```
#pragma once
#include <bits/stdc++.h>
using namespace std;
#define NUL nullptr
#define rep(i, n) for (int i = 0; i < (n); ++i)        // [0,n-1]
#define forr(i, a, b) for (int i = (a); i < (b); ++i)  // [a,b-1]
#define rrep(i, n) for (int i = (n) - 1; i >= 0; --i)  // [n-1,0]
using ll = long long;
using pii = pair<int, int>;
using pll = pair<ll, ll>;
#define pb push_back
#define eb emplace_back
#define all(x) (x).begin(), (x).end()
#define rall(x) (x).rbegin(), (x).rend()
#define SZ(a) ((int)(a).size())
#define vi vector<int>
#define vvi vector<vector<int>>
#ifdef DEBUG
#define DOUT cout
#else
#define DOUT 0 && cout
#endif
int main(){
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
}

template<typename T> bool is_prime(T n) {
  if(n < 2) return 0;
  if(n % 2 == 0) return n == 2;
  if(n % 3 == 0) return n == 3;
  for(T i= 5; i * i <= n; i+= 6)
    if(n % i == 0 || n % (i + 2) == 0) return 0;
  return true; }
/*Compile-time sieve of Eratosthenes, O(n) space O(1) time*/
constexpr size_t N= 1e7;
bool prime[N];
template<size_t N> struct Prime {
  constexpr Prime() {
    prime[0]= prime[1]= false;
    for(size_t i= 2; i <= N; i++) prime[i]= true;
    for(size_t i= 2; i * i <= N; i++)
      if(prime[i])
        for(size_t j= i * i; j <= N; j+= i) prime[j]= false; }};
```

```
/* @brief Disjoint set union with merge by rank, path compression*/
struct DSU {
  vector<int> p;
  vector<int> r; // [i] = height of tree i
  DSU(int n): p(n), r(n) { rep(i, n) p[i]= i; }
  /* @brief Get root of x and compress path*/
  int find(int x) {
    if(p[x] != x) p[x]= find(p[x]);
    return p[x]; }
  /* @brief Unite root of x and root of y by r*/
  bool unite(int x, int y) {
    int root_x= find(x); int root_y= find(y);
    if(root_x == root_y) return 0;
    if(r[root_x] < r[root_y]) p[root_x]= root_y;
    else if(r[root_x] > r[root_y]) p[root_y]= root_x;
    else {
      p[root_y]= root_x;
      r[root_x]++; }
    return 1; }
  /* @brief Check x and y in same set*/
  bool same(int x, int y) { return find(x) == find(y); }};
/* @brief Segment tree(max) with lazy propagation
 * get(a,b) and add(a,b,v) b included, 0-index
 * M for modify, ex: sum st
 * t[n]=t[LST]+t[RST]; for M1 M2
 * return get(ql,qr,LRST)+get(ql,qr,RRST); for M3 */
#define LST n << 1
#define RST n << 1 | 1
#define LRST n << 1, l, (l + r) >> 1
#define RRST n << 1 | 1, ((l + r) >> 1) + 1, r // right range
#define IS_INT(la, ra, lb, rb) ((rb) >= (la) && (ra) >= (lb))
#define IS_INC(la, ra, lb, rb) ((la) >= (lb) && (rb) >= (ra))
struct St {
  int n;
  vi t, lz;
  St(int n): n(n), t(4 * n), lz(4 * n) {}
  St(vi& v): n(SZ(v) - 1), t(4 * n), lz(4 * n) { build(v, 1, 0, n); }
  void build(vi& v, int n, int l, int r) {
    if(l == r) {
      t[n]= v[l];
      return; }
    build(v, LRST);
    build(v, RRST);
    t[n]= max(t[LST], t[RST]); /*M*/ }
  void lazy(int n, int l, int r) {
    if(lz[n]) {
      t[n]+= lz[n];
      if(l != r) {
        lz[LST]+= lz[n];
        lz[RST]+= lz[n]; }
      lz[n]= 0; }}
  void add(int ql, int qr, int val) {
    if(qr < ql || n < qr) return;
    add(ql, qr, val, 1, 0, n); }
  void add(int ql, int qr, int val, int n, int l, int r) {
    lazy(n, l, r);
    if(!IS_INT(l, r, ql, qr)) return;
    if(IS_INC(l, r, ql, qr)) {
      t[n]+= val;
      if(l != r) {
        lz[LST]+= val;
        lz[RST]+= val; }
      return; }
    add(ql, qr, val, LRST);
    add(ql, qr, val, RRST);
    t[n]= max(t[LST], t[RST]); /*M*/ }
  int get(int ql, int qr) { return get(ql, qr, 1, 0, n); }
  int get(int ql, int qr, int n, int l, int r) {
    if(!IS_INT(l, r, ql, qr)) return -INT_MAX;
    lazy(n, l, r);
    if(IS_INC(l, r, ql, qr)) return t[n];
    return max(get(ql, qr, LRST), get(ql, qr, RRST)); /*M*/ }
  void print() {
    cout << get(0, 0);
    forr(i, 1, n + 1) cout << " " << get(i, i);
    cout << "\n"; }};
```

```
/* @brief Big num support negative add sub mul di */
struct Bn {
  string n;
  Bn(string s): n(s) {}
  Bn(ll x) { n= to_string(x); }
  bool neg() const { return n[0] == '-'; }
  Bn abs() const { Bn b= *this;
    if(b.neg()) b.n.erase(b.n.begin());
    return b; }
  Bn flip() { if(neg())n.erase(n.begin());
    else n.insert(n.begin(), '-');
    return *this; }
  Bn trim() { Bn b= *this;
    while(SZ(b.n) > 1 && b.n[0] == '0') b.n.erase(b.n.begin());
    return b; }
  bool operator==(const Bn& o) const { return n == o.n; }
  bool operator<(const Bn& o) const { if(neg() != o.neg()) return neg();
    Bn a= abs().trim(), b= o.abs().trim();
    if(SZ(a.n) != SZ(b.n)) return neg() ? SZ(a.n) > SZ(b.n) : SZ(a.n) < SZ(b.n);
    return neg() ? a.n > b.n : a.n < b.n; }
  bool operator>(const Bn& o) const { return o < *this; }
  bool operator>=(const Bn& o) const { return !(*this < o); }
  Bn add(Bn o) { if(neg() && o.neg()) return abs().add(o.abs()).flip();
    if(neg()) return o.sub(abs());
    if(o.neg()) return sub(o.abs());
    string a= n, b= o.n;
    while(SZ(a) < SZ(b)) a= "0" + a;
    while(SZ(b) < SZ(a)) b= "0" + b;
    int c= 0;
    string r= "";
    rrep(i, SZ(a)) {
      int s= (a[i] - 48) + (b[i] - 48) + c;
      c= s / 10;
      r= char(s % 10 + 48) + r; }
    if(c) r= "1" + r;
    return Bn(r).trim(); }
  Bn sub(Bn o) { if(o.neg()) return add(o.abs());
    if(*this < o) return o.sub(*this).flip();
    string a= n, b= o.n;
    while(SZ(b) < SZ(a)) b= "0" + b;
    int c= 0;
    string r= "";
    rrep(i, SZ(a)) {
      int s= (a[i] - 48) - (b[i] - 48) - c;
      c= 0;
      if(s < 0) {
        s+= 10;
        c= 1; }
      r= char(s + 48) + r; }
    return Bn(r).trim(); }
  Bn mul(Bn o) { if(neg() != o.neg()) return abs().mul(o.abs()).flip();
    Bn a= abs(), b= o.abs();
    int s= SZ(a.n), m= SZ(b.n);
    vi v(s + m);
    rrep(i, s) rrep(j, m) v[i + j + 1]+= (a.n[i] - 48) * (b.n[j] - 48);
    rrep(i, s + m) if(v[i] > 9) {
      v[i - 1]+= v[i] / 10;
      v[i]%= 10; }
    string r= "";
    rep(i, s + m) r+= char(v[i] + 48);
    return Bn(r).trim(); }
  Bn div(Bn o) { Bn a= abs(), b= o.abs(), q= 0, one= 1;
    bool s= neg() != o.neg();
    while(a >= b) {
      a= a.sub(b);
      q= q.add(one); }
    if(s) q.flip();
    return q; }};
```

4 - Graph, Prim

```
/* @brief Adjacency list weighted
 * For each v, save all edge that v has.
 * O(1) add_v add_e add_ue O(|V|+|E|) rm_v query O(|E|) rm_e
 */
template<typename W, typename V> struct Adj_list_w {
  unordered_map<V, vector<pair<W, V>>> g;
  Adj_list_w() {}
  Adj_list_w(int n) {
    rep(i, n) {
      vector<pair<W, V>> t;
      g[i]= t; }}
  void add_e(W w, V u, V v) {
    g[u].eb(w, v);
    g[v].eb(w, u); }
  void add_ue(W w, V u, V v) { g[u].emplace_back(w, v); }
  Adj_list_w<W, V> get_adj_list() const { return g; }
  void print() {
    for(auto& u: g) {
      cout << u.first << ":";
      for(auto& v: u.second) { cout << v.second << "," << v.first << " "; }
      cout << "\n"; }}
  /* @brief The min cost of minimum spanning tree of `g` start from `s_v`
   */
  W mst_prim_w(V s_v) {
    priority_queue<pair<W, V>, vector<pair<W, V>>, greater<pair<W, V>>> min_heap;
    vector<bool> visited(SZ(g), false);
    W min_cost= 0;
    min_heap.push({ 0, s_v });
    while(!min_heap.empty()) {
      auto [w, s]= min_heap.top();
      min_heap.pop();
      if(visited[s]) continue;
      min_cost+= w;
      visited[s]= true;
      for(auto& [w, e]: g.at(s)) {
        if(!visited[e]) { min_heap.push({ w, e }); }}}
    return min_cost; }
  /* @brief Minimum spanning tree of `g`
   */
  unordered_map<V, vector<pair<W, V>>> mst_prim(V s_v) {
    unordered_map<V, vector<pair<W, V>>> mst;
    priority_queue<tuple<W, V, V>, vector<tuple<W, V, V>>, greater<tuple<W, V, V>>> min_heap;
    vector<bool> visited(SZ(g), false);
    bool first= true;
    min_heap.push({ -1, s_v, -1 });
    while(!min_heap.empty()) {
      auto [w, s, e]= min_heap.top();
      min_heap.pop();
      if(visited[s]) continue;
      // 將要選擇此點，並加入他的鄰點
      if(first == false) {
        mst[e].push_back({ w, s });
        mst[s].push_back({ w, e }); }
      first= false;
      visited[s]= 1;
      for(auto& [nw, ne]: g.at(s)) {
        if(visited[ne]) continue;
        min_heap.push(make_tuple(nw, ne, s)); }}
    return mst; }
};
```