Contents

1 Template

```
//#define NDEBUG

#include <bits/stdc++.h>
#include <bits/extc++.h>

#define StarBurstStream ios_base::
    sync_with_stdio(false); cin.tie(0); cout.
    tie(0);
#define iter(a) a.begin(), a.end()
#define riter(a) a.rbegin(), a.rend()
#define lsort(a) sort(iter(a))
#define gsort(a) sort(riter(a))
#define mp(a, b) make_pair(a, b)
#define pb(a) push_back(a)
#define eb(a) emplace_back(a)
#define pf(a) push_front(a)
#define pob pop_back()
#define pof pop_front()
#define F first
#define S second
#define printv(a, b) {bool pvaspace=false; \
for(auto pva : a){ \
  if(pvaspace) b << " "; pvaspace=true;\
  b << pva;\
}\
b << "\n";}
#define pii pair<int, int>
#define pll pair<ll, ll>
#define tiii tuple<int, int, int>
#define mt make_tuple
#define gt(t, i) get<i>(t)
#define iceil(a, b) ((a) / (b) + !!((a) % (b
    )))

//#define TEST

typedef long long ll;
typedef unsigned long long ull;

using namespace std;
using namespace __gnu_pbds;

const ll MOD = 1000000007;
const ll MAX = 2147483647;

template<typename A, typename B>
ostream& operator<<(ostream& o, pair<A, B> p
    ){
  return o << '(' << p.F << ',' << p.S << ')
    ';
}


int main(){
  StarBurstStream

  return 0;
}
```

2 Data Structure

2.1 Binary Indexed Tree

```
template<typename T>
struct BIT{

private:
  vector<T> bit;
  int lowbit(int x){
    return x & (-x);
  }

public:
  explicit BIT(int sz){
    bit.resize(sz + 1);
  }

  void modify(int x, T v){
    for(; x < bit.size(); x += lowbit(x))
   bit[x] += v;
  }

  T get(int x){
```

```
    T ans = T();
    for(; x; x -= lowbit(x)) ans += bit[x];
    return ans;
  }
};
```

2.2 Disjoint Set Union-Find

```
vector<int> dsu, rk;

void initDSU(int n){
  dsu.resize(n);
  rk.resize(n);
  for(int i = 0; i < n; i++) dsu[i] = i, rk[
   i] = 1;
}

int findDSU(int x){
  if(dsu[x] == x) return x;
  dsu[x] = findDSU(dsu[x]);
  return dsu[x];
}

void unionDSU(int a, int b){
  int pa = findDSU(a), pb = findDSU(b);
  if(rk[pa] > rk[pb]) swap(pa, pb);
  if(rk[pa] == rk[pb]) rk[pb]++;
  dsu[pa] = pb;
}
```

2.3 Segment Tree

```
template<typename T>
struct Node{
  T v = 0, tag = 0;
  int sz = 1, l = -1, r = -1;
  T rv(){
    return v + tag * sz;
  }
  void addTag(T t){
    tag += t;
  }
};

template<typename T>
T pullValue(T b, T c){
  return b + c;
}

template<typename T>
void pull(Node<T> &a, Node<T> &l, Node<T> &r
    ){
  a.v = pullValue(l.rv(), r.rv());
  a.sz = l.sz + r.sz;
}

template<typename T>
void push(Node<T> &a, Node<T> &l, Node<T> &r
    ){
  l.addTag(a.tag);
  r.addTag(a.tag);
  a.v = a.rv();
  a.tag = 0;
}

template<typename T>
struct SegmentTree{
  vector<Node<T>> st;
  int cnt = 0;

  explicit SegmentTree(int sz){
    st.resize(4 * sz);
  }

  int build(int l, int r, vector<T>& o){
    int id = cnt++;
    if(l == r){
      st[id].v = o[l];
      return id;
    }
    int m = (l + r) / 2;
    st[id].l = build(l, m, o);
    st[id].r = build(m + 1, r, o);
    pull(st[id], st[st[id].l], st[st[id].r])
   ;
    return id;
  }

  void modify(int l, int r, int v, int L,
   int R, int id){
    if(l == L && r == R){
      st[id].addTag(v);
      return;
    }
    int M = (L + R) / 2;
    if(r <= M) modify(l, r, v, L, M, st[id].
   l);
    else if(l > M) modify(l, r, v, M + 1, R,
    st[id].r);
    else{
      modify(l, M, v, L, M, st[id].l);
      modify(M + 1, r, v, M + 1, R, st[id].r
   );
    }
    pull(st[id], st[st[id].l], st[st[id].r])
   ;
  }

  T query(int l, int r, int L, int R, int id
   ){
    if(l == L && r == R) return st[id].rv();
    push(st[id], st[st[id].l], st[st[id].r])
   ;
    int M = (L + R) / 2;
    if(r <= M) return query(l, r, L, M, st[
   id].l);
    else if(l > M) return query(l, r, M + 1,
    R, st[id].r);
    else{
      return pullValue(query(l, M, L, M, st[
   id].l), query(M + 1, r, M + 1, R, st[id].
```

```
    r));
     }
  }

};
```

3 Graph

3.1 Dijkstra

```
//The first element in pair should be edge
    weight, and the second should be vertex
vector<vector<pii>> g;
int n;

int dijkstra(int start, int end){
  priority_queue<pii, vector<pii>, greater<
    pii>> q;
  for(pii p : g[start]){
    q.push(p);
  }
  q.push(mp(0, start));
  vector<int> dis(n, -1);
  dis[start] = 0;
  vector<int> visit(n);
  while(q.size()){
    int v = q.top().S;
    int d = q.top().F;
    if(v == end) break;
    q.pop();
    if(visit[v]) continue;
    visit[v] = true;
    for(pii p : g[v]){
      if(visit[p.S]) continue;
      if(dis[p.S] == -1 || d + p.F < dis[p.S
    ]){
        dis[p.S] = d + p.F;
        q.push(mp(dis[p.S], p.S));
      }
    }
  }
  return dis[end];
}
```

3.2 Floyd-Warshall

```
vector<vector<int>> g;
int n;

void floydwarshall(){
  for(int k = 0; k < n; k++)
    for(int i = 0; i < n; i++)
      for(int j = 0; j < n; j++)
        if(g[i][k] != -MAX && g[k][j] != -
    MAX && (g[i][j] == -MAX || g[i][k] + g[k
    ][j] < g[i][j]))
          g[i][j] = g[i][k] + g[k][j];
}
```

3.3 Kruskal

```
int kruskal(){
  int ans = 0;
  lsort(e);
  initDSU();
  for(auto& i : e){
    int a = i.S.F, b = i.S.S;
    if(findDSU(a) == findDSU(b)) continue;
    ans += i.F;
    unionDSU(a, b);
  }
  return ans;
}
```

3.4 Tarjan SCC

```
vector<vector<int>> g;
vector<int> st;
vector<bool> inst;
vector<int> scc;
vector<int> ts, low;
int tmp = 0;
int sccid = 0;

void initSCC(int n){
  tmp = 0;
  sccid = 0;
  st.clear();
  g.clear();
  g.resize(2 * n + 1);
  inst.clear();
  inst.resize(2 * n + 1);
  scc.clear();
  scc.resize(2 * n + 1);
  ts.clear();
  ts.resize(2 * n + 1, -1);
  low.clear();
  low.resize(2 * n + 1);
}

void dfs(int now){

  st.eb(now);
  inst[now] = true;
  ts[now] = ++tmp;
  low[now] = ts[now];

  for(int i : g[now]){
    if(ts[i] == -1){
      dfs(i);
      low[now] = min(low[now], low[i]);
    }
    else if(inst[i]) low[now] = min(low[now
    ], ts[i]);
  }

  if(low[now] == ts[now]){
    sccid++;
    int t;
    do{
      t = st.back();
      st.pob;
```

```
      inst[t] = false;
      scc[t] = sccid;
    }
    while(t != now);
  }

}
```

3.5 SPFA

```
const ll INFINITE = 2147483647;

int n;
vector<vector<pii>> g;

int spfa(int start, int end){

  vector<int> dis(n, INFINITE);
  int start;
  cin >> start;
  dis[start] = 0;

  queue<int> q;
  q.push(start);
  vector<bool> inq(n);
  inq[start] = true;
  vector<int> cnt(n);

  while(!q.empty()){
    int v = q.front();
    q.pop();
    inq[v] = false;
    for(pii p : g[v]){
      if(!(dis[p.F] == INFINITE || dis[v] +
    p.S < dis[p.F])) continue;
      cnt[p.F]++;
      if(cnt[p.F] >= n) return -INFINITE; //
    negetive cycle
      dis[p.F] = dis[v] + p.S;
      if(!inq[p.F]){
        inq[p.F] = true;
        q.push(p.F);
      }
    }

  }
  return dis[end];
}
```

4 String

4.1 KMP

```
vector<int> f;
void build(string& t){
  f.clear();
  f.resize(t.size());
  int p = -1;
  f[0] = -1;
  for(int i = 1; i < t.size(); i++){
    while(p != -1 && t[p + 1] != t[i]) p = f
    [p];
    if(t[p + 1] == t[i]) f[i] = p + 1;
    else f[i] = -1;
    p = f[i];
  }
}

int kmp(string& s, string& t){
  int ans = 0;
  int p = -1;
  for(int i = 0; i < s.size(); i++){
    while(p != -1 && t[p + 1] != s[i]) p = f
    [p];
    if(t[p + 1] == s[i]) p++;
    if(p + 1 == t.size()){
      ans++;
      p = f[p];
    }
  }
  return ans;
}
```

4.2 Z Value

```
vector<int> z;

void build(string s, int n){
  z.clear();
  z.resize(n);
  int l = 0;
  for(int i = 0; i < n; i++){
    if(l + z[l] >= i) z[i] = min(z[l] + l -
   i, z[i - l]);
    while(i + z[i] < n && s[z[i]] == s[i + z
    [i]]) z[i]++;
    if(i + z[i] > l + z[l]) l = i;
  }
}
```

5 Geometry

5.1 Vector Operations

```
template<typename T>
pair<T, T> operator+(pair<T, T> a, pair<T, T
    > b){
  return mp(a.F + b.F, a.S + b.S);
}

template<typename T>
pair<T, T> operator-(pair<T, T> a, pair<T, T
    > b){
  return mp(a.F - b.F, a.S - b.S);
}

template<typename T>
pair<T, T> operator*(pair<T, T> a, T b){
  return mp(a.F * b, a.S * b);
}
```

```
template<typename T>
pair<T, T> operator/(pair<T, T> a, T b){
  return mp(a.F / b, a.S / b);
}

template<typename T>
T dot(pair<T, T> a, pair<T, T> b){
  return a.F * b.F + a.S * b.S;
}

template<typename T>
T cross(pair<T, T> a, pair<T, T> b){
  return a.F * b.S - a.S * b.F;
}

template<typename T>
T abs2(pair<T, T> a){
  return a.F * a.F + a.S * a.S;
}
```

5.2 Convex Hull

```
template<typename T>
pair<T, T> operator-(pair<T, T> a, pair<T, T
    > b){
  return mp(a.F - b.F, a.S - b.S);
}

template<typename T>
T cross(pair<T, T> a, pair<T, T> b){
  return a.F * b.S - a.S * b.F;
}

template<typename T>
vector<pair<T, T>> getConvexHull(vector<pair
    <T, T>>& pnts){

  int n = pnts.size();
  lsort(pnts);

  vector<pair<T, T>> hull;
  hull.reserve(n);

  for(int i = 0; i < 2; i++){
    int t = hull.size();
    for(pair<T, T> pnt : pnts){
      while(hull.size() - t >= 2 && cross(
    hull.back() - hull[hull.size() - 2], pnt
    - hull[hull.size() - 2]) <= 0){
        hull.pop_back();
      }
      hull.pb(pnt);
    }
    hull.pop_back();
    reverse(iter(pnts));
  }

  return hull;
}
```

6 Number Theory

6.1 Prime Sieve

```
vector<int> prime;
vector<int> p;
void sieve(int n){
  prime.resize(n + 1, 1);
  for(int i = 2; i <= n; i++){
    if(prime[i] == 1){
      p.push_back(i);
      prime[i] = i;
    }
    for(int j : p){
      if((ll)i * j > n || j > prime[i])
   break;
      prime[i * j] = j;
    }
  }
}
```

7 DP Trick

7.1 Dynamic Convex Hull

```
const ll INF = 1LL << 60;

template<typename T>
struct Line{
  mutable T a, b, r = 0;

  Line(T a, T b) : a(a), b(b){}

  bool operator<(Line<T> l)const{
    return a < l.a;
  }

  bool operator<(T v)const{
    return r < v;
  }
};

template<typename T>
T divfloor(T a, T b){
  return a / b - ((a ^ b) < 0 && a % b);
}

template<typename T>
struct DynamicHull{
  multiset<Line<T>, less<>> s;

  int size(){
    return s.size();
  }

  bool intersect(typename set<Line<T>>::
   iterator a, typename set<Line<T>>::
   iterator &b){
    if(b == s.end()){
      a->r = INF;
```

```
      return false;
    }
    if(a->a == b->a){
      if(a->b > b->b) a->r = INF;
      else a->r = -INF;
    }
    else{
      a->r = divfloor(b->b - a->b, a->a - b->a);
    }
    return a->r >= b->r;
  }

  void insert(T a, T b){
    Line<T> l(a, b);
    auto it = s.insert(l), after = next(it), before = it;
    while(intersect(it, after)) after = s.erase(after);
    if(before != s.begin() && intersect(--before, it)){
      it = s.erase(it);
      intersect(before, it);
    }
    while((it = before) != s.begin() && (--before)->r >= it->r) intersect(before, it = s.erase(it));
  }

  T query(T v){
    Line<T> l = *s.lower_bound(v);
    return l.a * v + l.b;
  }
};
```

8 Numbers and Math

8.1 Fibonacci

$$f(n) = f(n-1) + f(n-2)$$

$$\begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

1	1	1	2	3	5
6	8	13	21	34	55
11	89	144	233	377	610
16	987	1597	2584	4181	6765
21	10946	17711	28657	46368	75025
26	121393	196418	317811	514229	832040
31	1346269	2178309	3524578	5702887	9227465

$f(45) \approx 10^9$
$f(88) \approx 10^{18}$

8.2 Catalan

$$C_0 = 1, C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$$

$$C_n = C_n^{2n} - C_{n-1}^{2n}$$

0	1	1	2	5	14
5	42	132	429	1430	4862
10	16796	58786	208012	742900	2674440
15	9694845	35357670	129644790	477638700	1767263190

8.3 Geometry

- Heron's formula:
  The area of a triangle whose lengths of sides is a,b,c and $s = (a+b+c)/2$ is $\sqrt{s(s-a)(s-b)(s-c)}$.
- Vector cross product:
  $v_1 \times v_2 = |v_1||v_2| \sin\theta = (x_1 \times y_2) - (x_2 \times y_1)$.
- Vector dot product:
  $v_1 \cdot v_2 = |v_1||v_2| \cos\theta = (x_1 \times y_1) + (x_2 \times y_2)$.

8.4 Prime Numbers

First 50 prime numbers:

1	2	3	5	7	11	13	17	19	23	29
11	31	37	41	43	47	53	59	61	67	71
21	73	79	83	89	97	101	103	107	109	113
31	127	131	137	139	149	151	157	163	167	173
41	179	181	191	193	197	199	211	223	227	229

Very large prime numbers:

1000001333 1000500889 2000000659 900004151 850001359

8.5 Number Theory

- Inversion:
  $aa^{-1} \equiv 1 \pmod{m}$. a^{-1} exists iff $\gcd(a, m) = 1$.
- Linear inversion:
  $a^{-1} \equiv (m - \lfloor \frac{m}{a} \rfloor) \times (m \bmod a)^{-1} \pmod{m}$
- Fermat's little theorem:
  $a^p \equiv a \pmod{p}$ if p is prime.
- Euler function:
  $\phi(n) = n \prod_{p|n} \frac{p-1}{p}$
- Euler theorem:
  $a^{\phi(n)} \equiv 1 \pmod{n}$ if $\gcd(a, n) = 1$.
- Extended Euclidean algorithm:
  $ax + by = \gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - \lfloor \frac{a}{b} \rfloor b) = bx_1 + (a - \lfloor \frac{a}{b} \rfloor b) y_1 = ay_1 + b(x_1 - \lfloor \frac{a}{b} \rfloor y_1)$
- Divisor function:
  $\sigma_x(n) = \sum_{d|n} d^x$. $n = \prod_{i=1}^{r} p_i^{a_i}$.
  $\sigma_x(n) = \prod_{i=1}^{r} \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$ if $x \neq 0$. $\sigma_0(n) = \prod_{i=1}^{r} (a_i + 1)$.