

# First to Penalty



## Contents

<b>1 Template</b>	<b>2</b>	<b>4.5 Non-Mod Binomial Coefficient and Permutations</b>	<b>11</b>
<b>2 Data structures</b>	<b>2</b>	<b>4.6 Modular Catalan Numbers</b>	<b>11</b>
2.1 Simplified DSU (Stolen from GGDem)	2	<b>4.7 Ceil Fraccionario</b>	<b>11</b>
2.2 Disjoint Set Union	2	<b>4.8 Numeros de Fibonacci</b>	<b>11</b>
2.3 Fenwick tree	3	<b>4.9 Sieve Of Eratosthenes</b>	<b>11</b>
2.4 Segment tree	3	<b>4.10 Sieve-based Factorization</b>	<b>11</b>
2.5 Segment tree Lazy	4	<b>4.11 Cycle Finding</b>	<b>12</b>
2.6 Trie	4	<b>4.12 Berlekamp Massey</b>	<b>12</b>
<b>3 Graphs</b>	<b>5</b>	<b>4.13 Modular Berlekamp Massey</b>	<b>12</b>
3.1 Graph Transversal	5	<b>4.14 Matrix exponentiation</b>	<b>13</b>
3.1.1 BFS	5	<b>4.15 Ecuaciones Diofantinas</b>	<b>13</b>
3.1.2 DFS	5	<b>4.16 Pollard-Rho, Stolen from GGDem</b>	<b>14</b>
3.2 Topological Sort	5	<b>4.17 FFT, Stolen from GGDem</b>	<b>14</b>
3.3 APSP: Floyd Warshall	6	<b>4.18 Euler Totient Function</b>	<b>16</b>
3.4 SSSP	6	<b>5 Geometry</b>	<b>16</b>
3.4.1 Lazy Dijkstra	6	<b>6 Strings</b>	<b>16</b>
3.4.2 Bellman-Ford	6	6.1 Explode by token	16
3.5 Strongly Connected Components: Kosaraju	7	6.2 Multiple Hashings DS	16
3.6 Articulation Points and Bridges: ModTarjan	8	6.3 Permute chars of string	17
3.7 Kth-Ancestor using Binary Lifting	8	6.4 Longest common subsequence	17
3.8 LCA using Binary Lifting	9	6.5 KMP	17
<b>4 Math</b>	<b>9</b>	6.6 Suffix Array	18
4.1 Identities	9	6.7 STL Suffix Array	19
4.2 Binary Exponentiation and modArith	10	<b>7 Clasicos</b>	<b>21</b>
4.3 Modular Inverse (dividir mod)	10	7.1 Job scheduling	21
4.4 Modular Binomial Coefficient and Permutations	10	7.1.1 One machine, linear penalty	21
		7.1.2 One machine, deadlines	21
		7.1.3 One machine, profit	21
		7.1.4 Two machines, min time	22
		<b>8 Flow</b>	<b>22</b>
		8.1 Dinic, thx GGDem	22
		<b>9 Miscellaneous</b>	<b>23</b>
		9.1 pbds	23
		9.2 Bit Manipulation	23
		<b>10 Testing</b>	<b>25</b>
		10.1 Gen and AutoRun testcases	25
		10.1.1 Gen.cpp	25
		10.1.2 Stress testing	25

10.1.3 Autorun . . . . .	25
10.2 Highly Composite Numbers . . . . .	25

# 1 Template

```

1  #include "bits/stdc++.h"
2  //assert(true/false) false leads to a 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 void solve(){
12
13 }
14 //-----SOLBEGIN-----
15 int main() {
16     ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
17     int tC; cin>>tC;
18
19     while(tC--) solve();
20
21 }
22 //-----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 Fenwick tree

```

1  //Fenwick tree, solo jala si la funcion cumple con ser:
2  //una binary associative function over a set with identity element and
   inverse elements
3  // incluyendo pero no limitandose a range sum
4  //define neutro y llena arr
    
```

```

5  #define MAXN 500010
6  long long int neutro;
7  vector<long long int> arr;
8  long long int fenwick[MAXN];
9
10 int get_low(int ind){return (ind&(ind+1));}
11 int get_upp(int ind){return (ind|(ind+1));}
12 long long int get_sum(int r){
13     if(r<0) return neutro;
14     return fenwick[r]+get_sum(get_low(r)-1);
15 }
16 long long int get_sum(int l, int r){return get_sum(r)-get_sum(l-1);}
17 void build(){
18     int size = arr.size();
19     for(int i = 0; i<size; i++ ) fenwick[i] = neutro;
20     for(int i = 0; i<size; i++){
21         fenwick[i]+=arr[i];
22         if(get_upp(i)<size)fenwick[get_upp(i)]+=fenwick[i];
23     }
24 }
25 void add(int ind, long long int d){
26     for(;ind<arr.size(); ind = get_upp(ind)) fenwick[ind]+=d;
27 }
28 //no funciona con range queries
29 void range_add(int l, int r, int d){add (l,d); add(r+1,-d);}
    
```

## 2.4 Segment tree

```

1  //MAXN = 2^k, n = tam arreglo inicial
2  #define MAXN 262160
3  int stsize; long long int neut;int n;
4  long long int* st = new long long int[2*MAXN-1]();
5  long long int fst(long long int a, long long int b);
6  long long int build(int sti,int csize){
7      if(csize == 1) return st[sti];
8      return st[sti] = fst(build(sti*2+1,csize/2),build(sti*2+2,csize/2));
9  }
10 void innit(){
11     for(int i = 0; i<stsize; i++) st[i] = neut;
12     /*int d = 0;
13     for(int i = stsize-n; i<stsize && d<n; i++){
14         st[i] = arr[d];d++;
15     }*/
    
```

```

16     build(0,n);
17 }
18 void upd(int ind, long long int val){
19     ind = stsize-n+ind;
20     st[ind] = val;ind--;ind/=2;
21     while(true){
22         st[ind] = fst(st[ind*2+1],st[ind*2+2]);
23         ind--;
24         if(ind<0) break;
25         ind/=2;
26     }
27 }
28 long long int rqu(int l, int r,int sti, int ls, int rs){
29     if(l<=ls && rs<= r) return st[sti];
30     if(r<ls || l>rs) return neut;
31     int m = (rs+ls)/2;
32     return fst(rqu(l,r,sti*2+1,ls,m),rqu(l,r,sti*2+2,m+1,rs));
33 }
34 long long int query(int l, int r){
35     return rqu(l,r,0,0,n-1);
36 }
37 //uso, inicializa neut, n = primera potencia de 2 >= n del problema,
38     stsize = 2*n-1
39 //llena arr de neutros hasta que su tam sea el nuevo n
40 //DEFINE LA FUNCION fst

```

## 2.5 Segment tree Lazy

```

1 //MAXN = 2^k, n = tam arreglo inicial
2 #define MAXN 262160
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; 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
24             *2+2]+=val;}
25         return;
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]+pendientes[sti*2+1],st[sti*2+2]+pendientes
31         [sti*2+2]);
32 }
33 void upd(int l, int r, long long int val){updrec(l,r,0,n-1,0,val);}
34
35 long long int rqu(int l, int r,int sti, int ls, int rs){
36     if(r<ls || l>rs) return neut;
37     if(l<=ls && rs<= r){
38         return st[sti]+pendientes[sti]*(rs-ls+1);
39     }
40     st[sti] += pendientes[sti]*(rs-ls+1);
41     if(hasChildren(sti)){pendientes[sti*2+1]+=pendientes[sti];pendientes
42         [sti*2+2]+=pendientes[sti];}
43     pendientes[sti] = 0;
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, n = primera potencia de 2 >= n del problema,
51     stsize = 2*n-1
52 //llena arr de neutros hasta que su tam sea el nuevo n
53 //DEFINE LA FUNCION fst

```

## 2.6 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         if(curr == NULL) return false;
27         return curr->ter;
28     }
29 };

```

## 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
31     //alcanzada
32     for(int i = 0; i<tam && !optimal; i++){
33         optimal = true;
34         for(edge elem: aristas){
35             if(dist[elem.from] + elem.cost < dist[elem.to]){
36                 dist[elem.to] = dist[elem.from] + elem.cost;
37                 //si algun vertice fue actualizado significa que puede
38                 //que
39                 //las distancias aun no sean optimas
40                 optimal = false;
41             }
42         }
43
44         //Se corre de nuevo para asegurar encontrar todos los ciclos
45         //negativos
46         for(int i = 0; i<tam && !optimal; i++){
47             optimal = true;
48             for(edge elem: aristas){
49                 if(dist[elem.from] + elem.cost < dist[elem.to]){
50                     //Si aun despues de correr V-1 veces se puede actualizar
51                     //Significa que esta en un ciclo negativo
52                     dist[elem.to] = NINF;
53                     //si algun vertice fue actualizado significa que puede

```

```

52         que
53         //las distancias aun no sean optimas
54         optimal = false;
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> lsc;
27     stack<int> fringe;
28     int curr;
29     for(int i = 1; i<=n; i++) vis[i] =0;
30     for(int i: orden){
31         //lsc.clear();
32         if(!vis[i]){

```

```

33     fringe.push(i);
34     while (fringe.size()){
35         curr = fringe.top();fringe.pop();
36         //lsc.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 lcsc
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 */
    
```

### 3.7 Kth-Ancestor using Binary Lifting

```

1  #define GS 100
2  //>log2(GS)
3  #define MAXANC 8
4  vector<int> graph[GS];
5  //NODO, 2**i ancestro
6  //inicializar todo en -1
7  int ancestro[GS][MAXANC];
    
```



```

8
9 //preprocesamiento, asume que graph es direccionado y rooteado
10 //agregar un bitset vis en caso de que falte
11 void buildAncestry(int curr,int h){
12     int ub = 31-__builtin_clz(h|0);
13     if(h==0) ub = 0;
14     for(int i = 1; i<=ub; i++){
15         ancestro[curr][i] = ancestro[ancestro[curr][i-1]][i-1];
16
17         for(int hijo: graph[curr]){
18             ancestro[hijo][0] = curr;
19             buildAncestry(hijo,h+1);
20         }
21     }
22
23 int kthAncestor(int curr, int k){
24     if(k==0) return curr;
25     int ub = 31-__builtin_clz(k);
26     if(ancestro[curr][ub] == -1) return -1;
27     return kthAncestor(ancestro[curr][ub],((1<<ub)^k));
28 }
    
```

### 3.8 LCA using Binary Lifting

```

1 //https://judge.yosupo.jp/problem/lca
2 #define GS 500000
3 //>log2(GS)
4 #define MAXANC 19
5 vector<int> graph[GS];
6 //NODO, 2**i ancestro
7 int ancestro[GS][MAXANC];
8 int dist[GS];
9 //preprocesamiento, asume que graph es direccionado y rooteado
10 //agregar un bitset vis en caso de que falte
11 void buildAncestry(int curr,int h){
12     dist[curr] = h;
13     int ub = 31-__builtin_clz(h|0);
14     if(h==0) ub = 0;
15     for(int i = 1; i<=ub; i++){
16         ancestro[curr][i] = ancestro[ancestro[curr][i-1]][i-1];
17
18         for(int hijo: graph[curr]){
19             ancestro[hijo][0] = curr;
    
```

```

20         buildAncestry(hijo,h+1);
21     }
22 }
23
24 int kthAncestor(int curr, int k){
25     if(k==0) return curr;
26     int ub = 31-__builtin_clz(k);
27     if(ancestro[curr][ub] == -1) return -1;
28     return kthAncestor(ancestro[curr][ub],((1<<ub)^k));
29 }
30
31 int lca(int a,int b){
32     int d = min(dist[a],dist[b]);
33     a = kthAncestor(a,dist[a]-d);
34     b = kthAncestor(b,dist[b]-d);
35     //encuentra el primer true
36     int l = 0,r = d,m;
37     while(l<r){
38         m = l+r; m/=2;
39         if(kthAncestor(a,m) == kthAncestor(b,m)) r = m;
40         else l = m+1;
41     }
42     return kthAncestor(a,l);
43 }
    
```

## 4 Math

### 4.1 Identities

#### Coefficientes binomiales.

$$(a+b)^n = \sum_{k=0}^n \binom{n}{k} a^{n-k} b^k$$

$$\binom{n}{k} = \binom{n}{n-k}$$

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

$$k \binom{n}{k} = n \binom{n-1}{k-1}$$

$$\sum_{k=0}^n \binom{n}{k} = 2^n$$

$$\sum_{k=0}^n (-1)^k \binom{n}{k} = 0$$

$$\binom{n+m}{t} = \sum_{k=0}^t \binom{n}{k} \binom{m}{t-k}$$

$$\sum_{j=k}^n \binom{j}{k} = \binom{n+1}{k+1}$$

#### Numeros Catalanos.

$$C_n = \frac{2(2n-1)}{n+1} C_{n-1}$$

$$C_n = \frac{1}{n+1} \binom{2n}{n}$$

$$C_n \sim \frac{4^n}{n^{3/2} \sqrt{\pi}}$$

$\Sigma(n) = O(\log(\log(n)))$  (number of divisors of  $n$ )

$$F_{2n+1} = F_n^2 + F_{n+1}^2$$

$$F_{2n} = F_{n+1}^2 - F_{n-1}^2$$

$$\sum_{i=1}^n F_i = F_{n+2} - 1$$

$$F_{n+i}F_{n+j} - F_nF_{n+i+j} = (-1)^n F_i F_j$$

(Möbius Function)

0 if  $n$  is square-free

1 if  $n$  got even amount of distinct prime factors

0 if  $n$  got odd amount of distinct prime factors

(Möbius Inv. Formula)

Let  $g(n) = \sum_{d|n} f(d)$ , then  $f(n) = \sum_{d|n} d \mu\left(\frac{n}{d}\right)$ .

Permutaciones objetos repetidos

$$P(n, k) = \frac{P(n, k)}{n_1! n_2! \dots}$$

Separadores, Ecuaciones lineares  $a$  variables =  $b$

$$\binom{a}{b} = \binom{a+b-1}{b} = \binom{a+b-1}{a-1}$$

Teorema chino

sean  $\{n_1, n_2, \dots, n_k\}$  primos relativos

$$P = n_1 \cdot n_2 \cdot \dots \cdot n_k$$

$$P_i = \frac{P}{n_i}$$

$$x \cong a_1(n_1)$$

$$x \cong a_2(n_2) \dots x \cong a_k(n_k)$$

$$P_1 S_1 \cong 1(n_1) \text{ Donde } S \text{ soluciones.}$$

$$x = P_1 S_1 a_1 + P_2 S_2 a_2 \dots P_k S_k a_k$$

## 4.2 Binary Exponentiation and modArith

```

1 long long int inf = 10000000007;
2 //suma (a+b)%m
3 //resta ((a-b)%m+m)%m
4 //mult (a*b)%m
5 long long binpow(long long b, long long e) {
6     long long res = 1; b%=inf;
7     while (e > 0) {
8         if (e & 1) res = (res * b)%inf;
9         b = (b * b)%inf;
10        e >>= 1;
11    }
12    return res;
13 }
```

## 4.3 Modular Inverse (dividir mod)

```

1 long long int inf = 10000000007;
```

```

2 long long int gcd(long long int a, long long int b, long long int& x,
3   long long int& y) {
4     x = 1, y = 0;
5     long long int x1 = 0, y1 = 1, a1 = a, b1 = b;
6     while (b1) {
7         long long int q = a1 / b1;
8         tie(x, x1) = make_tuple(x1, x - q * x1);
9         tie(y, y1) = make_tuple(y1, y - q * y1);
10        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
11    }
12    return a1;
13 }
14 long long int modinverse(long long int b, long long int m){
15     long long int x,y;
16     long long int d = gcd(b,inf,x,y);
17     if(d!=1) return -1;
18     return ((x%inf)+inf)%inf;
19 }
```

## 4.4 Modular Binomial Coefficient and Permutations

```

1 long long int inf = 10000000007;
2 //cat[n] = bincoef(2*n,n)/(n+1), cat[0] = 1
3 class binCoef{
4     long long int lim;
5     long long int* fact;
6 public:
7     binCoef(long long int l){
8         lim = l; fact = new long long int[l+1];fact[0]= 1;
9         for(long long int i = 1; i<=l; i++) fact[i] = (fact[i-1]*i)%inf;
10    }
11    //perm = (fact[n] * modinverse(fac[n-k],inf)%inf;
12    long long int query(long long int n, long long int k){
13        if(n<k) return 0;
14        return (fact[n] * modinverse((fact[n-k]*fact[k])%inf,inf))%inf;
15    }
16 };

1 //Usar esto es 0(k)
2 long long int bincoef(long long int n, long long int k){
3     if(k == 0 || k==n) return 1;
4     if(2LL*k > n) return bincoef(n,n-k);
5     return ((n * bincoef(n-1,k-1))%inf *modinverse(k))%inf;
6 }
```

## 4.5 Non-Mod Binomial Coefficient and Permutations

```

1 //Solo usar con n<=20
2 //cat[n] = bincoef(2*n,n)/(n+1), cat[0] = 1
3 unsigned long long int bincoef(unsigned long long int n, unsigned long
  long int k){
4     if(n<k) return 0;
5     unsigned long long int num = 1, den= 1;
6     for(unsigned long long int i = (n-k)+1; i<=n; i++) num*=i;
7     for(unsigned long long int i = 2; i<=k; i++) den*=i;
8     //perm = return num;
9     return num/den;
10 }
```

## 4.6 Modular Catalan Numbers

```

1 long long int inf = 10000000007;
2 class catalan{
3     long long int* cat; long long int lim
4 public:
5     catalan(long long int l){
6         lim = l; cat = new long long int[l+10];cat[0] = 1;
7         for(long long int i = 0;i<=l; i++) cat[i+1] = (((((4LL*i+2)%inf)
          *cat[i])%inf) *modinverse(n+2))%inf;
8     }
9     long long int query(long long int n){ return cat[n];}
10 };
```

## 4.7 Ceil Fraccionario

```

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

## 4.8 Numeros de Fibonacci

```

1 //en caso de ser usados mod un m pequeno
2 //recordar que los numeros de fibonacci se repiten por lo menos cada m^2
3 //0(n)
4 unsigned long long int fib(int n){
5     unsigned long long int a = 1,b = 1,aux;
6     if(n<=2){
7         return 1;
8     }
9     for(int i = 3; i<=n; i++){
```

```

10     aux = a+b;
11     a = b;
12     b = aux;
13 }
14 return b;
15 }
```

```

1 const long long int inf = 10000000007;
2 unordered_map<long long int,long long int> Fib;
3 //0(log n) :DD
4 long long int fib(long long int n)
5 {
6     if(n<2) return 1;
7     if(Fib.find(n) != Fib.end()) return Fib[n];
8     Fib[n] = (fib((n+1) / 2)*fib(n/2) + fib((n-1) / 2)*fib((n-2) / 2)) %
        inf;
9     return Fib[n];
10 }
```

## 4.9 Sieve Of Eratosthenes

```

1 #define MAXN 10e6
2 class soe{
3 public:
4     bitset<MAXN> isPrime;
5     soe(){
6         for(int i = 3; i<MAXN; i++) isPrime[i] = (i%2);
7         isPrime[2] = 1;
8         for(int i = 3; i*i<MAXN; i+=2)
9             if(isPrime[i])
10                 for(int j = i*i; j<MAXN; j+=i)
11                     isPrime[j] = 0;
12     }
13 };
```

## 4.10 Sieve-based Factorization

```

1 #define MAXN 10e6
2 class soe{
3 public:
4     int smolf[MAXN];
5     soe(){
6         for(int i = 2; i<MAXN; i++) smolf[i] = (i%2==0?2:i);
7     }
```

```

8     for(int i = 3; i*i<MAXN; i+=2)
9         if(smolf[i]==i)
10             for(int j = i*i; j<MAXN; j+=i)
11                 smolf[j] = min(smolf[j],smolf[i]);
12     }
13 };
    
```

#### 4.11 Cycle Finding

```

1 void cyclef(long long int sem){
2     long long int hare = f(sem),tort=f(sem);hare = f(hare);
3     //liebre avanza dos pasos, tortuga solo uno
4     while(hare!=tort){
5         tort = f(tort); hare = f(f(hare));
6     }
7     //Se detiene en el inicio del ciclo
8     tort = sem;
9     while(hare!=tort){
10         tort = f(tort); hare = f(hare);
11     }
12
13     int len = 1;
14     tort = f(sem);
15     while(hare!=tort){
16         tort=f(tort);
17         len++;
18     }
19 }
    
```

#### 4.12 Berlekamp Massey

```

1 typedef long long int ll;
2 //Obtiene recurrencia lineal dados los primeros elementos en O(n^2)
3 vector<ll> berlekampMassey(const vector<ll> &s) {
4     vector<ll> c;
5     vector<ll> oldC;
6     int f = -1;
7     for (int i=0; i<(int)s.size(); i++) {
8         ll delta = s[i];
9         for (int j=1; j<=(int)c.size(); j++) delta -= c[j-1] * s[i-j];
10        if (delta == 0) continue;
11        if (f == -1) {
12            c.resize(i + 1);
    
```

```

13        mt19937 rng(chrono::steady_clock::now().time_since_epoch().
14            count());
15        for (ll &x : c) x = rng();
16        f = i;
17    } else {
18        vector<ll> d = oldC;
19        for (ll &x : d) x = -x;
20        d.insert(d.begin(), 1);
21        ll df1 = 0;
22        for (int j=1; j<=(int)d.size(); j++) df1 += d[j-1] * s[f+1-j];
23        assert(df1 != 0);
24        ll coef = delta / df1;
25        for (ll &x : d) x *= coef;
26        vector<ll> zeros(i - f - 1);
27        zeros.insert(zeros.end(), d.begin(), d.end());
28        d = zeros;
29        vector<ll> temp = c;
30        c.resize(max(c.size(), d.size()));
31        for (int j=0; j<(int)d.size(); j++) c[j] += d[j];
32        if (i - (int) temp.size() > f - (int) oldC.size()) {oldC = temp;f = i;}
33    }
34    return c;
35 }
    
```

#### 4.13 Modular Berlekamp Massey

```

1 typedef long long int ll;
2 long long int inf = 1000000007;
3 vector<ll> bermas(vector<ll> x){
4     vector<ll> ls,cur;
5     int lf,ld;
6     for(int i = 0; i<x.size(); i++){
7         long long int t = 0;
8         for(int j = 0; j<cur.size(); j++) t=(t+x[i-j-1]*(long long int)
9             cur[j])%inf;
10        if((t-x[i])%inf==0)continue;
11        if(cur.size()==0){cur.resize(i+1);lf=i;ld=(t-x[i])%inf;continue;
12        };
13        long long int k = (x[i]-t)*powermod(ld,inf-2)%inf;
14        vector<ll>c(i-lf-1);c.push_back(k);
    
```

```

13     for(int j = 0; j<ls.size(); j++) c.push_back(-ls[j]*k%inf);
14     if(c.size()<cur.size()) c.resize(cur.size());
15     for(int j = 0; j<cur.size();j++) c[j]=(c[j]+cur[j])%inf;
16     if(i-lf+ls.size()>=cur.size())ls=cur,lf=i,ld=(t-x[i])%inf;
17     cur=c;
18 }
19     for(int i =0; i<cur.size(); i++) cur[i]=(cur[i]%inf+inf)%inf;
20     return cur;
21 }
    
```

#### 4.14 Matrix exponentiation

```

1 typedef vector<vector<long long int>> Matrix;
2 long long int inf = 1000000007;
3 Matrix ones(int n) {
4     Matrix r(n,vector<long long int>(n));
5     for(int i= 0; i<n; i++){
6         r[i][i]=1;
7     }
8     return r;
9 }
10 Matrix operator*(Matrix &a, Matrix &b) {
11     int n=a.size(),m=b[0].size(),z=a[0].size();
12     Matrix r(n,vector<long long int>(m));
13     for(int i=0; i<n; i++){
14         for(int j=0; j<m; j++){
15             for(int k=0;k<z; k++){
16                 r[i][j]+=((a[i][k]%inf)*(b[k][j]%inf))%inf;
17                 r[i][j]=inf;}}}
18     return r;
19 }
20 Matrix be(Matrix b, long long int e) {
21     Matrix r=ones(b.size());
22     while(e){if(e&1LL)r=r*b;b=b*b;e/=2;}
23     return r;
24 }
25
26 //Matrix mat(n,vector<long long int>(n));
    
```

#### 4.15 Ecuaciones Diofantinas

```

1 long long int gcd(long long int a, long long int b, long long int& x,
2     long long int& y) {
3     x = 1, y = 0;
    
```

```

3     long long int x1 = 0, y1 = 1, a1 = a, b1 = b;
4     while (b1) {
5         int q = a1 / b1;
6         tie(x, x1) = make_tuple(x1, x - q * x1);
7         tie(y, y1) = make_tuple(y1, y - q * y1);
8         tie(a1, b1) = make_tuple(b1, a1 - q * b1);
9     }
10    return a1;
11 }
12 long long int d;
13 bool findAnySol(long long int a, long long int& x, long long int b, long
14     long int& y, long long int c) {
15     long long int g = gcd(abs(a), abs(b), x, y);
16     if (c % g != 0) return false;
17     x *= c;
18     y *= c;
19     x /= g;
20     y /= g;
21     d = c / g;
22     if (a < 0) x = -x;
23
24     if (b < 0) y = -y;
25     return true;
26 }
27 //-----SOLBEGIN-----
28 int main() {
29     ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
30     long long int m, a, k, n;
31     long long int f, h,res;
32     //estira en n, y despues cada m
33     //estira en k+a, y despues cada a
34     cin >> n >> m >> a >> k;
35     while (n != 0 && m != 0 && a != 0 && k != 0) {
36         m = -m;
37         if (!findAnySol(m, f, a, h, k + a - n)) {
38             cout << "Impossible" << endl;
39         }else {
40             res = f * m+n;
41             while (res > 0) res -= m * d;
42             while (res < 0) res += m * d;
43
44             cout << res << endl;
45         }
46     }
    
```

```

45     cin >> n >> m >> a >> k;
46 }
47
48 }
49 //-----EOSOLUTION-----

```

#### 4.16 Pollard-Rho, Stolen from GGDem

```

1 long long int gcd(long long int a, long long int b){return a?gcd(b%a,a):
  b;}
2 long long int mulmod(long long int a, long long int b, long long int m)
  {
3     long long int r=a*b-(long long int)((long double)a*b/m+.5)*m;
4     return (r<0?r+m:r);
5 }
6 long long int expmod(long long int b, long long int e, long long int m){
7     if(!e)return 1;
8     long long int q=expmod(b,e/2,m);q=mulmod(q,q,m);
9     return (e&1?mulmod(b,q,m):q);
10 }
11 bool is_prime_prob(ll n, int a){
12     if(n==a)return true;
13     long long int s=0,d=n-1;
14     while(d%2==0)s++,d/=2;
15     long long int x=expmod(a,d,n);
16     if((x==1)|| (x+1==n))return true;
17     for(int i = 0; i<s-1; i++){
18         x=mulmod(x,x,n);
19         if(x==1)return false;
20         if(x+1==n)return true;
21     }
22     return false;
23 }
24 bool rabin(long long int n){ // true iff n is prime
25     if(n==1)return false;
26     int A[]={2,3,5,7,11,13,17,19,23};
27     for(int a: A) if(!is_prime_prob(n,a))return false;
28     return true;
29 }
30 long long int rho(long long int n){
31     if(!(n&1))return 2;
32     long long int x=2,y=2,d=1;
33     long long int c=rand()%n+1;

```

```

34     while(d==1){
35         x=(mulmod(x,x,n)+c)%n;
36         y=(mulmod(y,y,n)+c)%n;
37         y=(mulmod(y,y,n)+c)%n;
38         if(x>y)d=gcd(x-y,n);
39         else d=gcd(y-x,n);
40     }
41     return d==n?rho(n):d;
42 }
43 void fact(long long int n, map<long long int,int>& f){ //O (lg n)^3
44     if(n==1)return;
45     if(rabin(n)){f[n]++;return;}
46     long long int q=rho(n);
47     fact(q,f);fact(n/q,f);
48 }

```

#### 4.17 FFT, Stolen from GGDem

```

1 // SPOJ VFMUL - AC
2 // http://www.spoj.com/problems/VFMUL/
3 #include <bits/stdc++.h>
4 #define fst first
5 #define snd second
6 #define fore(i,a,b) for(int i=a,ThxDem=b;i<ThxDem;++i)
7 #define pb push_back
8 #define ALL(s) s.begin(),s.end()
9 #define FIN ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0)
10 #define SZ(s) int(s.size())
11 using namespace std;
12 typedef long long ll;
13 typedef pair<int,int> ii;
14
15 // MAXN must be power of 2 !!
16 // MOD-1 needs to be a multiple of MAXN !!
17 // big mod and primitive root for NTT:
18 const int MOD=998244353,RT=3,MAXN=1<<20;
19 typedef vector<int> poly;
20 // FFT
21 struct CD {
22     double r,i;
23     CD(double r=0, double i=0):r(r),i(i){}
24     double real()const{return r;}
25     void operator/=(const int c){r/=c, i/=c;}

```

```

26 };
27 CD operator*(const CD& a, const CD& b){
28     return CD(a.r*b.r-a.i*b.i,a.r*b.i+a.i*b.r);}
29 CD operator+(const CD& a, const CD& b){return CD(a.r+b.r,a.i+b.i);}
30 CD operator-(const CD& a, const CD& b){return CD(a.r-b.r,a.i-b.i);}
31 const double pi=acos(-1.0);
32 // NTT
33 /*
34 struct CD {
35     int x;
36     CD(int x):x(x){}
37     CD(){}
38     int get()const{return x;}
39 };
40 CD operator*(const CD& a, const CD& b){return CD(mulmod(a.x,b.x));}
41 CD operator+(const CD& a, const CD& b){return CD(addmod(a.x,b.x));}
42 CD operator-(const CD& a, const CD& b){return CD(submod(a.x,b.x));}
43 vector<int> rts(MAXN+9,-1);
44 CD root(int n, bool inv){
45     int r=rts[n]<0?rts[n]=pm(RT,(MOD-1)/n):rts[n];
46     return CD(inv?pm(r,MOD-2):r);
47 }
48 */
49 CD cp1[MAXN+9],cp2[MAXN+9];
50 int R[MAXN+9];
51 void dft(CD* a, int n, bool inv){
52     fore(i,0,n)if(R[i]<i)swap(a[R[i]],a[i]);
53     for(int m=2;m<=n;m*=2){
54         double z=2*pi/m*(inv?-1:1); // FFT
55         CD wi=CD(cos(z),sin(z)); // FFT
56         // CD wi=root(m,inv); // NTT
57         for(int j=0;j<n;j+=m){
58             CD w(1);
59             for(int k=j,k2=j+m/2;k2<j+m;k++,k2++){
60                 CD u=a[k];CD v=a[k2]*w;a[k]=u+v;a[k2]=u-v;w=w*wi;
61             }
62         }
63     }
64     if(inv)fore(i,0,n)a[i]/=n; // FFT
65     //if(inv){ // NTT
66     //    CD z(pm(n,MOD-2)); // pm: modular exponentiation
67     //    fore(i,0,n)a[i]=a[i]*z;
68     //}

```

```

69 }
70 poly multiply(poly& p1, poly& p2){
71     int n=p1.size()+p2.size()+1;
72     int m=1,cnt=0;
73     while(m<=n)m*=m,cnt++;
74     fore(i,0,m){R[i]=0;fore(j,0,cnt)R[i]=(R[i]<<1)|((i>>j)&1);}
75     fore(i,0,m)cp1[i]=0,cp2[i]=0;
76     fore(i,0,p1.size())cp1[i]=p1[i];
77     fore(i,0,p2.size())cp2[i]=p2[i];
78     dft(cp1,m,false);dft(cp2,m,false);
79     fore(i,0,m)cp1[i]=cp1[i]*cp2[i];
80     dft(cp1,m,true);
81     poly res;
82     n-=2;
83     fore(i,0,n)res.pb((1ll)floor(cp1[i].real()+0.5)); // FFT
84     //fore(i,0,n)res.pb(cp1[i].x); // NTT
85     return res;
86 }
87
88 char s[MAXN],t[MAXN],r[MAXN];
89
90 int main(){
91     int tn;
92     scanf("%d",&tn);
93     while(tn--){
94         vector<int> a,b,c;
95         scanf("%s%s",s,t);
96         for(int i=0;s[i];++i)a.pb(s[i]-'0');reverse(a.begin(),a.end());
97         for(int i=0;t[i];++i)b.pb(t[i]-'0');reverse(b.begin(),b.end());
98         c=multiply(a,b);
99         while(!c.empty()&&!c.back())c.pop_back();
100         if(c.empty()){puts("0");continue;}
101         int n=0;
102         ll x=0;
103         fore(i,0,c.size()){
104             x+=c[i];
105             r[n++]=x%10;
106             x/=10;
107         }
108         while(x){
109             r[n++]=x%10;
110             x/=10;
111         }

```

```

112     reverse(r,r+n);
113     bool p=false;
114     fore(i,0,n){
115         putchar(r[i]+'0');
116     }
117     puts("");
118 }
119 return 0;
120 }
```

## 4.18 Euler Totient Function

Es multiplicativa

```

1 void phi_1_to_n(int n) {
2     vector<int> phi(n + 1);
3     phi[0] = 0;
4     phi[1] = 1;
5     for (int i = 2; i <= n; i++)
6         phi[i] = i - 1;
7
8     for (int i = 2; i <= n; i++)
9         for (int j = 2 * i; j <= n; j += i)
10             phi[j] -= phi[i];
11 }
12
13 void phi_1_to_n(int n) {
14     vector<int> phi(n + 1);
15     for (int i = 0; i <= n; i++)
16         phi[i] = i;
17
18     for (int i = 2; i <= n; i++) {
19         if (phi[i] == i) {
20             for (int j = i; j <= n; j += i)
21                 phi[j] -= phi[j] / i;
22         }
23     }
24 }
```

## 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         ;}
10    bool operator != (multhash b) const { return !(h1== b.h1 && h2== b.h2)
11        ;}
12 public:
13     string s;
14     multhash(){
15         h1 = 0; h2 = 0; s = "";
16         for(char l = 'a'; l<='z'; l++) alf[l] = l-'a'+1;
17     }
18     void innit(){
19         unsigned long long int inf,p,op;
20
21         inf = 999727999;
22         p = 325255434; op = 325255434;
23         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 //6666665555777777
38 //986143414027351997
39 //974383618913296759
40 //973006384792642181
41 //953947941937929919
42 //9090909090909091
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 //0(|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
19     for(int r = 1; r<te.size(); r++){
20         for(int c = 1; c<pa.size(); c++){
21             dp[r][c] = dp[r-1][c-1]+score(te[r],pa[c]);
22             dp[r][c] = max(dp[r][c-1]+score(te[r], '*'),dp[r][c]);
23             dp[r][c] = max(dp[r-1][c]+score('*',pa[c]),dp[r][c]);
24         }
25     }
26
27     return dp[te.size()-1][pa.size()-1];
28 }
    
```

### 6.5 KMP

```

1 string T,P;
2 int bt[MAXN];
3 //0(|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     }
17 }
    
```

```

16         if(j==P.size()){//match, do anything
17             res++;j = bt[j];
18         }
19     }
20     return res;
21 }
    
```

## 6.6 Suffix Array

```

1 //se asume que la longitud de la cadena sera menor a 10**6, modificar el
2   ub a discrecion
3 #define ub 1000000LL
4 //pot de ub times two
5 #define ccd 12
6 //metodos y structs auxiliares para el suffix array
7 struct sufd{int id;long long int t;};
8 int getndigit(long long int num, int d){
9     while(d--) num/=10LL;
10    return (int) (num%10LL);
11 }
12 void radixSort(vector<sufd>& arr){
13     int count[10]; int n = arr.size();
14     vector<sufd> aux(n);
15     for(int d = 0; d<ccd; d++){
16         for(int i = 0; i<10; i++) count[i] = 0;
17         for(int i = 0; i<n; i++) count[getndigit(arr[i].t,d)]++;
18         for(int i = 1; i<10; i++) count[i]+=count[i-1];
19         for(int i = n-1; i>=0; i--){
20             count[getndigit(arr[i].t,d)]--;
21             aux[count[getndigit(arr[i].t,d)]] = arr[i];
22         }
23         for(int i = 0; i<n; i++) arr[i] = aux[i];
24     }
25 }
26 //El suffix array mismo, agregar caracter menor al alfabeto al final de
27   T
28 string T,P;
29 int* sa,*lcest;
30 int stsize;
31 void makesa(){
32     int n = T.size();
33     sa = new int[n+1](); int* ra = new int[2*n+2]();
    
```

```

33     for(int i = 0; i<n; i++){sa[i] = i; ra[i] = T[i];}
34
35     sufd aux;vector<sufd> arr(n);
36     for(int k = 1; k<n;k*=2){
37         arr.clear();
38         for(int i = 0; i<n; i++){
39             aux.id = sa[i]; aux.t = ra[sa[i]];aux.t*=ub;aux.t += ra[sa[i]
40                 ]+k];
41             arr.push_back(aux);
42         }
43         //en caso de TLE calar con STL sort
44         radixSort(arr);
45         sa[0] = arr[0].id; ra[sa[0]] = 0;
46         for(int i = 1; i<n; i++){
47             sa[i] = arr[i].id;
48             ra[sa[i]] = ra[sa[i-1]]+1;
49             if(arr[i].t == arr[i-1].t) ra[sa[i]]--;
50         }
51         if(ra[sa[n-1]]==n-1) break;
52     }
53     delete[]ra;
54 }
55 void makelce(){
56     int n = T.size();
57     int* lce = new int[n+2]();
58     int* rank = new int[n+2]();
59     for(int i = 0; i<n; i++) rank[sa[i]] = i;
60
61     int curr = 0;
62     for(int i= 0; i<n; i++){
63         if(rank[i]==0) continue;
64         for(int j = max(curr-1,0); j+max(i,sa[rank[i]-1])<n; j++){
65             if(T[i+j] == T[sa[rank[i]-1]+j]) curr = j;
66             if(T[i+j]!=T[sa[rank[i]-1]+j]){curr = j-1; break;}
67         }
68         curr++;lce[i] = curr;
69     }
70
71     int p = 1; while(p<=n) p*=2; stsize = 2*p-1;
72     lcest = new int[stsize+2]();
73     for(int i= p-1; i-(p-1)<n; i++) lcest[i] = lce[sa[i-(p-1)]];
74     for(int i = p-2; i>=0; i--) lcest[i] = min(lcest[2*i+1],lcest[2*i +
75         2]);
    
```

```

74     delete[] lce; delete[] rank;
75 }
76 int recque(int l, int r, int sti, int stil, int stir){
77     if(stir<l || stil>r) return ub;
78     if(l<=stil && stir<=r) return lcest[sti];
79     int stim = stil+stir; stim/=2;
80     return min(recque(l,r,sti*2+1,stil,stim),recque(l,r,sti*2+2,stim+1,
81         stir));
82 }
83 int getlce(int l, int r){
84     if(l>r) return 0;
85     return recque(l,r,0,0,ssize/2);
86 }
87 int buscarRec(int l, int r,int lcp,int eas){
88     if(l>r) return -1;
89     int m = (l+r)/2;
90     //string curr = T.substr(sa[m],T.size()-sa[m]);
91     int lce = (eas>m?getlce(m+1,eas):getlce(eas+1,m));
92     if(lce>lcp){
93         if(eas<m) return buscarRec(m+1,r,lcp,eas);
94         if(eas>m) return buscarRec(l,m-1,lcp,eas);
95     }
96     if(lce<lcp){
97         if(eas>m) return buscarRec(m+1,r,lcp,eas);
98         if(eas<m) return buscarRec(l,m-1,lcp,eas);
99     }
100     for(int i = lcp,n = T.size(); sa[m]+i<n && i<P.size(); i++){if(P[i]
101         !=T[sa[m]+i]) break; lcp++;}
102     if(lcp == P.size()) return m;
103     if(l==r) return -1;
104     return (P[lcp]>T[sa[m]+lcp]?buscarRec(m+1,r,lcp,m):buscarRec(l,m-1,
105         lcp,m));
106 }
107 int buscar(){
108     int n = T.size();
109     if(P.size()>n) return -1;
110     return buscarRec(1,n-1,0,0);
111 }
112 //CODIGO DE 100 LINEAS, TE HE FALLADO MarcosK
113 //Uso: lee T, agregar signo dolar, llama makesa(); makelce(); lee P para
114     despues buscar()
115 //delete[] sa; delete[] lcest; cuando leas de nuevo T
    
```

```

113 //O(|T| log(|T|)) preprocesamiento, O(|P|+log**2(|T|)) cada busqueda
114 //Buscar devuelve un indice cualquiera de sa tal que el sufijo denotado
115     tenga P como prefijo
116 //Se puede hacer mas corto?
    
```

## 6.7 STL Suffix Array

```

1 //se asume que la longitud de la cadena sera menor a 10**6, modificar el
2     ub a discrecion
3 #define ub 1000000LL
4 //pot de ub times two
5 #define ccd 12
6 //metodos y structs auxiliares para el suffix array
7 struct sufd{int id;long long int t;
8     bool operator<(const sufd b) const{return t<b.t;}}
9 };
10 //El suffix array mismo, agregar caracter menor al alfabeto al final de
11     T
12 string T,P;
13 int* sa,*lcest;
14 int ssize;
15 void makesa(){
16     int n = T.size();
17     sa = new int[n+1](); int* ra = new int[2*n+2]();
18     for(int i = 0; i<n; i++){sa[i] = i; ra[i] = T[i];}
19
20     sufd aux;vector<sufd> arr(n);
21     for(int k = 1; k<n;k*=2){
22         arr.clear();
23         for(int i = 0; i<n; i++){
24             aux.id = sa[i]; aux.t = ra[sa[i]];aux.t*=ub;aux.t += ra[sa[i]
25                 ]+k];
26             arr.push_back(aux);
27         }
28         //en caso de TLE calar con STL sort
29         sort(arr.begin(),arr.end());
30         sa[0] = arr[0].id; ra[sa[0]] = 0;
31         for(int i = 1; i<n; i++){
32             sa[i] = arr[i].id;
33             ra[sa[i]] = ra[sa[i-1]]+1;
34             if(arr[i].t == arr[i-1].t) ra[sa[i]]--;
35         }
36         if(ra[sa[n-1]]==n-1) break;
37     }
    
```

```

34     }
35     delete[] ra;
36 }
37 void makelce(){
38     int n = T.size();
39     int* lce = new int[n+2]();
40     int* rank = new int[n+2]();
41     for(int i = 0; i<n; i++) rank[sa[i]] = i;
42
43     int curr = 0;
44     for(int i = 0; i<n; i++){
45         if(rank[i]==0) continue;
46         for(int j = max(curr-1,0); j+max(i,sa[rank[i]-1])<n; j++){
47             if(T[i+j] == T[sa[rank[i]-1]+j]) curr = j;
48             if(T[i+j] != T[sa[rank[i]-1]+j]){curr = j-1; break;}
49         }
50         curr++;lce[i] = curr;
51     }
52
53     int p = 1; while(p<=n) p*=2; stsize = 2*p-1;
54     lcest = new int[stsize+2]();
55     for(int i = p-1; i-(p-1)<n; i++) lcest[i] = lce[sa[i-(p-1)]];
56     for(int i = p-2; i>=0; i--) lcest[i] = min(lcest[2*i+1],lcest[2*i +
57         2]);
58     delete[] lce; delete[] rank;
59 }
60 int recque(int l, int r, int sti, int stil, int stir){
61     if(stir<l || stil>r) return ub;
62     if(l<=stil && stir<=r) return lcest[sti];
63     int stim = stil+stir; stim/=2;
64     return min(recque(l,r,sti*2+1,stil,stim),recque(l,r,sti*2+2,stim+1,
65         stir));
66 }
67 int getlce(int l, int r){
68     if(l>r) return 0;
69     return recque(l,r,0,0,stsize/2);
70 }
71 int buscarRec(int l, int r,int lcp,int eas){
72     if(l>r) return -1;
73     int m = (l+r)/2;
74     //string curr = T.substr(sa[m],T.size()-sa[m]);
75     int lce = (eas>m?getlce(m+1,eas):getlce(eas+1,m));
76     if(lce>lcp){

```

```

75         if(eas<m) return buscarRec(m+1,r,lcp,eas);
76         if(eas>m) return buscarRec(l,m-1,lcp,eas);
77     }
78     if(lce<lcp){
79         if(eas>m) return buscarRec(m+1,r,lcp,eas);
80         if(eas<m) return buscarRec(l,m-1,lcp,eas);
81     }
82
83     for(int i = lcp,n = T.size(); sa[m]+i<n && i<P.size(); i++){if(P[i
84         ]!=T[sa[m]+i]) break; lcp++;}
85     if(lcp == P.size()) return m;
86     if(l==r) return -1;
87     return (P[lcp]>T[sa[m]+lcp]?buscarRec(m+1,r,lcp,m):buscarRec(l,m-1,
88         lcp,m));
89 }
90 int buscar(){
91     int n = T.size();
92     if(P.size()>n) return -1;
93     return buscarRec(1,n-1,0,0);
94 }
95 pair<int,int> primeraYUltimaOc(){
96     int sai = buscar();
97     pair<int,int>res = {sai,sai};
98     if(sai==-1) return res;
99
100     int l,r,m;
101
102     r = sai-1; l = 0;
103     while(l<=r){
104         m = (l+r)/2;
105         if(getlce(m+1,sai)>=P.size()){
106             res.first = m; r = m-1;
107         }else{
108             l = m+1;
109         }
110     }
111     l = sai+1;r = T.size()-1;
112     while(l<=r){
113         m = (l+r)/2;
114         if(getlce(sai+1,m)>=P.size()){
115             res.second = m; l = m+1;
116         }else{
117             r = m-1;

```

```

116     }
117 }
118 return res;
119 }
120 //CODIGO DE 100 LINEAS, TE HE FALLADO MarcosK
121 //Uso: lee T, agregar signo dolar, llama makesa(); makelce(); lee P para
    despues buscar()
122 //delete[] sa; delete[] lcest; cuando leas de nuevo T
123 //O(|T| log(|T|)) preprocesamiento, O(|P|+log*2(|T|)) cada busqueda
124 //Buscar devuelve un indice cualquiera de sa tal que el sufijo denotado
    tenga P como prefijo
125 //Se puede hacer mas corto?

```

## 7 Clasicos

### 7.1 Job scheduling

#### 7.1.1 One machine, linear penalty

```

1 //cuando se tiene que encontrar un orden optimo
2 //para trabajos con una funcion lineal de penalty, basta con hacer un
    sort en O(n log n)
3 struct trabajo{
4     long long int penalty,tiempo;
5     int ind;
6 };
7 bool comp(const trabajo a, const trabajo b){
8     if (a.tiempo * b.penalty == a.penalty * b.tiempo) return a.ind<b.ind
        ;
9     return a.tiempo * b.penalty < a.penalty * b.tiempo;
10 }

```

#### 7.1.2 One machine, deadlines

```

1 //calcula la maxima cantidad de jobs que se pueden hacer dados sus
    deadlines y duraciones en O(n log n)
2 struct Job {
3     int deadline, duration, idx;
4
5     bool operator<(Job o) const {
6         return deadline < o.deadline;
7     }
8 };
9 vector<int> compute_schedule(vector<Job> jobs) {

```

```

10     sort(jobs.begin(), jobs.end());
11
12     set<pair<int,int>> s;
13     vector<int> schedule;
14     for (int i = jobs.size()-1; i >= 0; i--) {
15         int t = jobs[i].deadline - (i ? jobs[i-1].deadline : 0);
16         s.insert(make_pair(jobs[i].duration, jobs[i].idx));
17         while (t && !s.empty()) {
18             auto it = s.begin();
19             if (it->first <= t) {
20                 t -= it->first;
21                 schedule.push_back(it->second);
22             } else {
23                 s.insert(make_pair(it->first - t, it->second));
24                 t = 0;
25             }
26             s.erase(it);
27         }
28     }
29     return schedule;
30 }

```

#### 7.1.3 One machine, profit

```

1 // Dado n Jobs y su profit, calcula cual es el mayor profit que se puede
    obtener en O(n^2)
2 struct Job{int start, finish, profit;};
3 bool jobComparataor(Job s1, Job s2){return (s1.finish < s2.finish);}
4 // Find the latest job (in sorted array) that doesn't
5 // conflict with the job[i]. If there is no compatible job,
6 // then it returns -1.
7 vector <Job> arr;
8 int* memo;
9 int latestNonConflict( int i){
10     for (int j = i - 1; j >= 0; j--)
11         if (arr[j].finish <= arr[i - 1].start)
12             return j;
13     return -1;
14 }
15 // A recursive function that returns the maximum possible
16 // profit from given array of jobs. The array of jobs must
17 // be sorted according to finish time.
18 int findMaxProfitRec( int n){

```

```

19 // Base case
20 if (n == 1) return arr[n - 1].profit;
21     if (memo[n]>=0) return memo[n];
22 // Find profit when current job is included
23 int inclProf = arr[n - 1].profit;
24 int i = latestNonConflict(n);
25 if (i != -1) inclProf += findMaxProfitRec( i + 1);
26
27 // Find profit when current job is excluded
28 int exclProf = findMaxProfitRec( n - 1);
29
30 return memo[n]=max(inclProf, exclProf);
31 }
32
33 // The main function that returns the maximum possible
34 // profit from given array of jobs
35 int findMaxProfit( int n){
36     sort(arr.begin(),arr.end(), jobComparataor);
37     return findMaxProfitRec(n);
38 }

```

#### 7.1.4 Two machines, min time

```

1 //Obtiene el ordenamiento optimo de Jobs en dos maquinas en O(n log n)
2 struct Job {
3     int a, b, idx;
4     bool operator<(Job o) const {return min(a, b) < min(o.a, o.b);}
5 };
6 vector<Job> johnsons_rule(vector<Job> jobs) {
7     sort(jobs.begin(), jobs.end());
8     vector<Job> a, b;
9     for (Job j : jobs) {
10         if (j.a < j.b)
11             a.push_back(j);
12         else
13             b.push_back(j);
14     }
15     a.insert(a.end(), b.rbegin(), b.rend());
16     return a;
17 }
18
19 pair<int, int> finish_times(vector<Job> const& jobs) {
20     int t1 = 0, t2 = 0;

```

```

21     for (Job j : jobs) {
22         t1 += j.a;
23         t2 = max(t2, t1) + j.b;
24     }
25     return make_pair(t1, t2);
26 }

```

## 8 Flow

### 8.1 Dinic, thx GGDem

```

1 #define pb push_back
2 #define mp make_pair
3 #define fst first
4 #define snd second
5 #define ALL(s) s.begin(),s.end()
6 #define SZ(x) int((x).size())
7 #define fore(i,a,b) for(int i=a,to=b;i<to;++i)
8 using namespace std;
9 typedef long long ll;
10
11 #define INF (1LL<<62)
12 // Min cut: nodes with dist>=0 vs nodes with dist<0
13 // Matching MVC: left nodes with dist<0 + right nodes with dist>0
14 struct Dinic{
15     int nodes,src,dst;
16     vector<int> dist,q,work;
17     struct edge {int to,rev;ll f,cap;};
18     vector<vector<edge>> g;
19     Dinic(int x):nodes(x),g(x),dist(x),q(x),work(x){}
20     void add_edge(int s, int t, ll cap){
21         g[s].pb((edge){t,SZ(g[t]),0,cap});
22         g[t].pb((edge){s,SZ(g[s])-1,0,0});
23     }
24     bool dinic_bfs(){
25         fill(ALL(dist),-1);dist[src]=0;
26         int qt=0;q[qt++]=src;
27         for(int qh=0;qh<qt;qh++){
28             int u=q[qh];
29             fore(i,0,SZ(g[u])){
30                 edge &e=g[u][i];int v=g[u][i].to;
31                 if(dist[v]<0&&e.f<e.cap)dist[v]=dist[u]+1,q[qt++]=v;
32             }

```

```

33     }
34     return dist[dst]>=0;
35 }
36 ll dinic_dfs(int u, ll f){
37     if(u==dst)return f;
38     for(int &i=work[u];i<SZ(g[u]);i++){
39         edge &e=g[u][i];
40         if(e.cap<=e.f)continue;
41         int v=e.to;
42         if(dist[v]==dist[u]+1){
43             ll df=dinic_dfs(v,min(f,e.cap-e.f));
44             if(df>0){e.f+=df;g[v][e.rev].f-=df;return df;}
45         }
46     }
47     return 0;
48 }
49 ll max_flow(int _src, int _dst){
50     src=_src;dst=_dst;
51     ll result=0;
52     while(dinic_bfs()){
53         fill(ALL(work),0);
54         while(ll delta=dinic_dfs(src,INF))result+=delta;
55     }
56     return result;
57 }
58 };
59
60 //-----SOLBEGIN-----
61 int main() {
62     ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
63     //l set,r set
64     int n,m;
65     cin>>n>>m;
66     m+=n;
67     Dinic d(n+m+2);
68     for(int i = 1; i<=n; i++) d.add_edge(0,i,1);
69     for(int i = n+1; i<=m; i++) d.add_edge(i,m+1,1);
70
71     int fin,q;
72     for(int i = 1; i<=n; i++){
73         cin>>q;
74         while(q--){
75             cin>>fin;

```

```

76         d.add_edge(i,n+fin,1);
77     }
78 }
79 int res =d.max_flow(0,m+1);
80 m-=n;
81 //how many were left unmatched
82 cout<<m-res<<endl;
83 }
84 //-----EOSOLUTION-----

```

## 9 Miscellaneous

### 9.1 pbds

```

1 #include "bits/stdc++.h"
2 #include <bits/extc++.h>
3 using namespace __gnu_pbds;
4 using namespace std;
5 typedef tree<pair<int,int>, null_type,less<pair<int,int>>, rb_tree_tag,
6     tree_order_statistics_node_update> ost;
7 using namespace std;
8 int main(){
9     ost arbol;
10    int n = 5;
11    for(int id = 1; id<=n; id++)
12        for(int val = 0; val<n; val++)
13            arbol.insert({val,id});
14    //te da el valor mas pequenio, en caso de empate te da el del id mas
15    //pequenio
16    cout<<(*arbol.find_by_order(0)).first<<" " <<(*arbol.find_by_order(0)
17    ).second<<endl;
18    //te da el indice (base 0) de la primera ocurrencia de .first
19    cout<<arbol.order_of_key({1,-1})<<endl;;
20 }

```

### 9.2 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;
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-----

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
13 // power of 2)
14 a = 15; n = 8-1;
15 a &= n;
16 cout << "a%n, a_u=15, n_u=2^3" << endl;
17 cout << a << endl;
18 //Apaga el bit menos significativo de a
19 a = 14;
20 b = (a & ((-1) * a));
21 a &= ~b;
22 cout << a << endl;
23 //enciende el ultimo cero de a
24 a = 9;
25 b = ~a;
26 b = (b & ((-1) * b));
27 a = a | b;
28 cout << a<<endl;
29 //contar bits encendidos en a
30 cout << __builtin_popcount(a)<<endl;
31 //checar la paridad de a
32 cout << (__builtin_parity(a) ? "IMPAR" : "PAR") << endl;
33 //contar leading zeroes en a
34 cout << __builtin_clz(a)<<endl;
35 //contar 9,trailing zeroes en a
36 cout << __builtin_ctz(a)<<endl;
37 }
38 //-----EOSOLUTION-----
    
```



## 10 Testing

### 10.1 Gen and AutoRun testcases

#### 10.1.1 Gen.cpp

```

1 #include <iostream>
2 #include <string.h>
3 #include <random>
4 #include <chrono>
5 using namespace std;
6 //args nombreDelEjecutable,seed, len
7 int main (int argc, char **argv) {
8     // argv is an array of strings
9     // atoi is a C function for converting a string into an int
10    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
11    srand(atoi(argv[1])); // srand sets the random seed
12    int n = atoi(argv[2]);
13    int d = rng()%n; d++;
14    string test = "";
15    for (int i = 0; i < n; i++) {
16        test+= 'a'+(rng()%26);
17    }
18    cout<<test<<"\n"<<d<<endl;
19 }
```

#### 10.1.2 Stress testing

```

1 g++ -std=c++14 gen.cpp -o gen
2 g++ -std=c++14 lazy.cpp -v -o lazy
3 g++ -std=c++14 lazyn.cpp -v -o lazyn
4 for i in `seq 1 $1`; do
5     # prints the current test number
6     # I like to do this so I can see progress is being made
7     #chmod +x test.sh
8     echo $i
9     ./gen $i $((1 + i%14)) > input.txt #pasa al generador una longitud
10    entre 1 y 14, para hacer operaciones matematicas, usar $((a+b))
11    ./lazy < input.txt > output.txt
12    ./lazyn < input.txt > answer.txt
13
14    diff output.txt answer.txt || break
15 done
```

#### 10.1.3 Autorun

```

1 g++ -std=c++14 gen.cpp -o gen
2 g++ -std=c++14 lazy.cpp -v -o lazy
3 for i in `seq 1 $1`; do
4     # prints the current test number
5     # I like to do this so I can see progress is being made
6     #chmod +x test.sh
7     echo $i
8
9     ./gen $i $((1 + i%14)) > input.txt
10    ./lazy < i${i}.txt > o${i}.txt
11
12    diff a${i}.txt o${i}.txt || break
13 done
```

### 10.2 Highly Composite Numbers

Particularly useful when testing number theoretical solutions.

1	1	
2	2	2
3	4	2 <sup>2</sup>
4	6	2*3
5	12	2 <sup>2</sup> *3
6	24	2 <sup>3</sup> *3
7	36	2 <sup>2</sup> *3 <sup>2</sup>
8	48	2 <sup>4</sup> *3
9	60	2 <sup>2</sup> *3*5
10	120	2 <sup>3</sup> *3*5
11	180	2 <sup>2</sup> *3 <sup>2</sup> *5
12	240	2 <sup>4</sup> *3*5
13	360	2 <sup>3</sup> *3 <sup>2</sup> *5
14	720	2 <sup>4</sup> *3 <sup>2</sup> *5
15	840	2 <sup>3</sup> *3*5*7
16	1260	2 <sup>2</sup> *3 <sup>2</sup> *5*7
17	1680	2 <sup>4</sup> *3*5*7
18	2520	2 <sup>3</sup> *3 <sup>2</sup> *5*7
19	5040	2 <sup>4</sup> *3 <sup>2</sup> *5*7
20	7560	2 <sup>3</sup> *3 <sup>3</sup> *5*7
21	10080	2 <sup>5</sup> *3 <sup>2</sup> *5*7
22	15120	2 <sup>4</sup> *3 <sup>3</sup> *5*7
23	20160	2 <sup>6</sup> *3 <sup>2</sup> *5*7
24	25200	2 <sup>4</sup> *3 <sup>2</sup> *5 <sup>2</sup> *7

25	27720	96	$2^3 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11$	68	1396755360	1536	$2^5 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
26	45360	100	$2^4 \cdot 3^4 \cdot 5 \cdot 7$	69	2095133040	1600	$2^4 \cdot 3^4 \cdot 4 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
27	50400	108	$2^5 \cdot 3^2 \cdot 5^2 \cdot 7$	70	2205403200	1680	$2^6 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$
28	55440	120	$2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11$	71	2327925600	1728	$2^5 \cdot 3^2 \cdot 2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
29	83160	128	$2^3 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11$	72	2793510720	1792	$2^6 \cdot 3^3 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
30	110880	144	$2^5 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11$	73	3491888400	1920	$2^4 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
31	166320	160	$2^4 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11$	74	4655851200	2016	$2^6 \cdot 3^2 \cdot 2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
32	221760	168	$2^6 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11$	75	5587021440	2048	$2^7 \cdot 3^3 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
33	277200	180	$2^4 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11$	76	6983776800	2304	$2^5 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
34	332640	192	$2^5 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11$	77	10475665200	2400	$2^4 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
35	498960	200	$2^4 \cdot 3^4 \cdot 5 \cdot 7 \cdot 11$	78	13967553600	2688	$2^6 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
36	554400	216	$2^5 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11$	79	20951330400	2880	$2^5 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
37	665280	224	$2^6 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11$	80	27935107200	3072	$2^7 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
38	720720	240	$2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13$	81	41902660800	3360	$2^6 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
39	1081080	256	$2^3 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$	82	48886437600	3456	$2^5 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
40	1441440	288	$2^5 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13$	83	64250746560	3584	$2^6 \cdot 3^3 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
41	2162160	320	$2^4 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$	84	73329656400	3600	$2^4 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
42	2882880	336	$2^6 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13$	85	80313433200	3840	$2^4 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
43	3603600	360	$2^4 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13$	86	97772875200	4032	$2^6 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
44	4324320	384	$2^5 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$	87	128501493120	4096	$2^7 \cdot 3^3 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
45	6486480	400	$2^4 \cdot 3^4 \cdot 4 \cdot 5 \cdot 7 \cdot 11 \cdot 13$	88	146659312800	4320	$2^5 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
46	7207200	432	$2^5 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13$	89	160626866400	4608	$2^5 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
47	8648640	448	$2^6 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$	90	240940299600	4800	$2^4 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
48	10810800	480	$2^4 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13$	91	293318625600	5040	$2^6 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
49	14414400	504	$2^6 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13$	92	321253732800	5376	$2^6 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
50	17297280	512	$2^7 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$	93	481880599200	5760	$2^5 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
51	21621600	576	$2^5 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13$	94	642507465600	6144	$2^7 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
52	32432400	600	$2^4 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13$	95	963761198400	6720	$2^6 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
53	36756720	640	$2^4 \cdot 3^3 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	96	1124388064800	6912	$2^5 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
54	43243200	672	$2^6 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13$	97	1606268664000	7168	$2^6 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 3 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
55	61261200	720	$2^4 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	98	1686582097200	7200	$2^4 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
56	73513440	768	$2^5 \cdot 3^3 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	99	1927522396800	7680	$2^7 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
57	110270160	800	$2^4 \cdot 3^4 \cdot 4 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	100	2248776129600	8064	$2^6 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
58	122522400	864	$2^5 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	101	3212537328000	8192	$2^7 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 3 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
59	147026880	896	$2^6 \cdot 3^3 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	102	3373164194400	8640	$2^5 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
60	183783600	960	$2^4 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	103	4497552259200	9216	$2^7 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
61	245044800	1008	$2^6 \cdot 3^2 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	104	6746328388800	10080	$2^6 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
62	294053760	1024	$2^7 \cdot 3^3 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	105	8995104518400	10368	$2^8 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
63	367567200	1152	$2^5 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	106	9316358251200	10752	$2^6 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29$
64	551350800	1200	$2^4 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	107	13492656777600	11520	$2^7 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
65	698377680	1280	$2^4 \cdot 3^3 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$	108	18632716502400	12288	$2^7 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29$
66	735134400	1344	$2^6 \cdot 3^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	109	26985313555200	12960	$2^8 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
67	1102701600	1440	$2^5 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17$	110	27949074753600	13440	$2^6 \cdot 3^4 \cdot 4 \cdot 5^2 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29$

111	32607253879200	13824	2^5*3^3*5^2*7^2*11*13*17*19*23*29	154	673209363589963200	96768	2^6*3^5*5^2*7^2*11*13*17*19*23*29*31*37
112	46581791256000	14336	2^6*3^3*5^3*7*11*13*17*19*23*29	155	748010403988848000	98304	2^7*3^3*5^3*7^2*11*13*17*19*23*29*31*37
113	48910880818800	14400	2^4*3^4*5^2*7^2*11*13*17*19*23*29	156	897612484786617600	103680	2^8*3^4*5^2*7^2*11*13*17*19*23*29*31*37
114	55898149507200	15360	2^7*3^4*5^2*7*11*13*17*19*23*29				
115	65214507758400	16128	2^6*3^3*5^2*7^2*11*13*17*19*23*29				
116	93163582512000	16384	2^7*3^3*5^3*7*11*13*17*19*23*29				
117	97821761637600	17280	2^5*3^4*5^2*7^2*11*13*17*19*23*29				
118	130429015516800	18432	2^7*3^3*5^2*7^2*11*13*17*19*23*29				
119	195643523275200	20160	2^6*3^4*5^2*7^2*11*13*17*19*23*29				
120	260858031033600	20736	2^8*3^3*5^2*7^2*11*13*17*19*23*29				
121	288807105787200	21504	2^6*3^3*5^2*7*11*13*17*19*23*29*31				
122	391287046550400	23040	2^7*3^4*5^2*7^2*11*13*17*19*23*29				
123	577614211574400	24576	2^7*3^3*5^2*7*11*13*17*19*23*29*31				
124	782574093100800	25920	2^8*3^4*5^2*7^2*11*13*17*19*23*29				
125	866421317361600	26880	2^6*3^4*5^2*7*11*13*17*19*23*29*31				
126	1010824870255200	27648	2^5*3^3*5^2*7^2*11*13*17*19*23*29*31				
127	1444035528936000	28672	2^6*3^3*5^3*7*11*13*17*19*23*29*31				
128	1516237305382800	28800	2^4*3^4*5^2*7^2*11*13*17*19*23*29*31				
129	1732842634723200	30720	2^7*3^4*5^2*7*11*13*17*19*23*29*31				
130	2021649740510400	32256	2^6*3^3*5^2*7^2*11*13*17*19*23*29*31				
131	2888071057872000	32768	2^7*3^3*5^3*7*11*13*17*19*23*29*31				
132	3032474610765600	34560	2^5*3^4*5^2*7^2*11*13*17*19*23*29*31				
133	4043299481020800	36864	2^7*3^3*5^2*7^2*11*13*17*19*23*29*31				
134	6064949221531200	40320	2^6*3^4*5^2*7^2*11*13*17*19*23*29*31				
135	8086598962041600	41472	2^8*3^3*5^2*7^2*11*13*17*19*23*29*31				
136	10108248702552000	43008	2^6*3^3*5^3*7^2*11*13*17*19*23*29*31				
137	12129898443062400	46080	2^7*3^4*5^2*7^2*11*13*17*19*23*29*31				
138	18194847664593600	48384	2^6*3^5*5^2*7^2*11*13*17*19*23*29*31				
139	20216497405104000	49152	2^7*3^3*5^3*7^2*11*13*17*19*23*29*31				
140	24259796886124800	51840	2^8*3^4*5^2*7^2*11*13*17*19*23*29*31				
141	30324746107656000	53760	2^6*3^4*5^3*7^2*11*13*17*19*23*29*31				
142	36389695329187200	55296	2^7*3^5*5^2*7^2*11*13*17*19*23*29*31				
143	48519593772249600	57600	2^9*3^4*5^2*7^2*11*13*17*19*23*29*31				
144	60649492215312000	61440	2^7*3^4*5^3*7^2*11*13*17*19*23*29*31				
145	72779390658374400	62208	2^8*3^5*5^2*7^2*11*13*17*19*23*29*31				
146	74801040398884800	64512	2^6*3^3*5^2*7^2*11*13*17*19*23*29*31*37				
147	106858629141264000	65536	2^7*3^3*5^3*7*11*13*17*19*23*29*31*37				
148	112201560598327200	69120	2^5*3^4*5^2*7^2*11*13*17*19*23*29*31*37				
149	149602080797769600	73728	2^7*3^3*5^2*7^2*11*13*17*19*23*29*31*37				
150	224403121196654400	80640	2^6*3^4*5^2*7^2*11*13*17*19*23*29*31*37				
151	299204161595539200	82944	2^8*3^3*5^2*7^2*11*13*17*19*23*29*31*37				
152	374005201994424000	86016	2^6*3^3*5^3*7^2*11*13*17*19*23*29*31*37				
153	448806242393308800	92160	2^7*3^4*5^2*7^2*11*13*17*19*23*29*31*37				