back(curr);
37                  if (!vis[curr]) {
38                      vis[curr] = 1;
39                      for (int h : graphI[curr]) fringe.push(h);
40                  }
41              }
42              res++;
43          }
44          //hacer lo que sea con lcss
45      }
46      return res;
47  }
48
49  //OJO esto solo jala con directed graphs
50  //por definicion todas las undirected graphs tienen un solo SCC
51  //NOTAR QUE LOS GRAFOS QUE USA CUMPLEN CON: 0<=VERTICE<=n
```

3.6   Articulation Points and Bridges: ModTarjan

```
1  #define GS 50
2  vector<int> graph[GS];
3  bitset<GS> vis, isArtic;
4  vector<int> padre;
5  //id por tiempo, menor id accesible
6  //ya sea por descendientes o por back edges
7  vector<int> tId,lId;
8  //cantidad de hijos que tiene en el bfs spanning tree
9  int rootChildren;
10 int cnt;
11 int dfsRoot;
12 void findAP_B(int p){
13     cnt++;vis[p] = 1;tId[p] = cnt;lId[p] = tId[p];
14
15     for(int hijo: graph[p]){
16         if(!vis[hijo]){
17             padre[hijo] = p;
18             if(p == dfsRoot) rootChildren++;
19
20             findAP_B(hijo);
21
22             //esto significa que ni por un back edge el hijo accede al
                   padre
23             //por lo que si el padre fuese eliminado el hijo quedaria
                   aislado
24             if(lId[hijo] >= tId[p]) isArtic[p] = 1;
25             if(lId[hijo] > tId[p]){
26                 //esto significa que si se eliminase el camino de padre
                       ->hijo
27                 //se lograria desconectar el grafo, aka bridge
28             }
29             lId[p] = min(lId[p],lId[hijo]);
30         }else{
31             //si hay un ciclo indirecto, actualiza el valor para el
                   padre
32             if(hijo != padre[p]) lId[p] = min(lId[p],tId[hijo]);
33         }
34     }
35 }
36 //OJO esto solo jala con Undirected graphs
37 /*
38     MAIN
39     for(int i = 0; i<n; i++){
40         if(!vis[i]){
41             rootChildren = 0;
42             dfsRoot = i;
43             findAP_B(i);
44             //el algoritmo no puede detectar si el nodo que lo origino
45             //es un articulation point, por lo que queda checar si
46             //en el spanning tree que genero tiene mas de un solo hijo
47             isArtic[i] = (rootChildren>1?1:0);
48         }
49     }
```

```
50  */
```

4   Math

4.1   Identities

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

(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

```cpp
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)

```cpp
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);
```

```cpp
        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 = gcd(b,inf,x,y);
    if(d!=1) return -1;
    return ((x%inf)+inf)%inf;
}
```

4.4   Modular Binomial Coeficient and Permutations

```cpp
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

```cpp
//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;
```

```cpp
     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

```cpp
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

```cpp
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;
}
```

```
25  //Matrix mat(n,vector<long long int>(n));
26
```

5   Geometry

6   Strings

6.1   Explode by token

```
1   //#include <sstream>
2
3   vector<string> explode(string const& s, char delim) {
4     vector<string> result;
5     istringstream iss(s);
6     for (string token; getline(iss, token, delim); )
7     {
8       result.push_back(move(token));
9     }
10    return result;
11  }
```

6.2   Multiple Hashings DS

```
1   struct multhash{
2       unsigned long long int h1,h2;
3       unsigned long long int alf[257];
4       bool operator < (multhash b) const {
5       if (h1 != b.h1) return h1 < b.h1;
6       return h2 < b.h2;
7       }
8     bool operator == (multhash b) const { return (h1== b.h1 && h2== b.h2)
            ;}
9     bool operator != (multhash b) const { return !(h1== b.h1 && h2== b.h2)
            ;}
10  public:
11      string s;
12      multhash(){
13          h1 = 0; h2 = 0;s = "";
14          for(char l = 'a'; l<='z'; l++) alf[l] = l-'a'+1;
15      }
16      void innit(){
17          unsigned long long int inf,p,op;
18
19          inf = 999727999;
20          p = 325255434;op = 325255434;
21          for(char l: s){
22              h1+=(p*alf[l])%inf;
23              p*=op;
24              p%=inf;
25          }
26
27          inf = 1070777777;
28          p = 10018302;op = 10018302;
29          for(char l: s){
30              h2+=(p*alf[l])%inf;
31              p*=op;
32              p%=inf;
33          }
34      }
35  };
36  //VALORES ALTERNATIVOS DE INF, LOG 17
37  //666666555557777777
38  //986143414027351997
39  //974383618913296759
40  //973006384792642181
41  //953947941937929919
42  //909090909090909091
43  //VALORES PARA P, USAR PRIMOS MAYORES A |Alfabeto|
44  //31,47,53,61,79
```

6.3   Permute chars of string

```
1   void permute(string str){
2     // Sort the string in lexicographically
3     // ascennding order
4     sort(str.begin(), str.end());
5
6     // Keep printing next permutation while there
7     // is next permutation
8     do {
9       cout<<str<<endl;
10    } while (next_permutation(str.begin(), str.end()));
11  }
```

6.4   Longest common subsequence

```
1   //O(|te|*|pa|)
```

```
2  //cambiar score para otros problemas, str all match = +2, miss/ins/del =
        -1
3  //usar char que no este en el alfabeto para denotar del/ins
4  string te,pa;
5  long long int ninf = -10e13;
6  long long int score(char a, char b){
7      if(a=='*' || b=='*') return 0;
8      if(a==b) return 1;
9      return ninf;
10 }
11 long long int lcs(){
12     long long int** dp;te = "*"+te; pa = "*"+pa;
13     long long int res = 0;
14
15     dp = new long long int*[te.size()];
16     for(int i = 0; i<te.size(); i++) dp[i] = new long long int[pa.size()
          ]();
17
18     for(int r = 1; r<te.size(); r++){
19         for(int c = 1; c<pa.size(); c++){
20             dp[r][c] = dp[r-1][c-1]+score(te[r],pa[c]);
21             dp[r][c] = max(dp[r][c-1]+score(te[r],'*'),dp[r][c]);
22             dp[r][c] = max(dp[r-1][c]+score('*',pa[c]),dp[r][c]);
23         }
24     }
25
26     return dp[te.size()-1][pa.size()-1];
27 }
```

6.5   KMP

```
1  string T,P;
2  int bt[MAXN];
3  //O(|Text|+|Pattern|)
4  void KMPpre(){
5      int i = 0,j = -1; bt[0] = -1;
6      while(i<P.size()){
7          while(j>=0 && P[i]!=P[(j>=0?j:0)]) j = bt[j];
8          i++;j++; bt[i] = j;
9      }
10 }
11 int kmp(){
12     int res =0, i = 0, j = 0;
```

```
13     while(i<T.size()){
14         while(j>=0 && T[i] != P[(j>=0?j:0)])  j = bt[j];
15         i++; j++;
16         if(j==P.size()){//match, do anything
17             res++;j = bt[j];
18         }
19     }
20     return res;
21 }
```

7   Flow

8   Miscellaneous

8.1   Bit Manipulation

```
1  #include "bits/stdc++.h"
2  using namespace std;
3  #define endl '\n'
4
5
6  int main() {
7    ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
8    //Se representan bitmasks de 30 a 62 bits
9    //usando signed int y signed long long int
10   //para evitar problemas con el complemento de dos
11   signed int a, b;
12   //para multiplicar un numero por dos solo es necesario aplicar un
13   // shifteo de sus bits a la izquierda
14   a = 1;
15   a= a << 3;
16   cout << a << endl;
17   //para dividir un numero entre dos es necesario aplicar un
18   //shifteo a la derecha
19   a = 32;
20   a = a >> 3;
21   cout << a << endl;
22   //para encender el bit n de a, solo hay que igualar a = a | pow(2,n-1)
23   //prende el tercer bit
24   a = 1;
25   b = 1 << 2;
26   a = a | b;
27   cout << a << endl;
```

```cpp
28    //para apagar el bit n de a, solo hay que  a &= ~pow(2,n-1)
29    //prende el tercer bit
30    a = 5;
31    b = 1 << 2;
32    a &= ~b;
33    cout << a << endl;
34    //para revisar si el bit n de a esta encendido
35    //revisa si el tercer bit esta encendido
36    a = 5;
37    b = 1 << 2;
38    a = a & b;
39    cout << (a?"SI":"NO") << endl;
40    //para volter el bit n de a, solo hay que igualar a = a ^ pow(2,n-1)
41    //apaga el tercer bit
42    a = 5;
43    b = 1 << 2;
44    a = a ^ b;
45    cout << a << endl;
46    //para obtener el bit menos significativo que esta encendido a& -a
47    a = 12;
48    cout << log2(a & ((-1) * a))+1 << endl;
49    //para prender todos los bits hasta n
50    a = (1<<4)-1;
51    cout << a << endl;
52 }
53 //-------------------EOSOLUTION------------------------------
```

```cpp
1  #include "bits/stdc++.h"
2  using namespace std;
3  #define endl '\n'
4  #pragma GCC optimize("O3")
5  #pragma GCC target("popcnt")
6
7  //no usar con visual c++
8  //solo con g++ like compilers
9  int main() {
10   ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
11   signed long long int a, b, n;
12   //Obtain the remainder (modulo) of a when it is divided by n (n is a
          power of 2)
13   a = 15; n = 8-1;
14   a &= n;
15   cout << "a%n, a = 15, n = 2^3" << endl;
16   cout << a << endl;
17   //Apaga el bit menos significativo de a
18   a = 14;
19   b = (a & ((-1) * a));
20   a &= ~b;
21   cout << a << endl;
22   //enciende el ultimo cero de a
23   a = 9;
24   b = ~a;
25   b = (b & ((-1) * b));
26   a = a | b;
27   cout << a<<endl;
28   //contar bits encendidos en a
29   cout << __builtin_popcount(a)<<endl;
30   //checar la paridad de a
31   cout << (__builtin_parity(a) ? "IMPAR" : "PAR") << endl;
32   //contar leading zeroes en a
33   cout << __builtin_clz(a)<<endl;
34   //contar 9,trailling zeroes en a
35   cout << __builtin_ctz(a)<<endl;
36 }
37 //-------------------EOSOLUTION-------------------------------
```

9 Testing