.

First to Penalty

Contents

1 Template

```cpp
#include "bits/stdc++.h"
//assert(true/false) false leads to a RTE
using namespace std;
#define endl '\n'
#define DBG(x) cerr<<#x<< "=" << (x) << endl;
#define RAYA cerr<<"======================"<<endl;
#define RAYAS cerr<<"......................"<<endl;
//#define DBG(x) ;
//#define RAYA ;
//#define RAYAS ;
void solve(){
  //use the following for strings with whitespaces in them
  //getline(cin,str)
}
//--------------------SOLBEGIN----------------------------------
int main() {
  ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
  int tC; cin>>tC;

  while(tC--) solve();

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

2 Data structures

2.1 Simplified DSU (Stolen from GGDem)

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

2.2 Disjoint Set Union

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

2.3 Fenwick tree

```
//Fenwick tree, solo jala si la funcion cumple con ser:
//una binary associative function over a set with identity element and
    inverse elements
// incluyendo pero no limitandose a range sum
//define neutro y llena arr
#define MAXN 500010
long long int neutro;
vector<long long int> arr;
long long int fenwick[MAXN];

int get_low(int ind){return (ind&(ind+1));}
int get_upp(int ind){return (ind|(ind+1));}
long long int get_sum(int r){
    if(r<0) return neutro;
    return fenwick[r]+get_sum(get_low(r)-1);
}
long long int get_sum(int l, int r){return get_sum(r)-get_sum(l-1);}
void build(){
    int size = arr.size();
    for(int i = 0; i<size; i++ ) fenwick[i] = neutro;
    for(int i = 0; i<size; i++){
        fenwick[i]+=arr[i];
        if(get_upp(i)<size)fenwick[get_upp(i)]+=fenwick[i];
    }
}
void add(int ind, long long int d){
    for(;ind<arr.size(); ind = get_upp(ind)) fenwick[ind]+=d;
}
//no funciona con range queries
void range_add(int l, int r, int d){add (l,d); add(r+1,-d);}
```

2.4 Segment tree

```
//MAXN = 2^k, n = tam arreglo inicial
#define MAXN 262160
int stsize; long long int neut;int n;
long long int* st = new long long int[2*MAXN-1]();
long long int fst(long long int a, long long int b);
long long int build(int sti,int csize){
    if(csize == 1) return st[sti];
    return st[sti] = fst(build(sti*2+1,csize/2),build(sti*2+2,csize/2));
}
void innit(){
    for(int i = 0; i<stsize; i++) st[i] = neut;
    /*int d = 0;
    for(int i = stsize-n; i<stsize && d<n; i++){
        st[i] = arr[d];d++;
```

```
    }*/
    build(0,n);
}
void upd(int ind, long long int val){
    ind = stsize-n+ind;
    st[ind] = val;ind--;ind/=2;
    while(true){
        st[ind] = fst(st[ind*2+1],st[ind*2+2]);
        ind--;
        if(ind<0) break;
        ind/=2;
    }
}
long long int rqu(int l, int r,int sti, int ls, int rs){
    if(l<=ls && rs<= r) return st[sti];
    if(r<ls || l>rs) return neut;
    int m = (rs+ls)/2;
    return fst(rqu(l,r,sti*2+1,ls,m),rqu(l,r,sti*2+2,m+1,rs));
}
long long int query(int l, int r){
    return rqu(l,r,0,0,n-1);
}
//uso, inicializa neut, n = primera potencia de 2 >= n del problema,
    stsize = 2*n-1
//llena arr de neutros hasta que su tam sea el nuevo n
//DEFINE LA FUNCION fst
```

2.5 Segment tree Lazy

```
//MAXN = 2^k, n = tam arreglo inicial
#define MAXN 262160
vector<int> arr;
int stsize; long long int neut;int n;
long long int* st = new long long int[2*MAXN-1]();
long long int* pendientes = new long long int[2*MAXN-1]();
long long int fst(long long int a, long long int b){return a+b;}
long long int build(int sti,int csize){
    if(csize == 1) return st[sti];
    return st[sti] = fst(build(sti*2+1,csize/2),build(sti*2+2,csize/2));
}
bool hasChildren(int sti){sti*=2;sti++;sti++;return sti<stsize;}
void innit(){
    for(int i = 0; i<stsize; i++) st[i] = neut;
    int d = 0;
    for(int i = stsize-n; i<stsize && d<n; i++) {st[i] = arr[d];d++;}
    build(0,n);
}
void updrec(int l,int r, int sl, int sr,int sti, long long int val){
    if(sr<l || r< sl) return;
    if(l<= sl && sr <=r){
        st[sti] += val*(sr-sl+1);
        if(hasChildren(sti)){pendientes[sti*2+1]+=val;pendientes[sti
            *2+2]+=val;}
        return;
    }

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

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

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

    int m = (rs+ls)/2;
    return fst(rqu(l,r,sti*2+1,ls,m),rqu(l,r,sti*2+2,m+1,rs));
}
long long int query(int l, int r){
    return rqu(l,r,0,0,n-1);
}
//uso, inicializa neut, n = primera potencia de 2 >= n del problema,
    stsize = 2*n-1
//llena arr de neutros hasta que su tam sea el nuevo n
//DEFINE LA FUNCION fst
```

2.6 Trie

```
struct triver {
    char alphabet;
    bool ter;
    vector<triver*> child;
    triver(char a): alphabet(a) { child.assign(26, NULL); ter = false; }
};
class trie{
private:
    triver* root;
public:
    trie() { root = new triver('!');}
    void insert(string s){
        triver* curr = root;
        for(char l: s){
            if(curr->child[l-'A'] == NULL) curr->child[l-'A'] =  new
                triver(l);
            curr = curr->child[l-'A'];
        }
        curr->ter = true;
    }
    bool search(string s){
        triver* curr = root;
        for(char l: s){
            if(curr == NULL) break;
            curr = curr->child[l-'A'];
        }
        if(curr == NULL) return false;
        return curr->ter;
    }
};
```

3 Graphs

3.1 Graph Transversal

3.1.1 BFS

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

3.1.2 DFS

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

3.2 Topological Sort

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

```
}
void topologicalSort(){
  vis.reset();
  topsort.clear();
  for(int i = 0; i<n; i++){if(!vis[i]){dfs(i);}}
  reverse(topsort.begin(),topsort.end());
}
```

3.3 APSP: Floyd Warshall

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

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

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

3.4 SSSP

3.4.1 Lazy Dijkstra

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

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

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

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

3.4.2 Bellman-Ford

```
//esta es la implementacion huevona
#define GS 1000
//cuidado con overflows!!
#define INF 100000000
#define NINF -100000000
//destino, costo
vector<pair<int,int>> graph[GS];
int dist[GS];
struct edge{
```

```
    int from,to,cost;
};
//Corre en O(VE)
void bellmanFord(int origen,int tam){
    for(int i = 0; i<=tam; i++){
        dist[i] = INF;
    }
    dist[origen] = 0;
    edge aux;
    vector<edge> aristas;
    bool optimal;

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

    //Si se relajan todos las aristas V-1 veces en un orden arbitrario
    //Se asegura que la distancia optima para cada vertice sera
        alcanzada
    for(int i = 0; i<tam && !optimal; i++){
        optimal = true;
        for(edge elem: aristas){
            if(dist[elem.from] + elem.cost < dist[elem.to]){
                dist[elem.to] = dist[elem.from] + elem.cost;
                //si algun vertice fue actualizado significa que puede
                    que
                //las distancias aun no sean optimas
                optimal = false;
            }
        }
    }

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

}
```

3.5 Strongly Connected Components: Kosaraju

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

void invertirGrafo(int n){
    for(int p = 1;p<= n; p++)
        for(int h: graph[p])graphI[h].push_back(p);
}
void obtOrd(int p,int n){
    vis[p] = 1;
    for(int h: graph[p]){
        if(!vis[h] && h<=n) obtOrd(h,n);
    }
    orden.push_back(p);
}
int findSCC(int n){
    int res = 0;
    invertirGrafo(n);
    orden.clear();
    for(int i = 1; i<=n; i++) vis[i] =0;
    for(int i = 1; i<=n; i++) if(!vis[i]) obtOrd(i,n);
    reverse(orden.begin(),orden.end());
    //cuenta los connected components
    //vector<int> lscc;
    stack<int> fringe;
    int curr;
    for(int i = 1; i<=n; i++) vis[i] =0;
    for(int i: orden){
```

```
        //lscc.clear();
        if(!vis[i]){
            fringe.push(i);
            while (fringe.size()){
                curr = fringe.top();fringe.pop();
                //lscc.push_back(curr);
                if (!vis[curr]) {
                    vis[curr] = 1;
                    for (int h : graphI[curr]) fringe.push(h);
                }
            }
            res++;
        }
        //hacer lo que sea con lcss
    }
    return res;
}

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

3.6 Articulation Points and Bridges: ModTarjan

```
#define GS 50
vector<int> graph[GS];
bitset<GS> vis, isArtic;
vector<int> padre;
//id por tiempo, menor id accesible
//ya sea por descendientes o por back edges
vector<int> tId,lId;
//cantidad de hijos que tiene en el bfs spanning tree
int rootChildren;
int cnt;
int dfsRoot;
void findAP_B(int p){
    cnt++;vis[p] = 1;tId[p] = cnt;lId[p] = tId[p];

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

            findAP_B(hijo);

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

3.7 Kth-Ancestor using Binary Lifting

```
#define GS 100
//>log2(GS)
#define MAXANC 8
vector<int> graph[GS];
//NODO, 2**i ancestro
```

```
//inicializar todo en -1
int ancestro[GS][MAXANC];

//preprocesamiento, asume que graph es direccionado y rooteado
//agregar un bitset vis en caso de que falte
void buildAncestry(int curr,int h){
    int ub = 31-__builtin_clz(h|0);
    if(h==0) ub = 0;
    for(int i = 1; i<=ub; i++)
        ancestro[curr][i] = ancestro[ancestro[curr][i-1]][i-1];

    for(int hijo: graph[curr]){
        ancestro[hijo][0] = curr;
        buildAncestry(hijo,h+1);
    }
}

int kthAncestor(int curr, int k){
    if(k==0) return curr;
    int ub = 31-__builtin_clz(k);
    if(ancestro[curr][ub] == -1) return -1;
    return kthAncestor(ancestro[curr][ub],((1<<ub)^k));
}
```

3.8 LCA using Binary Lifting

```
//https://judge.yosupo.jp/problem/lca
#define GS 500000
//>log2(GS)
#define MAXANC 19
vector<int> graph[GS];
//NODO, 2**i ancestro
int ancestro[GS][MAXANC];
int dist[GS];
//preprocesamiento, asume que graph es direccionado y rooteado
//agregar un bitset vis en caso de que falte
void buildAncestry(int curr,int h){
    dist[curr] = h;
    int ub = 31-__builtin_clz(h|0);
    if(h==0) ub = 0;
    for(int i = 1; i<=ub; i++)
        ancestro[curr][i] = ancestro[ancestro[curr][i-1]][i-1];

    for(int hijo: graph[curr]){
        ancestro[hijo][0] = curr;
        buildAncestry(hijo,h+1);
    }
}

int kthAncestor(int curr, int k){
    if(k==0) return curr;
    int ub = 31-__builtin_clz(k);
    if(ancestro[curr][ub] == -1) return -1;
    return kthAncestor(ancestro[curr][ub],((1<<ub)^k));
}

int lca(int a,int b){
    int d = min(dist[a],dist[b]);
    a = kthAncestor(a,dist[a]-d);
    b = kthAncestor(b,dist[b]-d);
    //encuentra el primer true
    int l = 0,r = d,m;
    while(l<r){
        m = l+r; m/=2;
        if(kthAncestor(a,m) == kthAncestor(b,m)) r = m;
        else l = m+1;
    }
    return kthAncestor(a,l);
}
```

4 Math

4.1 Coeficientes binomiales. (Combinaciones n en k)

Una combinación es una forma de elegir k elementos de un grupo de tamaño n, sin importar el orden en que los eliges.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

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

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

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

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

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

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

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

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

$n=0$				1			
$n=1$			1	1			
$n=2$		1	2	1			
$n=3$		1	3	3	1		
$n=4$	1	4	6	4	1		
$n=5$	1	5	10	10	5	1	
$n=6$	1	6	15	20	15	6	1
	0	1	2	3	4	5	6

$n=0$	1						
$n=1$	1	1					
$n=2$	1	2	1				
$n=3$	1	3	3	1			
$n=4$	1	4	6	4	1		
$n=5$	1	5	10	10	5	1	
$n=6$	1	6	15	20	15	6	1
	0	1	2	3	4	5	6

4.2 Numeros catalanes

Los números de Catalan son una secuencia de números enteros que aparecen en varios problemas combinatorios y estructurales en matemáticas. Se utilizan para contar estructuras específicas que siguen reglas particulares, como el número de maneras de emparejar paréntesis correctamente, las maneras de dividir un polígono convexo en triángulos, o las maneras de construir árboles binarios completos.
Los números de Catalan también se aplican en la cuenta de caminos específicos, aunque no para cualquier tipo de camino. En particular, se usan para contar caminos restringidos en una cuadrícula.
Un ejemplo clásico es el conteo de caminos que van desde el punto $(0,0)$ hasta el punto (n,n) en una cuadrícula, avanzando solo hacia la derecha o hacia abajo, pero con la condición de que el camino nunca debe cruzar la diagonal principal (es decir, que en todo momento debe estar por encima o sobre la línea $y = x$. En este contexto, el número de caminos que cumplen esta restricción para una cuadrícula de tamaño n es el n-ésimo número de Catalan.

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

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

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

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

4.3 Separadores o barras y estrellas

En combinatoria, el método de separadores (o barras y estrellas) es una técnica para contar formas de dividir n objetos idénticos en k grupos.
Imagina que tienes n caramelos idénticos y quieres repartirlos entre k niños. Para resolver esto, colocas $k-1$ separadores entre los caramelos, dividiendo el total en k partes. Cada parte representa la cantidad de caramelos que cada niño recibe, y puede ser cero.
El numero de formas de hacer esto es $\binom{n+k-1}{k-1}$ ya que estamos eligiendo posiciones para los separadores entre los $n+k-1$ espacios disponibles.

$$\left(\binom{n}{k}\right) = \binom{n-1}{k-1} = \binom{n+k-1}{k-1}$$

4.4 Permutaciones

Una permutación es una forma de elegir y organizar k elementos de un grupo de tamaño n, donde el orden sí importa. (i.e. 1,2,3 es diferente a 3,1,2)

$$P(n,k) = \frac{n!}{(n-k)!}$$

Permutaciones objetos repetidos

Si existen elementos iguales entre los que hay que elegir (i.e. CARTA tiene dos 'A'), definamos la cantidad que ocurre el i-esimo elemento como am_i (as in amount)-

$$P(n,k) = \frac{P(n,k)}{e_1!e_2!...} = \frac{n!}{(n-k)!e_1!e_2!...}$$

4.5 Derangement

Un Derangement es una permutación que no deja ningún elemento en su lugar original.

$$Derangement(n) = \begin{cases} 0 & \text{si } n = 1. \\ 1 & \text{si } n = 2. \\ (n-1)(Derangement(n-1) + Derangement(n-2)) & \text{otherwise.} \end{cases}$$

$$Derangement(n) = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!}$$

4.6 Numeros de Fibonacci

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

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

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

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

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

4.7 Cantidad de divisores

Para encontrar la cantidad de divisores de un número, se usa su *factorización prima*. Supongamos que un número N se factoriza como:

$$N = p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k}$$

donde $(p_1, p_2, \ldots, p_k)$ son los factores primos distintos de N, y $(e_1, e_2, \ldots, e_k)$ son sus respectivos exponentes.
Para obtener la cantidad total de divisores de N, se toma cada exponente, se le suma uno, y luego se multiplican todos estos valores:

$$\text{Cantidad de divisores} = (e_1 + 1) \times (e_2 + 1) \times \cdots \times (e_k + 1)$$

Para fines de programacion competitiva, considera que N tiene aproximadamente $\sqrt[3]{N}$ divisores.

4.8 Respuesta modulo m

En programacion competitiva, es comun encontrar problemas cuya respuesta pueda exceder el limite de long long int de C++ (10^{18}), por lo que para mantener la respuesta en un rango aceptable se pide calcularla modulo m.
Para calcular una respuesta modulo m, es necesario construir el codigo tomando en cuenta las siguientes propiedades.

$$(a+b)\%m = (a\%m + b\%m)\%m$$

$$(a-b)\%m = (a\%m - b\%m + m)\%m$$

$$(a*b)\%m = (a\%m * b\%m)\%m$$

En el caso de divisiones es necesario calcular el inverso modular del divisor, expresado por b^{-1}, esto esta codificado en modinverse.

$$(a/b)\%m = (a\%m * (b^{-1})\%m)\%m$$

4.9 Binary Exponentiation and modArith

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

4.10 Modular Inverse (dividir mod)

```
long long int inf = 1000000007;
long long int gcd(long long int a, long long int b, long long int& x,
    long long int& y) {
    x = 1, y = 0;
    long long int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        long long int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
    return a1;
}
long long int modinverse(long long int b, long long int m){
```

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

4.11 Modular Binomial Coeficient and Permutations

```
long long int inf = 1000000007;
//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((fact[n-k]*fact[k])%inf,inf))%inf;
    }
};
```

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

4.12 Non-Mod Binomial Coeficient and Permutations

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

4.13 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];}
};
```

4.14 Ceil Fraccionario

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

4.15 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];
}
```

4.16 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;
    }
};
```

4.17 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]);
    }
};
```

4.18 Cycle Finding

```
void cyclef(long long int sem){
    long long int hare = f(sem),tort=f(sem);hare = f(hare);
    //liebre avanza dos pasos, tortuga solo uno
    while(hare!=tort){
        tort = f(tort); hare = f(f(hare));
    }
    //Se detiene en el inicio del ciclo
    tort = sem;
    while(hare!=tort){
        tort = f(tort); hare = f(hare);
    }

    int len = 1;
    tort = f(sem);
    while(hare!=tort){
        tort=f(tort);
        len++;
    }
}
```

4.19 Berlekamp Massey

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

```
            ll coef = delta / df1;
            for (ll &x : d) x *= coef;
            vector<ll> zeros(i - f - 1);
            zeros.insert(zeros.end(), d.begin(), d.end());
            d = zeros;
            vector<ll> temp = c;
            c.resize(max(c.size(), d.size()));
            for (int j=0; j<(int)d.size(); j++) c[j] += d[j];
            if (i - (int) temp.size() > f - (int) oldC.size()) {oldC =
                temp;f = i;}
        }
    }
    return c;
}
```

4.20 Modular Berlekamp Massey

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

4.21 Matrix exponentiation

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

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

4.22 Ecuaciones Diofantinas

```
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) {
    int q = a1 / b1;
    tie(x, x1) = make_tuple(x1, x - q * x1);
    tie(y, y1) = make_tuple(y1, y - q * y1);
    tie(a1, b1) = make_tuple(b1, a1 - q * b1);
  }
  return a1;
}
long long int d;
bool findAnySol(long long int a, long long int& x, long long int b, long
     long int& y, long long int c) {
```

```
  long long int g = gcd(abs(a), abs(b), x, y);
  if (c % g != 0) return false;
  x *= c;
  y *= c;
  x /= g;
  y /= g;
  d = c / g;
  if (a < 0) x = -x;

  if (b < 0) y = -y;
  return true;
}
//-------------------SOLBEGIN---------------------------------
int main() {
  ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
  long long int m, a, k, n;
  long long int f, h,res;
  //estira en n, y despues cada m
  //estira en k+a, y despues cada a
  cin >> n >> m >> a >> k;
  while (n != 0 && m != 0 && a != 0 && k != 0) {
    m = -m;
    if (!findAnySol(m, f, a, h, k + a - n)) {
      cout << "Impossible" << endl;
    }else {
      res = f * m+n;
      while (res > 0) res -= m * d;
      while (res < 0) res += m * d;

      cout << res << endl;
    }
    cin >> n >> m >> a >> k;
  }

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

4.23 Pollard-Rho, Stolen from GGDem

```
long long int gcd(long long int a, long long int b){return a?gcd(b%a,a):
    b;}
long long int mulmod(long long int a, long long int b, long long int m)
    {
  long long int  r=a*b-(long long int)((long double)a*b/m+.5)*m;
  return (r<0?r+m:r);
}
long long int expmod(long long int b, long long int e, long long int m){
  if(!e)return 1;
  long long int q=expmod(b,e/2,m);q=mulmod(q,q,m);
  return (e&1?mulmod(b,q,m):q);
}
bool is_prime_prob(ll n, int a){
  if(n==a)return true;
  long long int s=0,d=n-1;
  while(d%2==0)s++,d/=2;
  long long int x=expmod(a,d,n);
  if((x==1)||(x+1==n))return true;
  for(int i = 0; i<s-1; i++){
    x=mulmod(x,x,n);
    if(x==1)return false;
    if(x+1==n)return true;
  }
  return false;
}
bool rabin(long long int n){ // true iff n is prime
  if(n==1)return false;
  int A[]={2,3,5,7,11,13,17,19,23};
    for(int a: A) if(!is_prime_prob(n,a))return false;
  return true;
}
long long int rho(long long int n){
    if(!(n&1))return 2;
    long long int x=2,y=2,d=1;
    long long int c=rand()%n+1;
    while(d==1){
        x=(mulmod(x,x,n)+c)%n;
        y=(mulmod(y,y,n)+c)%n;
        y=(mulmod(y,y,n)+c)%n;
        if(x>=y)d=gcd(x-y,n);
        else d=gcd(y-x,n);
    }
    return d==n?rho(n):d;
}
void fact(long long int n, map<long long int,int>& f){ //O (lg n)^3
  if(n==1)return;
  if(rabin(n)){f[n]++;return;}
```

```
  long long int q=rho(n);
  fact(q,f);fact(n/q,f);
}
```

4.24 FFT, Stolen from GGDem

```
// SPOJ VFMUL - AC
// http://www.spoj.com/problems/VFMUL/
#include <bits/stdc++.h>
#define fst first
#define snd second
#define fore(i,a,b) for(int i=a,ThxDem=b;i<ThxDem;++i)
#define pb push_back
#define ALL(s) s.begin(),s.end()
#define FIN ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0)
#define SZ(s) int(s.size())
using namespace std;
typedef long long ll;
typedef pair<int,int> ii;

// MAXN must be power of 2 !!
// MOD-1 needs to be a multiple of MAXN !!
// big mod and primitive root for NTT:
const int MOD=998244353,RT=3,MAXN=1<<20;
typedef vector<int> poly;
// FFT
struct CD {
  double r,i;
  CD(double r=0, double i=0):r(r),i(i){}
  double real()const{return r;}
  void operator/=(const int c){r/=c, i/=c;}
};
CD operator*(const CD& a, const CD& b){
  return CD(a.r*b.r-a.i*b.i,a.r*b.i+a.i*b.r);}
CD operator+(const CD& a, const CD& b){return CD(a.r+b.r,a.i+b.i);}
CD operator-(const CD& a, const CD& b){return CD(a.r-b.r,a.i-b.i);}
const double pi=acos(-1.0);
// NTT
/*
struct CD {
  int x;
  CD(int x):x(x){}
  CD(){}
  int get()const{return x;}
};
CD operator*(const CD& a, const CD& b){return CD(mulmod(a.x,b.x));}
CD operator+(const CD& a, const CD& b){return CD(addmod(a.x,b.x));}
CD operator-(const CD& a, const CD& b){return CD(submod(a.x,b.x));}
vector<int> rts(MAXN+9,-1);
CD root(int n, bool inv){
  int r=rts[n]<0?rts[n]=pm(RT,(MOD-1)/n):rts[n];
  return CD(inv?pm(r,MOD-2):r);
}
*/
CD cp1[MAXN+9],cp2[MAXN+9];
int R[MAXN+9];
void dft(CD* a, int n, bool inv){
  fore(i,0,n)if(R[i]<i)swap(a[R[i]],a[i]);
  for(int m=2;m<=n;m*=2){
    double z=2*pi/m*(inv?-1:1); // FFT
    CD wi=CD(cos(z),sin(z)); // FFT
    // CD wi=root(m,inv); // NTT
    for(int j=0;j<n;j+=m){
      CD w(1);
      for(int k=j,k2=j+m/2;k2<j+m;k++,k2++){
        CD u=a[k];CD v=a[k2]*w;a[k]=u+v;a[k2]=u-v;w=w*wi;
      }
    }
  }
  if(inv)fore(i,0,n)a[i]/=n; // FFT
  //if(inv){ // NTT
  //  CD z(pm(n,MOD-2)); // pm: modular exponentiation
  //  fore(i,0,n)a[i]=a[i]*z;
  //}
}
poly multiply(poly& p1, poly& p2){
  int n=p1.size()+p2.size()+1;
  int m=1,cnt=0;
  while(m<=n)m+=m,cnt++;
  fore(i,0,m){R[i]=0;fore(j,0,cnt)R[i]=(R[i]<<1)|((i>>j)&1);}
  fore(i,0,m)cp1[i]=0,cp2[i]=0;
  fore(i,0,p1.size())cp1[i]=p1[i];
  fore(i,0,p2.size())cp2[i]=p2[i];
  dft(cp1,m,false);dft(cp2,m,false);
  fore(i,0,m)cp1[i]=cp1[i]*cp2[i];
  dft(cp1,m,true);
```

```
  poly res;
  n-=2;
  fore(i,0,n)res.pb((ll)floor(cp1[i].real()+0.5)); // FFT
  //fore(i,0,n)res.pb(cp1[i].x); // NTT
  return res;
}

char s[MAXN],t[MAXN],r[MAXN];

int main(){
  int tn;
  scanf("%d",&tn);
  while(tn--){
    vector<int> a,b,c;
    scanf("%s%s",s,t);
    for(int i=0;s[i];++i)a.pb(s[i]-'0');reverse(a.begin(),a.end());
    for(int i=0;t[i];++i)b.pb(t[i]-'0');reverse(b.begin(),b.end());
    c=multiply(a,b);
    while(!c.empty()&&!c.back())c.pop_back();
    if(c.empty()){puts("0");continue;}
    int n=0;
    ll x=0;
    fore(i,0,c.size()){
      x+=c[i];
      r[n++]=x%10;
      x/=10;
    }
    while(x){
      r[n++]=x%10;
      x/=10;
    }
    reverse(r,r+n);
    bool p=false;
    fore(i,0,n){
      putchar(r[i]+'0');
    }
    puts("");
  }
  return 0;
}
```

4.25 Euler Totient Function

Es multiplicativa

```
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    phi[0] = 0;
    phi[1] = 1;
    for (int i = 2; i <= n; i++)
        phi[i] = i - 1;

    for (int i = 2; i <= n; i++)
        for (int j = 2 * i; j <= n; j += i)
              phi[j] -= phi[i];
}

void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

5 Geometry

6 Strings

6.1 Explode by token

```
//#include <sstream>

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

```
}
```

6.2 Multiple Hashings DS

```
struct multhash{
    unsigned long long int h1,h2;
    unsigned long long int alf[257];
    bool operator < (multhash b) const {
    if (h1 != b.h1) return h1 < b.h1;
    return h2 < b.h2;
  }
  bool operator == (multhash b) const { return (h1== b.h1 && h2== b.h2)
      ;}
  bool operator != (multhash b) const { return !(h1== b.h1 && h2== b.h2)
      ;}
public:
    string s;
    multhash(){
        h1 = 0; h2 = 0;s = "";
        for(char l = 'a'; l<='z'; l++) alf[l] = l-'a'+1;
    }
    void innit(){
        unsigned long long int inf,p,op;

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

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

6.3 Permute chars of string

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

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

6.4 Longest common subsequence

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

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

    for(int r = 1; r<te.size(); r++){
```

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

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

6.5 KMP

```
string T,P;
int bt[MAXN];
//O(|Text|+|Pattern|)
void KMPpre(){
    int i = 0,j = -1; bt[0] = -1;
    while(i<P.size()){
        while(j>=0 && P[i]!=P[(j>=0?j:0)]) j = bt[j];
        i++;j++; bt[i] = j;
    }
}
int kmp(){
    int res =0, i = 0, j = 0;
    while(i<T.size()){
        while(j>=0 && T[i] != P[(j>=0?j:0)])  j = bt[j];
        i++; j++;
        if(j==P.size()){//match, do anything
            res++;j = bt[j];
        }
    }
    return res;
}
```

6.6 Suffix Array

```
//se asume que la longitud de la cadena sera menor a 10**6, modificar el
     ub a discrecion
#define ub 1000000LL
//pot de ub times two
#define ccd 12

//metodos y structs auxiliares para el suffix array
struct sufd{int id;long long int t;};
int getndigit(long long int num, int d){
    while(d--) num/=10LL;
    return (int) (num%10LL);
}
void radixSort(vector<sufd>& arr){
    int count[10]; int n = arr.size();
    vector<sufd> aux(n);
    for(int d = 0; d<ccd; d++){
        for(int i = 0; i<10; i++) count[i] = 0;
        for(int i = 0; i<n; i++) count[getndigit(arr[i].t,d)]++;
        for(int i = 1; i<10; i++) count[i]+=count[i-1];
        for(int i = n-1; i>=0; i--){
            count[getndigit(arr[i].t,d)]--;
            aux[count[getndigit(arr[i].t,d)]] = arr[i];
        }
        for(int i = 0; i<n; i++) arr[i] = aux[i];
    }
}
//El suffix array mismo, agregar caracter menor al alfabeto al final de
    T
string T,P;
int* sa,*lcest;
int stsize;
void makesa(){
    int n = T.size();
    sa = new int[n+1](); int* ra = new int[2*n+2]();
    for(int i = 0; i<n; i++){sa[i] = i; ra[i] = T[i];}

    sufd aux;vector<sufd> arr(n);
    for(int k = 1; k<n;k*=2){
        arr.clear();
        for(int i = 0; i<n; i++){
            aux.id = sa[i]; aux.t = ra[sa[i]];aux.t*=ub;aux.t += ra[sa[i
                ]+k];
            arr.push_back(aux);
        }
        //en caso de TLE calar con STL sort
        radixSort(arr);
        sa[0] = arr[0].id; ra[sa[0]] = 0;
        for(int i = 1; i<n; i++){
            sa[i] = arr[i].id;
            ra[sa[i]] = ra[sa[i-1]]+1;
            if(arr[i].t == arr[i-1].t) ra[sa[i]]--;
```

```
        }
        if(ra[sa[n-1]]==n-1) break;
    }
    delete[]ra;
}
void makelce(){
    int n = T.size();
    int* lce = new int[n+2]();
    int* rank = new int[n+2]();
    for(int i = 0; i<n; i++) rank[sa[i]] = i;

    int curr = 0;
    for(int i= 0; i<n; i++){
        if(rank[i]==0) continue;
        for(int j = max(curr-1,0); j+max(i,sa[rank[i]-1])<n; j++){
            if(T[i+j] == T[sa[rank[i]-1]+j]) curr = j;
            if(T[i+j]!=T[sa[rank[i]-1]+j]){curr = j-1; break;}
        }
        curr++;lce[i] = curr;
    }

    int p = 1; while(p<=n) p*=2; stsize = 2*p-1;
    lcest = new int[stsize+2]();
    for(int i= p-1; i-(p-1)<n; i++) lcest[i] = lce[sa[i-(p-1)]];
    for(int i = p-2; i>=0; i--) lcest[i] = min(lcest[2*i+1],lcest[2*i +
        2]);
    delete[] lce; delete[] rank;
}
int recque(int l, int r, int sti, int stil, int stir){
    if(stir<l || stil>r) return ub;
    if(l<=stil && stir<=r) return lcest[sti];
    int stim = stil+stir; stim/=2;
    return min(recque(l,r,sti*2+1,stil,stim),recque(l,r,sti*2+2,stim+1,
        stir));
}
int getlce(int l, int r){
    if(l>r) return 0;
    return recque(l,r,0,0,stsize/2);
}
int buscarRec(int l, int r,int lcp,int eas){
    if(l>r) return -1;
    int m = (l+r)/2;
    //string curr = T.substr(sa[m],T.size()-sa[m]);
    int lce = (eas>m?getlce(m+1,eas):getlce(eas+1,m));
    if(lce>lcp){
        if(eas<m) return buscarRec(m+1,r,lcp,eas);
        if(eas>m) return buscarRec(l,m-1,lcp,eas);
    }
    if(lce<lcp){
        if(eas>m) return buscarRec(m+1,r,lcp,eas);
        if(eas<m) return buscarRec(l,m-1,lcp,eas);
    }

    for(int i = lcp,n = T.size(); sa[m]+i<n && i<P.size(); i++){if(P[i
        ]!=T[sa[m]+i]) break; lcp++;}
    if(lcp == P.size()) return m;
    if(l==r) return -1;
    return (P[lcp]>T[sa[m]+lcp]?buscarRec(m+1,r,lcp,m):buscarRec(l,m-1,
        lcp,m));
}
int buscar(){
    int n = T.size();
    if(P.size()>n) return -1;
    return buscarRec(1,n-1,0,0);
}
//CODIGO DE 100 LINEAS, TE HE FALLADO MarcosK
//Uso: lee T, agregar signo dolar, llama makesa(); makelce(); lee P para
    despues buscar()
//delete[] sa; delete[] lcest; cuando leas de nuevo T
//O(|T| log(|T|)) preprocesamiento, O(|P|+log**2(|T|)) cada busqueda
//Buscar devuelve un indice cualquiera de sa tal que el sufijo denotado
    tenga P como prefijo
//Se puede hacer mas corto?
```

6.7 STL Suffix Array

```
//se asume que la longitud de la cadena sera menor a 10**6, modificar el
    ub a discrecion
#define ub 1000000LL
//pot de ub times two
#define ccd 12
//metodos y structs auxiliares para el suffix array
struct sufd{int id;long long int t;
    bool operator<(const sufd b) const{return t<b.t;}
};
//El suffix array mismo, agregar caracter menor al alfabeto al final de
```

```
    T
string T,P;
int* sa,*lcest;
int stsize;
void makesa(){
    int n = T.size();
    sa = new int[n+1](); int* ra = new int[2*n+2]();
    for(int i = 0; i<n; i++){sa[i] = i; ra[i] = T[i];}

    sufd aux;vector<sufd> arr(n);
    for(int k = 1; k<n;k*=2){
        arr.clear();
        for(int i = 0; i<n; i++){
            aux.id = sa[i]; aux.t = ra[sa[i]];aux.t*=ub;aux.t += ra[sa[i
                ]+k];
            arr.push_back(aux);
        }
        //en caso de TLE calar con STL sort
        sort(arr.begin(),arr.end());
        sa[0] = arr[0].id; ra[sa[0]] = 0;
        for(int i = 1; i<n; i++){
            sa[i] = arr[i].id;
            ra[sa[i]] = ra[sa[i-1]]+1;
            if(arr[i].t == arr[i-1].t) ra[sa[i]]--;
        }
        if(ra[sa[n-1]]==n-1) break;
    }
    delete[]ra;
}
void makelce(){
    int n = T.size();
    int* lce = new int[n+2]();
    int* rank = new int[n+2]();
    for(int i = 0; i<n; i++) rank[sa[i]] = i;

    int curr = 0;
    for(int i= 0; i<n; i++){
        if(rank[i]==0) continue;
        for(int j = max(curr-1,0); j+max(i,sa[rank[i]-1])<n; j++){
            if(T[i+j] == T[sa[rank[i]-1]+j]) curr = j;
            if(T[i+j]!=T[sa[rank[i]-1]+j]){curr = j-1; break;}
        }
        curr++;lce[i] = curr;
    }

    int p = 1; while(p<=n) p*=2; stsize = 2*p-1;
    lcest = new int[stsize+2]();
    for(int i= p-1; i-(p-1)<n; i++) lcest[i] = lce[sa[i-(p-1)]];
    for(int i = p-2; i>=0; i--) lcest[i] = min(lcest[2*i+1],lcest[2*i +
        2]);
    delete[] lce; delete[] rank;
}
int recque(int l, int r, int sti, int stil, int stir){
    if(stir<l || stil>r) return ub;
    if(l<=stil && stir<=r) return lcest[sti];
    int stim = stil+stir; stim/=2;
    return min(recque(l,r,sti*2+1,stil,stim),recque(l,r,sti*2+2,stim+1,
        stir));
}
int getlce(int l, int r){
    if(l>r) return 0;
    return recque(l,r,0,0,stsize/2);
}
int buscarRec(int l, int r,int lcp,int eas){
    if(l>r) return -1;
    int m = (l+r)/2;
    //string curr = T.substr(sa[m],T.size()-sa[m]);
    int lce = (eas>m?getlce(m+1,eas):getlce(eas+1,m));
    if(lce>lcp){
        if(eas<m) return buscarRec(m+1,r,lcp,eas);
        if(eas>m) return buscarRec(l,m-1,lcp,eas);
    }
    if(lce<lcp){
        if(eas>m) return buscarRec(m+1,r,lcp,eas);
        if(eas<m) return buscarRec(l,m-1,lcp,eas);
    }

    for(int i = lcp,n = T.size(); sa[m]+i<n && i<P.size(); i++){if(P[i
        ]!=T[sa[m]+i]) break; lcp++;}
    if(lcp == P.size()) return m;
    if(l==r) return -1;
    return (P[lcp]>T[sa[m]+lcp]?buscarRec(m+1,r,lcp,m):buscarRec(l,m-1,
        lcp,m));
}
int buscar(){
    int n = T.size();
```

```
    if(P.size()>n) return -1;
    return buscarRec(1,n-1,0,0);
}
pair<int,int> primeraYUltimaOc(){
    int sai = buscar();
    pair<int,int>res = {sai,sai};
    if(sai==-1) return res;

    int l,r,m;

    r = sai-1; l = 0;
    while(l<=r){
        m = (l+r)/2;
        if(getlce(m+1,sai)>=P.size()){
            res.first = m; r = m-1;
        }else{
            l = m+1;
        }
    }
    l = sai+1;r  = T.size()-1;
    while(l<=r){
        m = (l+r)/2;
        if(getlce(sai+1,m)>=P.size()){
            res.second = m; l = m+1;
        }else{
            r = m-1;
        }
    }
    return res;
}
//CODIGO DE 100 LINEAS, TE HE FALLADO MarcosK
//Uso: lee T, agregar signo dolar, llama makesa(); makelce(); lee P para despues buscar()
//delete[] sa; delete[] lcest; cuando leas de nuevo T
//O(|T| log(|T|)) preprocesamiento, O(|P|+log**2(|T|)) cada busqueda
//Buscar devuelve un indice cualquiera de sa tal que el sufijo denotado tenga P como prefijo
//Se puede hacer mas corto?
```

7 Clasicos

7.1 Job scheduling

7.1.1 One machine, linear penalty

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

7.1.2 One machine, deadlines

```
//calcula la maxima cantidad de jobs que se pueden hacer dados sus deadlines y duraciones en O(n log n)
struct Job {
    int deadline, duration, idx;

    bool operator<(Job o) const {
        return deadline < o.deadline;
    }
};
vector<int> compute_schedule(vector<Job> jobs) {
    sort(jobs.begin(), jobs.end());

    set<pair<int,int>> s;
    vector<int> schedule;
    for (int i = jobs.size()-1; i >= 0; i--) {
        int t = jobs[i].deadline - (i ? jobs[i-1].deadline : 0);
        s.insert(make_pair(jobs[i].duration, jobs[i].idx));
        while (t && !s.empty()) {
            auto it = s.begin();
            if (it->first <= t) {
                t -= it->first;
                schedule.push_back(it->second);
            } else {
```

```
                s.insert(make_pair(it->first - t, it->second));
                t = 0;
            }
            s.erase(it);
        }
    }
    return schedule;
}
```

7.1.3 One machine, profit

```
// Dado n Jobs y su profit, calcula cual es el mayor profit que se puede
    obtener en O(n^2)
struct Job{int start, finish, profit;};
bool jobComparataor(Job s1, Job s2){return (s1.finish < s2.finish);}
// Find the latest job (in sorted array) that doesn't
// conflict with the job[i]. If there is no compatible job,
// then it returns -1.
vector <Job> arr;
int* memo;
int latestNonConflict( int i){
  for (int j = i - 1; j >= 0; j--)
    if (arr[j].finish <= arr[i - 1].start)
      return j;
  return -1;
}
// A recursive function that returns the maximum possible
// profit from given array of jobs.  The array of jobs must
// be sorted according to finish time.
int findMaxProfitRec( int n){
  // Base case
  if (n == 1) return arr[n - 1].profit;
    if (memo[n]>=0) return memo[n];
  // Find profit when current job is included
  int inclProf = arr[n - 1].profit;
  int i = latestNonConflict(n);
  if (i != -1) inclProf += findMaxProfitRec( i + 1);

  // Find profit when current job is excluded
  int exclProf = findMaxProfitRec( n - 1);

  return memo[n]=max(inclProf, exclProf);
}

// The main function that returns the maximum possible
// profit from given array of jobs
int findMaxProfit( int n){
  sort(arr.begin(),arr.end(), jobComparataor);
  return findMaxProfitRec(n);
}
```

7.1.4 Two machines, min time

```
//Obtiene el ordenamiento optimo de Jobs en dos maquinas en O(n log n)
struct Job {
    int a, b, idx;
    bool operator<(Job o) const {return min(a, b) < min(o.a, o.b);}
};
vector<Job> johnsons_rule(vector<Job> jobs) {
    sort(jobs.begin(), jobs.end());
    vector<Job> a, b;
    for (Job j : jobs) {
        if (j.a < j.b)
            a.push_back(j);
        else
            b.push_back(j);
    }
    a.insert(a.end(), b.rbegin(), b.rend());
    return a;
}

pair<int, int> finish_times(vector<Job> const& jobs) {
    int t1 = 0, t2 = 0;
    for (Job j : jobs) {
        t1 += j.a;
        t2 = max(t2, t1) + j.b;
    }
    return make_pair(t1, t2);
}
```

8 Flow

8.1 Dinic, thx GGDem

```
#define pb push_back
#define mp make_pair
#define fst first
```

```cpp
#define snd second
#define ALL(s) s.begin(),s.end()
#define SZ(x) int((x).size())
#define fore(i,a,b) for(int i=a,to=b;i<to;++i)
using namespace std;
typedef long long ll;

#define INF (1LL<<62)
// Min cut: nodes with dist>=0 vs nodes with dist<0
// Matching MVC: left nodes with dist<0 + right nodes with dist>0
struct Dinic{
  int nodes,src,dst;
  vector<int> dist,q,work;
  struct edge {int to,rev;ll f,cap;};
  vector<vector<edge>> g;
  Dinic(int x):nodes(x),g(x),dist(x),q(x),work(x){}
  void add_edge(int s, int t, ll cap){
    g[s].pb((edge){t,SZ(g[t]),0,cap});
    g[t].pb((edge){s,SZ(g[s])-1,0,0});
  }
  bool dinic_bfs(){
    fill(ALL(dist),-1);dist[src]=0;
    int qt=0;q[qt++]=src;
    for(int qh=0;qh<qt;qh++){
      int u=q[qh];
      fore(i,0,SZ(g[u])){
        edge &e=g[u][i];int v=g[u][i].to;
        if(dist[v]<0&&e.f<e.cap)dist[v]=dist[u]+1,q[qt++]=v;
      }
    }
    return dist[dst]>=0;
  }
  ll dinic_dfs(int u, ll f){
    if(u==dst)return f;
    for(int &i=work[u];i<SZ(g[u]);i++){
      edge &e=g[u][i];
      if(e.cap<=e.f)continue;
      int v=e.to;
      if(dist[v]==dist[u]+1){
        ll df=dinic_dfs(v,min(f,e.cap-e.f));
        if(df>0){e.f+=df;g[v][e.rev].f-=df;return df;}
      }
    }
    return 0;
  }
  ll max_flow(int _src, int _dst){
    src=_src;dst=_dst;
    ll result=0;
    while(dinic_bfs()){
      fill(ALL(work),0);
      while(ll delta=dinic_dfs(src,INF))result+=delta;
    }
    return result;
  }
};

//--------------------SOLBEGIN----------------------------------
int main() {
  ios_base::sync_with_stdio(false); cout.tie(NULL); cin.tie(NULL);
    //l set,r set
    int n,m;
    cin>>n>>m;
    m+=n;
    Dinic d(n+m+2);
    for(int i = 1; i<=n; i++) d.add_edge(0,i,1);
    for(int i = n+1; i<=m; i++) d.add_edge(i,m+1,1);

    int fin,q;
    for(int i = 1; i<=n; i++){
        cin>>q;
        while(q--){
            cin>>fin;
            d.add_edge(i,n+fin,1);
        }
    }
    int res =d.max_flow(0,m+1);
    m-=n;
    //how many were left unmatched
    cout<<m-res<<endl;
}
//--------------------EOSOLUTION----------------------------------
```

9 Miscellaneous

9.1 pbds (Ordered set)

```cpp
#include "bits/stdc++.h"
#include <bits/extc++.h>
using namespace __gnu_pbds;
using namespace std;
typedef tree<pair<int,int>, null_type,less<pair<int,int>>, rb_tree_tag,
    tree_order_statistics_node_update> ost;
using namespace std;
int main(){
    ost arbol;
    int n = 5;
    for(int id = 1; id<=n; id++)
        for(int val = 0; val<n; val++)
            arbol.insert({val,id});
    //te da el valor mas pequenio, en caso de empate te da el del id mas
        pequenio
    cout<<(*arbol.find_by_order(0)).first<<"␣"<<(*arbol.find_by_order(0)
        ).second<<endl;
    //te da el indice (base 0) de la primera ocurrencia de .first
    cout<<arbol.order_of_key({1,-1})<<endl;;
}
```

9.2 Bit Manipulation

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


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

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

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

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

10 Testing

10.1 Gen and AutoRun testcases

10.1.1 Gen.cpp

```
#include <iostream>
#include <string.h>
#include <random>
#include <chrono>
using namespace std;
//args nombreDelEjecutable,seed, len
int main (int argc, char **argv) {
    // argv is an array of strings
    // atoi is a C function for converting a string into an int
    mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
    srand(atoi(argv[1])); // srand sets the random seed
    int n = atoi(argv[2]);
    int d = rng()%n; d++;
    string test = "";
    for (int i = 0; i < n; i++) {
        test+= 'a'+(rng()%26);
    }
    cout<<test<<" "<<d<<endl;
}
```

10.1.2 Stress testing

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

    diff output.txt answer.txt || break
done
```

10.1.3 Autorun

```
g++ -std=c++14 gen.cpp -o gen
g++ -std=c++14 lazy.cpp -v -o lazy
for i in `seq 1 $1`; do
    # prints the current test number
    # I like to do this so I can see progress is being made
    #chmod +x test.sh
    echo $i

    ./gen $i $((1 + i%14)) > input.txt
    ./lazy < i${i}.txt > o${i}.txt

```

```
    diff a${i}.txt o${i}.txt || break
done
```

10.2 Highly Composite Numbers

Particularly useful when testing number theoretical solutions.

1	1	
2	2	2
4	3	2^2
6	4	2*3
12	6	2^2*3
24	8	2^3*3
36	9	2^2*3^2
48	10	2^4*3
60	12	2^2*3*5
120	16	2^3*3*5
180	18	2^2*3^2*5
240	20	2^4*3*5
360	24	2^3*3^2*5
720	30	2^4*3^2*5
840	32	2^3*3*5*7
1260	36	2^2*3^2*5*7
1680	40	2^4*3*5*7
2520	48	2^3*3^2*5*7
5040	60	2^4*3^2*5*7
7560	64	2^3*3^3*5*7
10080	72	2^5*3^2*5*7
15120	80	2^4*3^3*5*7
20160	84	2^6*3^2*5*7
25200	90	2^4*3^2*5^2*7
27720	96	2^3*3^2*5*7*11
45360	100	2^4*3^4*5*7
50400	108	2^5*3^2*5^2*7
55440	120	2^4*3^2*5*7*11
83160	128	2^3*3^3*5*7*11
110880	144	2^5*3^2*5*7*11
166320	160	2^4*3^3*5*7*11
221760	168	2^6*3^2*5*7*11
277200	180	2^4*3^2*5^2*7*11
332640	192	2^5*3^3*5*7*11
498960	200	2^4*3^4*5*7*11
554400	216	2^5*3^2*5^2*7*11
665280	224	2^6*3^3*5*7*11
720720	240	2^4*3^2*5*7*11*13
1081080	256	2^3*3^3*5*7*11*13
1441440	288	2^5*3^2*5*7*11*13
2162160	320	2^4*3^3*5*7*11*13
2882880	336	2^6*3^2*5*7*11*13
3603600	360	2^4*3^2*5^2*7*11*13
4324320	384	2^5*3^3*5*7*11*13
6486480	400	2^4*3^4*5*7*11*13
7207200	432	2^5*3^2*5^2*7*11*13
8648640	448	2^6*3^3*5*7*11*13
10810800	480	2^4*3^3*5^2*7*11*13
14414400	504	2^6*3^2*5^2*7*11*13
17297280	512	2^7*3^3*5*7*11*13
21621600	576	2^5*3^3*5^2*7*11*13
32432400	600	2^4*3^4*5^2*7*11*13
36756720	640	2^4*3^3*5*7*11*13*17
43243200	672	2^6*3^3*5^2*7*11*13
61261200	720	2^4*3^2*5^2*7*11*13*17
73513440	768	2^5*3^3*5*7*11*13*17
110270160	800	2^4*3^4*5*7*11*13*17
122522400	864	2^5*3^2*5^2*7*11*13*17
147026880	896	2^6*3^3*5*7*11*13*17
183783600	960	2^4*3^3*5^2*7*11*13*17
245044800	1008	2^6*3^2*5^2*7*11*13*17
294053760	1024	2^7*3^3*5*7*11*13*17
367567200	1152	2^5*3^3*5^2*7*11*13*17
551350800	1200	2^4*3^4*5^2*7*11*13*17
698377680	1280	2^4*3^3*5*7*11*13*17*19
735134400	1344	2^6*3^3*5^2*7*11*13*17
1102701600	1440	2^5*3^4*5^2*7*11*13*17
1396755360	1536	2^5*3^3*5*7*11*13*17*19
2095133040	1600	2^4*3^4*5*7*11*13*17*19
2205403200	1680	2^6*3^4*5^2*7*11*13*17
2327925600	1728	2^5*3^2*5^2*7*11*13*17*19
2793510720	1792	2^6*3^3*5*7*11*13*17*19
3491888400	1920	2^4*3^3*5^2*7*11*13*17*19
4655851200	2016	2^6*3^2*5^2*7*11*13*17*19
5587021440	2048	2^7*3^3*5*7*11*13*17*19
6983776800	2304	2^5*3^3*5^2*7*11*13*17*19
10475665200	2400	2^4*3^4*5^2*7*11*13*17*19
13967553600	2688	2^6*3^3*5^2*7*11*13*17*19
20951330400	2880	2^5*3^4*5^2*7*11*13*17*19
27935107200	3072	2^7*3^3*5^2*7*11*13*17*19

41902660800	3360	2^6*3^4*5^2*7*11*13*17*19
48886437600	3456	2^5*3^3*5^2*7^2*11*13*17*19
64250746560	3584	2^6*3^3*5*7*11*13*17*19*23
73329656400	3600	2^4*3^4*5^2*7^2*11*13*17*19
80313433200	3840	2^4*3^3*5^2*7*11*13*17*19*23
97772875200	4032	2^6*3^3*5^2*7^2*11*13*17*19
128501493120	4096	2^7*3^3*5*7*11*13*17*19*23
146659312800	4320	2^5*3^4*5^2*7^2*11*13*17*19
160626866400	4608	2^5*3^3*5^2*7*11*13*17*19*23
240940299600	4800	2^4*3^4*5^2*7*11*13*17*19*23
293318625600	5040	2^6*3^4*5^2*7^2*11*13*17*19
321253732800	5376	2^6*3^3*5^2*7*11*13*17*19*23
481880599200	5760	2^5*3^4*5^2*7*11*13*17*19*23
642507465600	6144	2^7*3^3*5^2*7*11*13*17*19*23
963761198400	6720	2^6*3^4*5^2*7*11*13*17*19*23
1124388064800	6912	2^5*3^3*5^2*7^2*11*13*17*19*23
1606268664000	7168	2^6*3^3*5^3*7*11*13*17*19*23
1686582097200	7200	2^4*3^4*5^2*7^2*11*13*17*19*23
1927522396800	7680	2^7*3^4*5^2*7*11*13*17*19*23
2248776129600	8064	2^6*3^3*5^2*7^2*11*13*17*19*23
3212537328000	8192	2^7*3^3*5^3*7*11*13*17*19*23
3373164194400	8640	2^5*3^4*5^2*7^2*11*13*17*19*23
4497552259200	9216	2^7*3^3*5^2*7^2*11*13*17*19*23
6746328388800	10080	2^6*3^4*5^2*7^2*11*13*17*19*23
8995104518400	10368	2^8*3^3*5^2*7^2*11*13*17*19*23
9316358251200	10752	2^6*3^3*5^2*7*11*13*17*19*23*29
13492656777600	11520	2^7*3^4*5^2*7^2*11*13*17*19*23
18632716502400	12288	2^7*3^3*5^2*7*11*13*17*19*23*29
26985313555200	12960	2^8*3^4*5^2*7^2*11*13*17*19*23
27949074753600	13440	2^6*3^4*5^2*7*11*13*17*19*23*29
32607253879200	13824	2^5*3^3*5^2*7^2*11*13*17*19*23*29
46581791256000	14336	2^6*3^3*5^3*7*11*13*17*19*23*29
48910880818800	14400	2^4*3^4*5^2*7^2*11*13*17*19*23*29
55898149507200	15360	2^7*3^4*5^2*7*11*13*17*19*23*29
65214507758400	16128	2^6*3^3*5^2*7^2*11*13*17*19*23*29
93163582512000	16384	2^7*3^3*5^3*7*11*13*17*19*23*29
97821761637600	17280	2^5*3^4*5^2*7^2*11*13*17*19*23*29
130429015516800	18432	2^7*3^3*5^2*7^2*11*13*17*19*23*29
195643523275200	20160	2^6*3^4*5^2*7^2*11*13*17*19*23*29
260858031033600	20736	2^8*3^3*5^2*7^2*11*13*17*19*23*29
288807105787200	21504	2^6*3^3*5^2*7*11*13*17*19*23*29*31
391287046550400	23040	2^7*3^4*5^2*7^2*11*13*17*19*23*29
577614211574400	24576	2^7*3^3*5^2*7*11*13*17*19*23*29*31
782574093100800	25920	2^8*3^4*5^2*7^2*11*13*17*19*23*29
866421317361600	26880	2^6*3^4*5^2*7*11*13*17*19*23*29*31
1010824870255200	27648	2^5*3^3*5^2*7^2*11*13*17*19*23*29*31
1444035528936000	28672	2^6*3^3*5^3*7*11*13*17*19*23*29*31
1516237305382800	28800	2^4*3^4*5^2*7^2*11*13*17*19*23*29*31
1732842634723200	30720	2^7*3^4*5^2*7*11*13*17*19*23*29*31
2021649740510400	32256	2^6*3^3*5^2*7^2*11*13*17*19*23*29*31
2888071057872000	32768	2^7*3^3*5^3*7*11*13*17*19*23*29*31
3032474610765600	34560	2^5*3^4*5^2*7^2*11*13*17*19*23*29*31
4043299481020800	36864	2^7*3^3*5^2*7^2*11*13*17*19*23*29*31
6064949221531200	40320	2^6*3^4*5^2*7^2*11*13*17*19*23*29*31
8086598962041600	41472	2^8*3^3*5^2*7^2*11*13*17*19*23*29*31
10108248702552000	43008	2^6*3^3*5^3*7^2*11*13*17*19*23*29*31
12129898443062400	46080	2^7*3^4*5^2*7^2*11*13*17*19*23*29*31
18194847664593600	48384	2^6*3^5*5^2*7^2*11*13*17*19*23*29*31
20216497405104000	49152	2^7*3^3*5^3*7^2*11*13*17*19*23*29*31
24259796886124800	51840	2^8*3^4*5^2*7^2*11*13*17*19*23*29*31
30324746107656000	53760	2^6*3^4*5^3*7^2*11*13*17*19*23*29*31
36389695329187200	55296	2^7*3^5*5^2*7^2*11*13*17*19*23*29*31
48519593772249600	57600	2^9*3^4*5^2*7^2*11*13*17*19*23*29*31
60649492215312000	61440	2^7*3^4*5^3*7^2*11*13*17*19*23*29*31
72779390658374400	62208	2^8*3^5*5^2*7^2*11*13*17*19*23*29*31
74801040398884800	64512	2^6*3^3*5^2*7^2*11*13*17*19*23*29*31*37
106858629141264000	65536	2^7*3^3*5^3*7*11*13*17*19*23*29*31*37
112201560598327200	69120	2^5*3^4*5^2*7^2*11*13*17*19*23*29*31*37
149602080797769600	73728	2^7*3^3*5^2*7^2*11*13*17*19*23*29*31*37
224403121196654400	80640	2^6*3^4*5^2*7^2*11*13*17*19*23*29*31*37
299204161595539200	82944	2^8*3^3*5^2*7^2*11*13*17*19*23*29*31*37
374005201994424000	86016	2^6*3^3*5^3*7^2*11*13*17*19*23*29*31*37
448806242393308800	92160	2^7*3^4*5^2*7^2*11*13*17*19*23*29*31*37
673209363589963200	96768	2^6*3^5*5^2*7^2*11*13*17*19*23*29*31*37
748010403988848000	98304	2^7*3^3*5^3*7^2*11*13*17*19*23*29*31*37
897612484786617600	103680	2^8*3^4*5^2*7^2*11*13*17*19*23*29*31*37