Team notebook

chungdinh, distiled, 5onyy

November 15, 2024

Contents

1 data-structures

1.1 fenwick

```
template <class T>
class fenwick {
  vector<T> a;
  int n;

public:
  fenwick() {}

  fenwick(int n_) : n(n_) {
    // check initial value
    a.assign(n + 5, static_cast<T>(0));
  }
  T query(int r) {
    T q = 0; // check default value of query
    for (; r > 0; r -= (r & -r)) {
      q += a[r]; // check query modification
    }
    return q;
  }

  void update(int r, T del) {
    for (; r <= n; r += (r & -r)) {
      a[r] += del; //check update rule
    }
  }
};
```

1.2 fenwick2d

```
// from online git I don't remember, all
    credits go to them
template<
    typename T // need to support +, -
> struct Fenwick2D {
    // 1-indexed
    int n1, n2;
    vector<vector<T>> f;

    Fenwick2D(int _n1, int _n2) : n1(_n1 + 1),
         n2(_n2 + 1), f(1+n1, vector<T> (1+n2,
         T(0))) {}
    Fenwick2D() {}
    // a[x][y] += val
    void update(ll x, ll y, T val) {
        assert(0 <= x && x < n1);
        assert(0 <= y && y < n2);
        ++x; ++y;
        for (int u = x; u <= n1; u += u & -u) {
            for (int v = y; v <= n2; v += v &
                -v) {
                f[u][v] += val;
            }
        }
    }

    T get(ll x1, ll y1, ll x2, ll y2) const {
        // [x1,y1] -> [x2,y2]
        return _get(x1, y1, x2 + 1, y2 + 1);
    }
    T get(ll x, ll y) const { // [0,0] -> [x,y]
        return _get(x + 1, y + 1);
    }

private:
    // return rect sum of [0, 0] -> [x-1, y-1]
    T _get(ll x, ll y) const {
        assert(0 <= x && x <= n1);
        assert(0 <= y && y <= n2);
        T res(0);
        for (int u = x; u > 0; u -= u & -u) {
            for (int v = y; v > 0; v -= v & -v)
                {
                res += f[u][v];
            }
        }
        return res;
    }

    // returns rect sum of [x1, y1] -> [x2-1,
        y2-1]
    T _get(ll x1, ll y1, ll x2, ll y2) {
        if (x1 == x2 || y1 == y2) return T(0);
        return get(x2, y2) - get(x1, y2) -
            get(x2, y1) + get(x1, y1);
    }
};
```

1.3 lazysegtree

```
// lazy propagation with one-based indexing
//
// build:
//   lazySegtree<int> seg(n); // vector<int>
    a(n + 1)
//   seg.build(a); // vector ‘a‘ must be
    one-indexed
//
// update:
//   seg.update(l, r, inc); //increase a[l ->
    r] += inc
//   //you may want to set this
// query:
//   seg.query(l, r)

template <class T>
struct lazySegTree {
  vector<T> st, lazy;
  int n;
  lazySegTree(int _n) : n(_n) {
    T default_value = 0;
    st.assign(4 * n + 5, default_value);
    lazy.assign(4 * n + 5, default_value);
  }
  void build(const vector<T> &a) {
    build(a, 1, 1, n);
  }
  void build(const vector<T> &a, int id, int
      l, int r) {
    lazy[id] = 0;
    if(l == r) {
      if(l < a.size()) {
        st[id] = a[l];
      }
      return;
    }
    int mid = (l + r) >> 1;
    build(a, id * 2, l, mid);
    build(a, id * 2 | 1, mid + 1, r);
    st[id] = max(st[id * 2], st[id * 2 | 1]);
  }
  void down(int id) {
    if(lazy[id] == 0)
      return;
    st[id * 2] += lazy[id];
    st[id * 2 | 1] += lazy[id];
    lazy[id * 2] += lazy[id];
    lazy[id * 2 | 1] += lazy[id];
    lazy[id] = 0;
  }
  void update(int u, int v, T val) {
    update(u, v, val, 1, 1, n);
  }
  void update(int u, int v, T val, int id, int
      l, int r) {
    if(v < l || r < u) return;
    if(u <= l && r <= v) {
      lazy[id] += val;
      st[id] += val;
      return;
    }
    down(id);
    int mid = (l + r) >> 1;
    update(u, v, val, id * 2, l, mid);
    update(u, v, val, id * 2 | 1, mid + 1, r);
    st[id] = max(st[id * 2], st[id * 2 | 1]);
  }
  T query(int u, int v) {
    return query(u, v, 1, 1, n);
  }
  T query(int u, int v, int id, int l, int r) {
    if(v < l || r < u) {
      return -INF;
    }
    if(u <= l && r <= v) {
      return st[id];
    }
    int mid = (l + r) >> 1;
    down(id);
    return max(query(u, v, id * 2, l, mid),
        query(u, v, id * 2 | 1, mid + 1, r));
  }
};
```

1.4 make-pbds

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

/**
 * Description: Policy-based data structures
     that supports normal set operation and
 *   * find_by_order(k) returns iterator to
     kth element starting from 0;
 *   * order_of_key(k) returns count of
     elements strictly [smaller] than k;
 * Caution:
 *   * do not define int long long before
     below declaration
 */

template <class T, class cmp = std::less<T>>
using ordered_set = tree<T, null_type, cmp,
    rb_tree_tag,tree_order_statistics_node_update>;

template <class key, class value, class cmp =
    std::less<key>>
using ordered_map = tree<key, value, cmp,
    rb_tree_tag,
    tree_order_statistics_node_update>;
```

1.5 rmq

```
template<class T>
struct RMQ {
  T st[LG + 1][N];
  int n;
  void build (const vector<T> &a) {
    n = a.size() - 1;
    for (int i = 1; i <= n; ++i) st[0][i] =
        a[i];
    for (int j = 1; j <= LG; ++j)
      for (int i = 1; i + (1 << j) - 1 <= n;
          ++i)
         st[j][i] = min(st[j - 1][i], st[j -
             1][i + (1 << (j - 1))]);
  }
  T query(int l, int r) {
    int k = __lg(r - l + 1);
    return min(st[k][l], st[k][r - (1 << k) +
        1]);
  }
};
```

1.6 segment-tree

```
// segtree with one-based indexing
//
// build:
//   segtree<int> seg(n); // vector<int> a(n
    + 1);
//   seg.build(a); // vector 'a' must be
    one-indexed
//
// update:
//   seg.update(pos, inc); //increase a[pos]
    to a[pos] + inc
//   //you may want to set this
// query:
//   seg.query(l, r)

template<class T>
struct segTree {
  vector<T> st;
  int n;
  segTree(int _n) : n(_n) {
    T default_value = 0;
    st.assign(4 * n + 5, default_value);
  }
  void build(const vector<T> &a) {
    build(a, 1, 1, n);
  }
  void build(const vector<T>& a, int id, int
      l, int r) {
    if(l == r) {
      st[id] = a[l];
      return;
    }
    int mid = (l + r) >> 1;
    build(a, id * 2, l, mid);
    build(a, id * 2 | 1, mid + 1, r);
    st[id] = st[id * 2] + st[id * 2 | 1];
  }
  void update(int pos, T inc) {
    update(pos, inc, 1, 1, n);
  }
  void update(int pos, T inc, int id, int l,
      int r) {
    if(pos < l || r < pos) {
      return;
    }
    if(l == r) {
      st[id] += inc;
      return;
    }
    int mid = (l + r) >> 1;
    update(pos, inc, id * 2, l, mid);
    update(pos, inc, id * 2 | 1, mid + 1, r);
    st[id] = st[id * 2] + st[id * 2 | 1];
  }
  T query(int u, int v) {
    return query(u, v, 1, 1, n);
  }
  T query(int u, int v, int id, int l, int r) {
    if(v < l || r < u) {
      return 0;
    }
    if(u <= l && r <= v) {
      return st[id];
    }
    int mid = (l + r) >> 1;
    return query(u, v, id * 2, l, mid) +
        query(u, v, id * 2 | 1, mid + 1, r);
  }
};
```

1.7 treap-pbds

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

/// *** Needs C++11 or C++14 ***/
/// Treap supporting duplicating values in set
/// Maximum value of treap * ADD must fit in
    long long

struct Treap { /// hash = 96814
  int len;
  const int ADD = 1000010;
  const int MAXVAL = 1000000010;
  unordered_map <long long, int> mp; ///
      Change to int if only int in treap
  tree<long long, null_type, less<long long>,
      rb_tree_tag,
      tree_order_statistics_node_update> T;

  Treap(){
    len = 0;
    T.clear(), mp.clear();
  }

  inline void clear(){
    len = 0;
    T.clear(), mp.clear();
  }

  inline void insert(long long x){
    len++, x += MAXVAL;
    int c = mp[x]++;
    T.insert((x * ADD) + c);
  }

  inline void erase(long long x){
    x += MAXVAL;
    int c = mp[x];
    if (c){
      c--, mp[x]--, len--;
      T.erase((x * ADD) + c);
    }
  }

  /// 1-based index, returns the K'th element
      in the treap, -1 if none exists
  inline long long kth(int k){
    if (k < 1 || k > len) return -1;
    auto it = T.find_by_order(--k);
    return ((*it) / ADD) - MAXVAL;
  }

  /// Count of value < x in treap
  inline int count(long long x){
    x += MAXVAL;
    int c = mp[--x];
    return (T.order_of_key((x * ADD) + c));
  }

  /// Number of elements in treap
  inline int size(){
    return len;
  }
};
```

1.8 treap

```
//replace sum with many other function, just
    like cnt
struct item {
    long long prior, value, cnt, sum;
    bool rev;
    item *l, *r;
    item(long long value): prior(rand()),
        value(value), cnt(1), rev(false),
        sum(value), l(0), r(0) {}
};

long long cnt (item* it) {
    return it ? it->cnt : 0;
}

long long sum (item* it) {
    return it ? it->sum : 0;
}

void upd_cnt (item* it) {
    if (it)
        it->cnt = cnt(it->l) + cnt(it->r) + 1;
}

void upd_sum(item* it) {
    if (it)
        it->sum = sum(it->l) + sum(it->r) +
            it->value;
}

void push (item* it) {
    if (it && it->rev) {
        it->rev = false;
        swap(it->l, it->r);
        if (it->l) it->l->rev ^= true;
        if (it->r) it->r->rev ^= true;
    }
}

void merge (item* & t, item* l, item* r) {
    push (l);
    push (r);
    if (!l || !r)
        t = l ? l : r;
    else if (l->prior > r->prior)
        merge (l->r, l->r, r), t = l;
    else
        merge (r->l, l, r->l), t = r;
    upd_cnt(t);
    upd_sum(t);
}

// split so that the number of the node in
    the left is "num"
void split (item* t, item* & l, item* & r,
    long long num, long long add = 0) {
    if (!t)
        return void( l = r = 0 );
    push (t);
    int cur_num = add + cnt(t->l);
    if (num <= cur_num)
        split (t->l, l, t->l, num, add), r = t;
    else
        split (t->r, t->r, r, num, add + 1 +
            cnt(t->l)), l = t;

    upd_cnt(t);
    upd_sum(t);
}

// l, r are 1-indexed
```

```
void reverse (item* t, int l, int r) {
    item *t1, *t2, *t3;
    split (t, t1, t2, l-1);
    split (t2, t2, t3, r-l+1);
    t2->rev ^= true;
    merge (t, t1, t2);
    merge (t, t, t3);
}

void traverse (item* t) {
    if (!t) return;
    push (t);
    traverse (t->l);
    // do stuff with current value
    cout << t->value << " ";
    traverse (t->r);
}
```

1.9 z-function

```
vector<int> z_function(const string& s) {
  int n = (int) s.size();
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r)
      z[i] = min (r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i +
        z[i]])
      ++z[i];
    if (i + z[i] - 1 > r)
      l = i, r = i + z[i] - 1;
  }
  return z;
}
```

2 graphs

2.1 DAG

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

#define pb push_back
#define fi first
#define se second

#define ii pair<int,int>
#define ll long long

using namespace std;

const int N = 3e5;
const ll MOD = 1e9 + 7;
const int iINF = 1e9;

int n,m;
vector<int> g[N];

vector<int> topo;

bool vsd[N];
void dfs(int u){
    vsd[u]=1;
    for (int v : g[u]) if (!vsd[v]) dfs(v);
    topo.pb(u);
}

void CountPath(int s){
    int d[N];

    memset(d,0,sizeof d); //number of paths
        from s to i

    int k=0;
    for (int i=0;i<=topo.size();i++) {
        if (topo[i]==s){
            k=i;
            break;
        }
    }

    d[s]=1;
    for (int i=k;i<topo.size();i++) {
        int u = topo[i];
        for (int v:g[u]) d[v]+=d[u];
    }
    cout<<"Number of paths: ";
    for (int i=1;i<=n;i++)cout<<d[i]<<" ";
    cout<<endl;
}

void LongestPath(){
    // Longest path on DAG
    int d[N]; //dp[i]: longest path starting
        from i

    for (int i=topo.size()-1;i>=0;i--){ //
        loop backwards
        int u=topo[i];
        int &res=d[u];

        res=0;
        for (int v : g[u]) res=max(res,d[v]+1);
    }
    cout<<"Longest path: ";for (int
        i=1;i<=n;i++) cout<<d[i]<<" ";
    cout<<endl;
}

int main(){
    freopen("main.inp","r",stdin);
    //Topological sorting

    cin>>n>>m;
    while (m--) {
        int u,v;cin>>u>>v;
        g[u].pb(v);
    }

    for (int i=1;i<=n;i++) if(!vsd[i]) dfs(i);
    reverse(topo.begin(),topo.end());
    for (int num:topo) cout<<num<<"
        ";cout<<endl;


    LongestPath();
    CountPath(1);


    return 0;
}
```

2.2 DFSMatching

```
/**
 * Description: Simple bipartite matching
     algorithm. Graph $g$ should be a list
 * of neighbors of the left partition, and
     $btoa$ should be a vector full of
 * -1's of the same size as the right
     partition. Returns the size of
 * the matching. $btoa[i]$ will be the match
     for vertex $i$ on the right side,
 * or $-1$ if it's not matched.
 * Time: O(VE)
 * Usage: vi btoa(m, -1); dfsMatching(g, btoa);
 * Status: works
 */

bool find(int j, vector<vi>& g, vi& btoa, vi&
    vis) {
       if (btoa[j] == -1) return 1;
       vis[j] = 1; int di = btoa[j];
       for (int e : g[di])
              if (!vis[e] && find(e, g, btoa,
                  vis)) {
                     btoa[e] = di;
                     return 1;
              }
       return 0;
}
int dfsMatching(vector<vi>& g, vi& btoa) {
       vi vis;
       rep(i,0,sz(g)) {
              vis.assign(sz(btoa), 0);
              for (int j : g[i])
                     if (find(j, g, btoa,
                         vis)) {
                            btoa[j] = i;
                            break;
                     }
       }
       return sz(btoa) -
           (int)count(all(btoa), -1);
}
```

2.3 Dinic

```
// Maximum flow from s to t in a flow
    network. O(E*V^2)
// Tested: https://oj.vnoi.info/problem/nkflow
// Tested: https://oj.vnoi.info/problem/nknet

#define ll long long
#define sz(x) (int) (x).size()

const int N = 1e3 + 5;
int n,m,s,t;
int c[N][N];

struct Dinic {
       struct Edge {
              int to, rev;
              ll c, oc;
              ll flow() { return max(oc - c,
                  0LL); } // if you need flows
       };
       vector<int> lvl, ptr, q;
       vector<vector<Edge>> adj;
       Dinic(int n) : lvl(n), ptr(n), q(n),
           adj(n) {}
       void addEdge(int a, int b, ll c, ll
           rcap = 0) {
              adj[a].push_back({b,
                  sz(adj[b]), c, c});
              adj[b].push_back({a, sz(adj[a])
                  - 1, rcap, rcap});
       }
       ll dfs(int v, int t, ll f) {
              if (v == t || !f) return f;
              for (int& i = ptr[v]; i <
                  sz(adj[v]); i++) {
                     Edge& e = adj[v][i];
                     if (lvl[e.to] == lvl[v]
                         + 1)
                            if (ll p =
                                dfs(e.to, t,
                                min(f,
                                e.c))) {
                                   e.c -= p,
                                       adj[e.to][e.rev].c
                                 += p;
                             return p;
                     }
       }
       return 0;
}
ll calc(int s, int t) {
       ll flow = 0; q[0] = s;
       for (int L = 0; L < 31; L++) do
           { // 'int L=30' maybe
           faster for random data
              lvl = ptr =
                  vector<int>(sz(q));
              int qi = 0, qe = lvl[s]
                  = 1;
              while (qi < qe &&
                  !lvl[t]) {
                     int v = q[qi++];
                     for (Edge e :
                         adj[v])
                            if
                                (!lvl[e.to]
                                &&
                                e.c
                                >>
                                (30 -
                                L))
                                   q[qe++]
                                       =
                                       e.to,
                                       lvl[e.
                                       =
                                       lvl[v]
                                       +
                                       1;
              }
              while (ll p = dfs(s, t,
                  LLONG_MAX)) flow +=
                  p;
       } while (lvl[t]);
       return flow;
}
bool leftOfMinCut(int a) { return
    lvl[a] != 0; }
};
```

```
signed main() {
    cin >> n >> m >> s >> t;
    Dinic dinic(n + 1);
    for (int i = 1; i <= m; i++) {
        int u,v,C;
        cin >> u >> v >> C;
        dinic.addEdge(u,v,C);
        c[u][v] = 1;
    }
    cout << dinic.calc(s,t) << '\n';

    /// Find all edges in min-cut, can be
         optimized using edges list
    for (int i = 1; i <= n; i++)     for (int
         j = 1; j <= n; j++) {
        if (c[i][j] && dinic.leftOfMinCut(i)
             && !dinic.leftOfMinCut(j))
            cout << i << " " << j << '\n';
    }
}
```

2.4 EdmondKarp

```
int n,m,s,t;
vector<ii> g[N];

void add(int u, int v,int w){
    g[u].pb({v,w});
    g[v].pb({u,0});
}
int bfs(int s, int t, int p[N]){
    for (int i=1;i<=n;i++)p[i]=-1;p[s]=-2;

    queue<ii> q;
    q.push({iINF,s});

    while (!q.empty()){
        int u=q.front().second;
        int flow = q.front().first;
        q.pop();

        for (ii a : g[u]){
            int v = a.first,w = a.second;

            if (w > 0 && p[v] == -1){
                p[v]=u;
                int new_flow = min(flow,w);

                if (v == t) return new_flow;

                q.push({new_flow,v});
            }
        }
    }

    return 0;
}
int max_flow(int s, int t){
    int p[N];

    int max_flow=0;

    int flow;
    while (flow = bfs(s,t,p)){
        for (int v = t; v != s; v = p[v]){
            int u = p[v];
            //g[u][v].second -= flow;
            for (ii &a : g[u]) if (a.fi == v)
                 a.se -= flow;
            //g[v][u].second += flow;
            for (ii &a : g[v]) if (a.fi == u)
                 a.se += flow;
        }

        max_flow += flow;
    }
    return max_flow;
}

/*
Remember:

After getting the max flow, the graph had
     been changed
and cannot be used anymore (Consider make a
     copy of it.)
You are using Adjacent List. Loop through all
     the edges
and adjust it.(Rather than naively using
     indexes)
*/
```

2.5 EdmondKarp$_new$

```
// Maximum flow from s to t in a flow
     network. O(E^2*V)
// Tested: https://oj.vnoi.info/problem/nkflow
// Tested: https://oj.vnoi.info/problem/nknet

const int N = 1e3 + 5;
const int INF = numeric_limits<int>::max() /
     2;
int c[N][N], par[N], f[N][N], d[N];
int n,m,s,t;

int bfs(int s, int t) {
    queue<int> q;
    fill(par, par + N, -1);
    fill(d, d + N, 0);
    q.push(s); d[s] = INF;
    while (!q.empty()) {
        int u = q.front();    q.pop();
        for (int i = 1; i <= n; i++) {
            if (c[u][i] - f[u][i] > 0 && par[i]
                 == -1) {
                d[i] = min(d[u], c[u][i] -
                     f[u][i]);
                par[i] = u;    q.push(i);
                if (i == t)    return d[t];
            }
        }
    }
    return 0;
}

int max_flow(int s, int t) {
    int cur = 0, ans = 0;
    while ((cur = bfs(s,t)) > 0) {
        ans += cur;
        int v = t;
        while (v != s) {
            f[par[v]][v] += cur;
            f[v][par[v]] -= cur;
            v = par[v];
        }
    }
    return ans;
}

int vis[N];
void dfs_min_cut(int u) {    /// Find all
     edges in Min-Cut, edges u-v in Min-Cut
     has vis[u] = 1 and vis[v] = 0
    vis[u] = 1;
    for (int v = 1; v <= n; v++)     {
        if (abs(f[u][v]) < c[u][v] && !vis[v])
             dfs_min_cut(v);
    }
}

vector<pair<int, int>> min_cut() { // Can
     optimized using edge list
    dfs_min_cut(s);
    vector<pair<int, int>> ans;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            if (f[i][j] && vis[i] && !vis[j])
                ans.push_back({i,j});
    return ans;
}

signed main() {
    cin >> n >> m >> s >> t;
    for (int i = 1; i <= m; i++) {
        int u,v,C;
        cin >> u >> v >> C;
        c[u][v] += C;
    }
    cout << max_flow(s,t) << '\n';
    auto v = min_cut();
    for (auto ii : v)     cout << ii.first <<
         " " << ii.second << '\n';
    return 0;
}
```

2.6 HLD

```
/**
 * You are given a tree
 * We will ask you to perfrom some
     instructions of the following form:
 * CHANGE i ti : change the cost of the i-th
     edge to ti
 * or
 * QUERY a b : ask for the maximum edge cost
     on the path from node a to node b
 * Status: Tested:
     https://vjudge.net/problem/SPOJ-QTREE
*/

#include<bits/stdc++.h>
using namespace std;

#define int long long
#define pb push_back
const int N = 1e5 + 5;

int sz[N];
int n, dfsTime;
int nhanh, cnt;
int id[N]; // <=> thu tu cua chain chua dinh
     i = id[i]
int head[N]; // <=> gia tri dau tien cua
     chain thu i = head[i]
int pos[N]; // <=> thu tu cua dinh u sau khi
     duoc trai phang = pos[i]
int in[N], out[N], h[N], a[N], p[N][20];
vector<int> adj[N];

struct seg2D { // 0-index
    int n;
    vector<int> seg;
    seg2D() = default;
    seg2D(int _n) : seg(2 * _n, INT_MIN),
        n(_n) {}
    void update(int u, int val) {
        for (seg[u += n] = val; u > 1; u /= 2)
            {
            seg[u >> 1] = max(seg[u], seg[u ^
                1]);
            }
    }
    int get(int u, int v) {
        int ans = INT_MIN;
        for (u += n,v += n; u <= v; u/=2, v /=
            2) {
            if (u & 1)     ans = max(ans,
                seg[u++]);
            if (~v & 1)    ans = max(ans,
                seg[v--]);
        }
        return ans;
    }
}seg;


void reset() {
    nhanh = 1;
    dfsTime = 0;
    cnt = 0;
    for (int i = 1; i <= n; i++){
        adj[i].clear();
        h[i] = 0;
        in[i] = 0;
        out[i] = 0;
        pos[i] = 0;
        head[i] = 0;
        id[i] = 0;
        a[i] = 0;
        sz[i] = 1;
    }
    memset(p,0,sizeof p);
}


void dfs(int u, int par = -1) {
    p[u][0] = par;
    for (int i = 1; i < 20; i++)     p[u][i] =
         p[p[u][i-1]][i-1];
    ++dfsTime;
    in[u] = dfsTime;
    sz[u] = 1;
    for (int v : adj[u]) if (v != par) {
        h[v] = h[u] + 1;
        dfs(v,u);
        sz[u]+=sz[v];
    }
    out[u] = dfsTime;
}

bool cha(int u, int v) {
    return (in[u] <= in[v] && out[u] >=
        out[v]);
}

void build_hld(int u, int par = -1) {
    if (head[nhanh] == 0) {
        head[nhanh] = u;
    }
    ++cnt;
    pos[u] = cnt;
    id[u] = nhanh;

    int tmp = -1;
    for (int v : adj[u])     if (v != par) {
        if (tmp == -1 || sz[tmp] < sz[v])
            tmp = v;
    } // Tim dinh v co sz lon nhat

    // heavy edge
    if (tmp != -1)    build_hld(tmp,u);

    for (int v : adj[u])  if (v != par && v !=
        tmp) {
        /// light edge
        nhanh++;
        build_hld(v,u);
    }
}

int ans_hld(int u, int v) {  // the maximum
    edge cost on the path from node u to node
    v
    int ans = INT_MIN;
    while (id[u] != id[v]) {
        if (h[head[id[u]]] > h[head[id[v]]])
            swap(u,v);
        int v2 = head[id[v]];
        ans = max(ans, seg.get(pos[v2] - 1,
            pos[v] - 1));
        v = p[v2][0];
    }
    /// id[u] == id[v]
    if (pos[u] > pos[v])  swap(u,v);
    ans = max(ans, seg.get(pos[u], pos[v] -
        1)); /// Khi u va v o cung nhanh, ta
        chi nhay tu v len dinh gan u nhat vi u
        o day la lca (ta phai bo a[lca] di do
        qua trinh chuyen duong di)
    return ans;
}

int lca(int u, int v) { // Bin lift
    if (h[u] < h[v])     swap(u,v);
    int k = h[u] - h[v];
    for (int j = 0; j < 20; j++) if (k >> j &
        1)   u = p[u][j];
    if (u == v)   return u;
    for (int i = 19; i >= 0; i--) {
        if (p[u][i] != p[v][i])  u = p[u][i],
            v = p[v][i];
    }
    return p[u][0];
}

void precalc() {
    dfs(1,1);
    build_hld(1);
}

void solve() {
    cin >> n;
    reset();
    vector<array<int,3>> ed(n + 1);
    for (int i = 1; i < n; i++)  {
        int u,v,w; cin >> u >> v >> w;
        ed[i] = {u,v,w};
        adj[u].pb(v);
        adj[v].pb(u);
    }
    precalc();
    seg = seg2D(cnt);
    for (int i = 1; i < n; i++) {
        int u = ed[i][0], v = ed[i][1], w =
            ed[i][2];
        if (!cha(u,v))    swap(u,v);
        a[v] = w;
        seg.update(pos[v] - 1, w);
    }
    while (1) {
        string typ;
        cin >> typ;
        if (typ == "DONE")    return;
        if (typ == "QUERY") {
            int u,v;   cin >> u >> v;
            int LCA = lca(u,v);
            cout << max(ans_hld(u,LCA),
                ans_hld(v,LCA)) << '\n';
        }
        else {
            int id;    cin >> id;
            int nval;  cin >> nval;
            int u = ed[id][0], v = ed[id][1], w
                = ed[id][2];
            if (!cha(u,v))    swap(u,v);    //
                v la con u
            a[v] = nval;
            ed[id][2] = nval;
            seg.update(pos[v] - 1, nval);
        }
    }
}

int32_t main() {
    ios_base :: sync_with_stdio(0);
        cin.tie(nullptr); cout.tie(nullptr);
    int test;  cin >> test;
    while (test--) {
        solve();
        cout << '\n';
    }
    return 0;
}

/*
1
3
1 2 1
2 3 2
QUERY 1 2
```

```
CHANGE 1 3
QUERY 1 2
DONE
*/
```

2.7 HLD$_2$

```
/**
 * We will ask you to perfrom some
     instructions of the following form:
 * 0 i : change the color of the i-th node
     (from white to black, or from black to
     white);
 * or
 * 1 v : ask for the id of the first black
     node on the path from node 1 to node v.
     if it doesn't exist, you may return -1
     as its result.
 * Tested:
     https://www.spoj.com/problems/QTREE3/en/
*/

#include<bits/stdc++.h>
using namespace std;

#define int long long
#define pb push_back
const int N = 1e6 + 5;

vector<int> adj[N];
int id[N], h[N], head[N], sz[N], pos[N], e[N];
int p[20][N];
int nhanh = 1, cnt;
set<int> s;

void dfs(int u, int par = 1) {
    sz[u] = 1;
    p[0][u] = par;
    for (int i = 1; i < 20; i++)     p[i][u] =
        p[i - 1][p[i - 1][u]];
    for (int v : adj[u])  if (v != par)  {
        h[v] = h[u] + 1;
        dfs(v,u);
        sz[u] += sz[v];
    }
}

void hld(int u, int par = -1) {
    if (head[nhanh] == 0)    head[nhanh] = u;
    ++cnt;
    e[cnt] = u;
    pos[u] = cnt;
    id[u] = nhanh;

    int target = -1;
    for (int v : adj[u])  if (v != par)  {
        if (target == -1 || sz[target] <
            sz[v]) target = v;
    }

    if (target != -1)     hld(target, u);

    for (int v : adj[u])  if (v != par)  {
        if (v != target)      {
            ++nhanh;
            hld(v,u);
        }
    }
}

void init() {
    dfs(1);
    hld(1);
}

int query(int l, int r) {
    auto it = s.lower_bound(l);
    if (it == s.end()) return INT_MAX;
    if (*it >= l && *it <= r)    return (*it);
    return INT_MAX;
}

int ans_hld(int u, int v) {  // u la cha v
    int ans = INT_MAX;
    while (id[u] != id[v]) {
        int v2 = head[id[v]];
        ans = min(ans, query(pos[v2], pos[v]));
        v = p[0][v2];
    }
    ans = min(ans, query(pos[u], pos[v]));
    if (ans == INT_MAX)   return -1;
    return e[ans];
}

void solve() {
    int n,q;   cin >> n >> q;
    for (int i = 1; i < n; i++)  {
        int u,v;   cin >> u >> v;
        adj[u].pb(v);  adj[v].pb(u);
    }
    init();
    while (q--) {
        int typ;   cin >> typ;
        if (typ == 1)  {
            int v;   cin >> v;
            cout << ans_hld(1,v) << '\n';
        }
        else {
            int v;  cin >> v;
            if (s.find(pos[v]) != s.end())
                s.erase(s.find(pos[v]));
            else s.insert(pos[v]);
        }
    }
}

int32_t main() {
    ios_base :: sync_with_stdio(0);
        cin.tie(nullptr); cout.tie(nullptr);
    solve();
    return 0;
}

/*
9 8
1 2
1 3
2 4
2 9
5 9
7 9
8 9
6 8
```

```
1 3
0 8
1 6
1 7
0 2
1 9
0 2
1 9
*/
```

2.8 HLD$_3$

```
/**
 * Given a rooted tree of nodes, where each
     node is uniquely numbered in between
     [1..N]. The node 1 is the root of the
     tree. Each node has an integer value
     which is initially 0.
 * You need to perform the following two kinds
     of queries on the tree:
    * add t value: Add value to all nodes in
        subtree rooted at t
    * max a b: Report maximum value on the
        path from a to b
 * Tested:
     https://vjudge.net/problem/HackerRank-subtrees-and-paths
*/

#include<bits/stdc++.h>

using namespace std;

#define int long long
#define pb push_back

const int INF = 1e9 + 7;
const int N = 2e5 + 5;

vector<int> adj[N];
int id[N], head[N], sz[N], in[N], out[N],
    h[N], pos[N];
int p[20][N];
int nhanh = 1, cnt;

int seg[4 * N], lazy[4 * N];

void down(int id) {
    int t = lazy[id];
    int lhs = id * 2, rhs = id * 2 + 1;
    seg[lhs] += t; seg[rhs] += t;
    lazy[lhs] += t;   lazy[rhs] += t;
    lazy[id] = 0;
}

void update(int id, int l, int r, int u, int
    v, int val) {
    if (l > v || r < u)   return;
    if (l >= u && r <= v) {
        seg[id] += val;
        lazy[id] += val;
        return;
    }
    down(id);
    int mid = (l + r) / 2;
    update(id * 2, l, mid, u, v, val);
    update(id * 2 + 1, mid + 1, r, u, v, val);
    seg[id] = max(seg[id * 2], seg[id * 2 +
         1]);
}

int get(int id, int l, int r, int u, int v) {
    if (l > v || r < u)   return -INF;
    if (l >= u && r <= v) return seg[id];
    down(id);
    int mid = (l + r) / 2;
    return max(get(id * 2, l, mid, u, v),
         get(id * 2 + 1, mid + 1, r, u, v));
}

void dfs(int u, int par = 1) {
    sz[u] = 1;
    p[0][u] = par;
    for (int i = 1; i < 20; i++)     p[i][u] =
         p[i - 1][p[i-1][u]];
    for (int v : adj[u])  if (v != par) {
        h[v] = h[u] + 1;
        dfs(v,u);
        sz[u] += sz[v];
    }
}

void hld(int u, int par = -1) {
    if (head[nhanh] == 0)    {
        head[nhanh] = u;
    }
    ++cnt;
    in[u] = cnt;
    id[u] = nhanh;
    pos[u] = cnt;

    int target = -1;
    for (int v : adj[u])  if (v != par) {
        if (target == -1 || sz[target] < sz[v])
            target = v;
    }

    if (target != -1) hld(target,u);

    for (int v : adj[u])  if (v != par) {
        if (v != target) {
            ++nhanh;
            hld(v,u);
        }
    }
    out[u] = cnt;
}

int query(int u, int v) {
    int ans = -INF;
    while (id[u] != id[v]) {
        if (h[head[id[u]]] > h[head[id[v]]])
             swap(v,u);
        int v2 = head[id[v]];
        ans = max(ans, get(1,1,cnt,pos[v2],
             pos[v]));
        v = p[0][v2];
    }
    if (pos[u] > pos[v])     swap(u,v);
    ans = max(ans, get(1,1,cnt,pos[u],
         pos[v]));
    return ans;
}
```

```
void solve() {
    int n; cin >> n;
    for (int i = 1; i < n; i++)  {
        int u,v;   cin >> u >> v;
        adj[u].pb(v);
        adj[v].pb(u);
    }

    dfs(1);
    hld(1);

    int q; cin >> q;
    while (q--) {
        string typ;   cin >> typ;
        if (typ == "add") {
            int u,val;    cin >> u >> val;
            update(1,1,cnt,in[u], out[u], val);
        }
        else {
            int u,v;   cin >> u >> v;
            cout << query(u,v) << '\n';
        }
    }
}

int32_t main() {
    ios_base :: sync_with_stdio(0);
         cin.tie(nullptr); cout.tie(nullptr);
    solve();
    return 0;
}

/*
5
1 2
2 3
2 4
5 1
6
add 4 30
add 5 20
max 4 5
add 2 -20
max 4 5
max 3 4
*/
```

2.9 HopcroftKarp

```
/**
 * Description: Fast bipartite matching
     algorithm. Graph $g$ should be a list
 * of neighbors of the left partition, and
     $btoa$ should be a vector full of
 * -1's of the SAME SIZE as the right
     partition. Returns the size of
 * the matching. btoa[i] will be the match for
     vertex i on the right side,
 * or -1 if it's not matched.
 * Usage: vector<int> btoa(m, -1);
     hopcroftKarp(g, btoa);
 * Time: O(sqrt(V)*E)
 * Tested: https://oj.vnoi.info/problem/match1
 * Tested: https://oj.vnoi.info/problem/fmatch
 */

bool dfs(int a, int L, vector<vector<int>>&
    g, vector<int>& btoa, vector<int>& A,
    vector<int>& B) {
       if (A[a] != L) return 0;
       A[a] = -1;
       for (int b : g[a]) if (B[b] == L + 1) {
              B[b] = 0;
              if (btoa[b] == -1 ||
                  dfs(btoa[b], L + 1, g,
                  btoa, A, B))
                     return btoa[b] = a, 1;
       }
       return 0;
}

int hopcroftKarp(vector<vector<int>>& g,
    vector<int>& btoa) {
       int res = 0;
       vector<int> A(g.size()),
           B(btoa.size()), cur, next;
       for (;;) {
              fill(A.begin(), A.end(), 0);
              fill(B.begin(), B.end(), 0);
              /// Find the starting nodes for
                  BFS (i.e. layer 0).
              cur.clear();
              for (int a : btoa) if(a != -1)
                  A[a] = -1;
              for (int a = 0; a < g.size();
                  a++) if(A[a] == 0)
                  cur.push_back(a);
              /// Find all layers using bfs.
              for (int lay = 1;; lay++) {
                     bool islast = 0;
                     next.clear();
                     for (int a : cur) for
                         (int b : g[a]) {
                            if (btoa[b] ==
                                -1) {
                                   B[b] =
                                       lay;
                                   islast =
                                       1;
                            }
                            else if (btoa[b]
                                != a &&
                                !B[b]) {
                                   B[b] =
                                       lay;
                                   next.push_back(bt
                            }
                     }
                     if (islast) break;
                     if (next.empty()) return
                         res;
                     for (int a : next) A[a]
                         = lay;
                     cur.swap(next);
              }
              /// Use DFS to scan for
                  augmenting paths.
       for (int a = 0; a < g.size(); a++)
                     res += dfs(a, 0, g,
                         btoa, A, B);
       }
}
```

```
signed main() {
    int n,m,p;
    cin >> n >> m >> p;
    vector<vector<int>> g(n + 1,vector<int>());
    while (p--) {
        int u,v;
        cin >> u >> v;
        g[u].push_back(v);
    }
    vector<int> btoa(m + 1, -1);
    cout << hopcroftKarp(g, btoa) << '\n';

    /// Find all matching, btoa[i] is the left
         side of i(The right side)
    for (int i = 0; i < btoa.size(); i++)
        if (btoa[i] != -1)
            cout << btoa[i] << "->" << i <<
                 "\n";
}
```

2.10 MinimumVertexCover

```
/**
 * Description: Finds a minimum vertex cover
      in a bipartite graph.
 *  The size is the same as the size of a
      maximum matching, and
 *  the complement is a maximum independent
      set.
 * Status: stress-tested
 */

/**
 * DAG
 * Maximum Independent Set == Minimum Path
      Cover
 * Minimum Path Cover = n - Maximum Bipartie
      Matching
 * Minimum vertex cover of DAG == Maximum
      Bipartie Matching
 */

#include "DFSMatching.cpp"
```

```
vi cover(vector<vi>& g, int n, int m) {
        vi match(m, -1);
        int res = dfsMatching(g, match);
        vector<bool> lfound(n, true), seen(m);
        for (int it : match) if (it != -1)
            lfound[it] = false;
        vi q, cover;
        rep(i,0,n) if (lfound[i])
            q.push_back(i);
        while (!q.empty()) {
                int i = q.back(); q.pop_back();
                lfound[i] = 1;
                for (int e : g[i]) if (!seen[e]
                    && match[e] != -1) {
                        seen[e] = true;
                        q.push_back(match[e]);
                }
        }
        rep(i,0,n) if (!lfound[i])
            cover.push_back(i);
        rep(i,0,m) if (seen[i])
            cover.push_back(n+i);
        assert(sz(cover) == res);
        return cover;
}
```

2.11 Minimum$_{path_{C}over_{i}n_{D}AG}$

```
#include <bits/stdtr1c++.h>

#define MAX 505
#define clr(ar) memset(ar, 0, sizeof(ar))
#define read() freopen("lol.txt", "r", stdin)
#define dbg(x) cout << #x << " = " << x <<
    endl
#define ran(a, b) ((((rand() << 15) ^ rand())
    % ((b) - (a) + 1)) + (a))

using namespace std;
/// Minimum path cover/Maximum independent
    set in DAG
namespace dag{
    /// For transitive closure and minimum
        path cover with not necessarily
        disjoint vertex
    bool ar[MAX][MAX];

    vector <int> adj[MAX];
    bool visited[MAX], first_set[MAX],
         second_set[MAX];
    int n, L[MAX], R[MAX], D[MAX], Q[MAX],
         dis[MAX], parent[MAX];

    inline void init(int nodes){ /// Number of
         vertices in DAG
        n = nodes;
        for (int i = 0; i < MAX; i++)
            adj[i].clear();
    }

    inline void add_edge(int u, int v){ /// 0
         based index, directed edge of DAG
        adj[u].push_back(v);
    }

    bool dfs(int i){
        int len = adj[i].size();
        for (int j = 0; j < len; j++){
            int x = adj[i][j];
            if (L[x] == -1 || (parent[L[x]] ==
                i)){
                if (L[x] == -1 || dfs(L[x])){
                    L[x] = i, R[i] = x;
                    return true;
                }
            }
        }
        return false;
    }

    bool bfs(){
        clr(visited);
        int i, j, x, d, f = 0, l = 0;

        for (i = 0; i < n; i++){
            if (R[i] == -1){
                visited[i] = true;
            Q[l++] = i, dis[i] = 0;
        }
    }

    while (f < l){
        i = Q[f++];
        int len = adj[i].size();
        for (j = 0; j < len; j++){
            x = adj[i][j], d = L[x];
            if (d == -1) return true;

            else if (!visited[d]){
                Q[l++] = d;
                parent[d] = i, visited[d] =
                    true, dis[d] = dis[i] +
                    1;
            }
        }
    }
    return false;
}

void get_path(int i){
    first_set[i] = true;
    int j, x, len = adj[i].size();

    for (j = 0; j < len; j++){
        x = adj[i][j];
        if (!second_set[x] && L[x] != -1){
            second_set[x] = true;
            get_path(L[x]);
        }
    }
}

void transitive_closure(){ /// Transitive
    closure in O(n * m)
    clr(ar);
    int i, j, k, l;
    for (i = 0; i < n; i++){
        l = adj[i].size();
        for (j = 0; j < l; j++){
            ar[i][adj[i][j]] = true;
        }
        adj[i].clear();
```

```
    }

    for (k = 0; k < n; k++){
        for (i = 0; i < n; i++){
            if (ar[i][k]){
                for (j = 0; j < n; j++){
                    if (ar[k][j]) ar[i][j] =
                        true;
                }
            }
        }
    }

    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            if (i != j && ar[i][j]){
                adj[i].push_back(j);
            }
        }
    }
}
/// Minimum vertex disjoint path cover in
    DAG. Handle isolated vertices
    appropriately
int minimum_disjoint_path_cover() {
    int i, res = 0;
    memset(L, -1, sizeof(L));
    memset(R, -1, sizeof(R));

    while (bfs()){
        for (i = 0; i < n; i++){
            if (R[i] == -1 && dfs(i)) res++;
        }
    }

    return n - res;
}

int minimum_path_cover(){ /// Minimum path
    cover in DAG. Handle isolated vertices
    appropriately
    transitive_closure();
    return minimum_disjoint_path_cover();
}
```

```
    /// Minimum vertex cover of DAG, equal to
        maximum bipartite matching
    vector <int> minimum_vertex_cover(){
        int i, res = 0;
        memset(L, -1, sizeof(L));
        memset(R, -1, sizeof(R));

        while (bfs()){
            for (i = 0; i < n; i++){
                if (R[i] == -1 && dfs(i)) res++;
            }
        }

        vector <int> v;
        clr(first_set), clr(second_set);
        for (i = 0; i < n; i++){
            if (R[i] == -1) get_path(i);
        }

        for (i = 0; i < n; i++){
            if (!first_set[i] || second_set[i])
                v.push_back(i);
        }

        return v;
    }
    /// Maximum independent set of DAG, all
        vertices not in minimum vertex cover
    vector <int> maximum_independent_set() {
        vector <int> v =
            minimum_vertex_cover();
        clr(visited);
        int i, len = v.size();
        for (i = 0; i < len; i++)
            visited[v[i]] = true;

        vector <int> res;
        for (i = 0; i < n; i++){
            if (!visited[i]) res.push_back(i);
        }
        return res;
    }
}
```

2.12 Tarjan

```
int n,m;
vector<int> g[N];

bool onStack[N];
stack<int> st;
int tin[N], low[N];
int dfsTime=0;

int SCCCount=0;
/*
memset(tin,0,sizeof tin);
memset(onStack,false,sizeof onStack);
*/

void dfs(int u) {
    onStack[u]=true;st.push(u);
    low[u]=tin[u]=++dfsTime;

    for(int v : g[u]){
       // cout<<u<<" -> "<<v<<endl;
        if (!tin[v]) dfs(v);
        if (onStack[v])
             low[u]=min(low[u],low[v]); //
             low[v] not tin[v]
    }

    if (low[u]==tin[u]){ // new SCC found
        while (!st.empty()){
            int a = st.top();
            onStack[a]=false;st.pop();
            printf("%d ",a);
            if (a == u){
                printf("\n");
                break;
            }
        }
        SCCCount++;
    }
}
```

2.13 bridge

```
int n, m;
vector<int> g[N];

int tin[N], low[N];
int dfsTime = 0;

bool articulation_point[N];
int bridge_count = 0;


void dfs(int u, int pre = -1) {
    tin[u] = low[u] = ++dfsTime;
    int child = 0;

    for (int v : g[u]) {
        if (v == pre) continue;
        if (tin[v]) low[u] = min(low[u],
             tin[v]);
        else {
            dfs(v, u);
            low[u] = min(low[u], low[v]);
            child++;

            if (low[v] > tin[u]) bridge_count++;
            if (low[v] >= tin[u] && pre != -1)
                 articulation_point[u] = true;
        }
    }

    if (pre == -1 && child >= 2)
         articulation_point[u] = true;
}
```

2.14 dijkstra

```
struct dijkstra {
        int n;
        vector<vector<pair<int, long long>> >
             g;
        int s;
        vector<long long> dist;

        dijkstra(int n): n(n) {
                g.assign(n + 1,
                     vector<pair<int, long long>
                     >());
                dist.assign(n + 1, 1e18);
        }

        void add_edge(int u, int v, ll c) {
                // change this whether directed
                     or not
                g[u].push_back(make_pair(v,c));
                g[v].push_back(make_pair(u,c));
        }

        void build() {
                set<pair<long long, int> > st;

                st.insert(make_pair(0, s));
                     dist[s] = 0;

                while (st.size()) {
                        long long du =
                             st.begin()->fi; int
                             u = st.begin()->se;
                        st.erase(st.begin());
                        if (du != dist[u])
                             continue;

                        for (auto &[v, c] :
                             g[u]) {
                                if (dist[v] >
                                     dist[u] + c)
                                     {
                                        dist[v] =
                                             dist[u]
                                             + c;
                                        st.insert(make_pa
                                             v));
                                }
                        }
                }
        }
};
```

2.15 dsu

```
class dsu {
  vector<int> parent, size;

public:
  dsu() {}

  dsu(int n) {
    parent.resize(n);
    size.assign(n, 1);
    iota(all(parent), 0);
  }

  int find(int v) {
    if (v == parent[v]) return v;
    return parent[v] = find(parent[v]);
  }

  bool unite(int a, int b) {
    a = find(a);
    b = find(b);
    if (a != b) {
      if (size[a] < size[b]) swap(a, b);
      parent[b] = a;
      size[a] += size[b];
      return true;
    } else {
      return false;
    }
  }
};
```

2.16 lca-binarylift

```
/**
 * Description: Finding LCA and Kth Ancestor
     using Binary Lifting
 * Caution:
 *    * adj must be one-indexed
 *    * root must be 1 (can be modified if
     required)
 * Source :
     https://oj.vnoi.info/submission/4618381
 * Verification :
     https://oj.vnoi.info/problem/hbtlca
 *
 * usage LCA lca(n + 1, root) // default root
     is 1
 */

struct LCA {
  vector<vector<int>> par;
  vector<int> dep;
  int LG;
  void dfs(int u, int p = 0) {
    par[u][0] = p;
    for (int i = 1; i < LG; i++)
      par[u][i] = par[par[u][i - 1]][i - 1];
    for (int v : adj[u]) {
      if (v == p) continue;
      dep[v] = dep[u] + 1;
      dfs(v, u);
    }
  }

  int ancestor(int u, int k) {
    for (int i = 0; i < LG; i++)
      if (k & (1 << i))
        u = par[u][i];
    return u;
  }

  int lca(int u, int v) {
    if (dep[u] < dep[v]) swap(u, v);
    u = ancestor(u, dep[u] - dep[v]);
    if (u == v) return u;
    for (int i = LG - 1; i >= 0; i--)
      if (par[u][i] != par[v][i])
        u = par[u][i], v = par[v][i];
    return par[u][0];
  }

  LCA (int _n, int root = 1) {
    int n = _n;
    LG = 64 - __builtin_clzll(n);
    par.assign(n, vector<int>(LG, 0));
    dep.assign(n, 0);
    dfs(root);
  }
};
```

2.17 tree-diameter

```
/**
 * Description: Finding the diameter of tree
     in O(V + E)
 */

struct TD {
  int ans = 0;
  vector<vector<int>> g;

  int Dfs (int u, int pr) {
    pi lol = {0, 0};
    for (int v : g[u]) {
      if (v == pr) continue;
      int te = Dfs(v, u);
      if (lol.ff < te) {
        lol.ss = lol.ff;
        lol.ff = te;
      } else if (lol.ss < te) {
        lol.ss = te;
      }
    }
    ans = max(ans, lol.ff + lol.ss);
    return lol.ff + 1;
  }

  int run() {
    Dfs(1, 1);
    return ans;
  }

  TD(vector<vector<int>>& _g) : g(_g) {}
};
```

3 mathematics

3.1 MI

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

#define ii pair<int,int>
#define ll long long

using namespace std;

const int N = 3e5;
const ll MOD = 1e9 + 7;

//Necessary function
ll mul(ll a, ll b){return
    (a%MOD)*(b%MOD)%MOD;}
ll add(ll a, ll b){return ((a+MOD)%MOD +
    (b+MOD)%MOD)%MOD;}
ll fpow(ll a, ll b){
    ll res=1;
    while (b){
        if (b & 1) res=mul(res,a);
        b>>=1;
        a=mul(a,a);
    }
    return res;
}
ll inv(ll a) {return fpow(a,MOD-2);}

// Euler's totient function. Source:
    cp-algorithms.com
vector<ll> phi_1_to_n(int n) {
    vector<ll> 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;
        }
    }
    return phi;
}

int main(){
    //freopen("main.inp","r",stdin);

    /*
    Calculate MI of factorials
    inv(fact[i]%MOD) (calculate the factorial
        with modulo first then calc the
        inverse.)
    */

    vector<ll> factinv(N,0), fact(N,1);
    for (int i=1;i<N;i++) {
        fact[i]=mul(fact[i-1],i);
        factinv[i]=inv(fact[i]);

        if (i <= 10) cout<<fact[i]<<" ";
    }cout<<endl;

    assert(mul(fact[4],factinv[4]) == 1);
}
```

3.2 Point-Line

```
#define pt pair<long long, long long>
#define x first
#define y second

struct Line {
    pt a, ab;

    Line(pt _a, pt _ab): a(_a), ab(_ab) {}
};

pt operator - (const pt& a, const pt& b)
    {return make_pair(a.x - b.x, a.y - b.y);}

ll operator * (const pt& a, const pt& b)
    {return a.x*b.x + a.y*b.y;}
ll operator ^ (const pt& a, const pt& b)
    {return a.x*b.y - a.y*b.x;}

pt refine(pt a) { // only for pt = pair<ll,
    ll>
    ll g = gcd(a.x, a.y);
    if (!g) return a;

    if (g < 0) g = -g;
    a.x /= g, a.y /= g;
    return a;
}

int dir(pt a, pt b, pt c) {
    ll x = (b - a) ^ (c - a);

    if (x == 0) return 0;
    else if (x > 0) return 1;
    else return 2;
}
```

3.3 combinatorics-power

```
vector<Mint> fact(1, 1);
vector<Mint> inv_fact(1, 1);

Mint power(const Mint a, const int b) {
  assert(b >= 0);
  Mint x = a, res = 1;
  U p = b;
  while (p > 0) {
    if (p & 1) res *= x;
    x *= x;
    p >>= 1;
  }
  return res;
}

/*
Mint C(int n, int k) {
  if (k < 0 || k > n) {
    return 0;
  }
  k = min(k, n - k);
  while ((int) fact.size() < k + 1) {
    fact.push_back(fact.back() * (int)
        fact.size());
    inv_fact.push_back(1 / fact.back());
  }
  Mint ret = inv_fact[k];
  for (int i = 1; i <= k; i++) {
    ret *= n - i + 1;
  }
  return ret;
}
*/

Mint C(int n, int k) {
  if (k < 0 || k > n) {
    return 0;
  }
  k = min(k, n - k);
  while ((int) fact.size() < n + 1) {
    fact.push_back(fact.back() * (int)
        fact.size());
    inv_fact.push_back(1 / fact.back());
  }
  return fact[n] * inv_fact[k] * inv_fact[n -
      k];
}
```

3.4 convex-hull

```
struct Point {
    long long x, y;
};

bool operator < (const Point & a, const Point
    & b) {
    return ii(a.x, a.y) < ii(b.x, b.y);
}

ostream & operator << (ostream & out, const
    Point& a) {
    out << "(" << a.x << ", " << a.y << ")";
    return out;
}

istream & operator >> (istream & is, Point&
    a) {
    is >> a.x >> a.y;
    return is;
}

// A -> B -> C form counterclockwise direction
bool ccw(const Point &A, const Point &B,
    const Point &C) {
    return (B.x - A.x) * (C.y - A.y) - 1LL *
        (C.x - A.x) * (B.y - A.y) > 0;
}

// CONVEX HULL (CLOCKWISE)
// Source: from VNOI Wiki with small
    modification
// for speed might change to vector<Point>& p
vector<Point> convexHull(vector<Point> p) {
    int n = p.size();

    if (n < 3) {
        // cannot form convex hull
        return vector<Point>();
    }

    // SORT POINTS
    sort(p.begin(), p.end(), [](const Point
        &A, const Point &B) {
        if (A.x != B.x) return A.x < B.x;
        return A.y < B.y;
    });

    vector<Point> hull;
    hull.push_back(p[0]);

    // UPPER HULL
    for (int i = 1; i < n; ++i) {
        while (hull.size() >= 2 &&
             ccw(hull[hull.size() - 2],
             hull.back(), p[i])) {
            hull.pop_back();
        }
        hull.push_back(p[i]);
    }

    // LOWER HULL
    for (int i = n - 2; i >= 0; --i) {
        while (hull.size() >= 2 &&
             ccw(hull[hull.size() - 2],
             hull.back(), p[i])) {
            hull.pop_back();
        }
        hull.push_back(p[i]);
    }

    // REMOVE 1 OVERLAPPING
    if (n > 1) hull.pop_back();

    return hull;
}
```

3.5 matrix-mod

```
//Template from VNOI:
    https://wiki.vnoi.info/algo/trick/matrix-multipli
    with some modification
// ofc this is 0-indexed
template<typename T>
struct matrix_mod {
    T add(T a, T b) {
        return ((a+MOD)%MOD + (b+MOD)%MOD)%MOD;
    }
    T mul(T a, T b) {
        a *= b;
        if (a > MOD) a %= MOD;
        return a;
    }
    vector <vector <T> > data;

    int row() const { return data.size(); }
    int col() const { return data[0].size(); }

    auto & operator [] (int i) { return
        data[i]; }
    const auto & operator[] (int i) const {
        return data[i]; }

    matrix_mod() = default;
matrix_mod(int r, int c): data(r, vector
    <T> (c)) { }
matrix_mod(const vector <vector <T> > &d):
    data(d) {
    assert(d.size());
    int size = d[0].size();
    assert(size);
    for (auto x : d) assert(x.size() ==
        size);
}

friend ostream & operator << (ostream
    &out, const matrix_mod &d) {
    for (auto x : d.data) {
        for (auto y : x) out << y << ' ';
        out << '\n';
    }
    return out;
}

static matrix_mod identity(long long n) {
    matrix_mod a = matrix_mod(n, n);
    while (n--) a[n][n] = 1;
    return a;
}

matrix_mod operator * (const matrix_mod
    &b) {
    matrix_mod a = *this;
    assert(a.col() == b.row());
    matrix_mod c(a.row(), b.col());
    for (int i = 0; i < a.row(); ++i)
        for (int j = 0; j < b.col(); ++j)
            for (int k = 0; k < a.col();
                ++k)
                c[i][j] = add(c[i][j],
                    mul(a[i][k], b[k][j]));
    return c;
}

matrix_mod pow(long long exp) {
    assert(row() == col());
    matrix_mod base = *this, ans =
        identity(row());
        for (; exp > 0; exp >>= 1, base = base
            * base)
            if (exp & 1) ans = ans * base;
        return ans;
    }
};
```

3.6 modint

```
/*
 * Caution:
 *   * need to cast into int before printing
 * source: Benq
*/
struct Mint {
  int v;
  explicit operator int() const { return v; }
  Mint() { v = 0; }
  Mint(long long _v) : v(_v % MOD) { v += (v <
      0) * MOD; }
};
Mint &operator+=(Mint &a, Mint b) {
  if ((a.v += b.v) >= MOD) a.v -= MOD;
  return a;
}
Mint &operator-=(Mint &a, Mint b) {
  if ((a.v -= b.v) < 0) a.v += MOD;
  return a;
}
Mint operator+(Mint a, Mint b) { return a +=
    b; }
Mint operator-(Mint a, Mint b) { return a -=
    b; }
Mint operator*(Mint a, Mint b) { return
    Mint((long long)a.v * b.v); }
Mint &operator*=(Mint &a, Mint b) { return a
    = a * b; }
Mint pow(Mint a, long long p) {
  assert(p >= 0);
  return p == 0 ? 1 : pow(a * a, p / 2) * (p &
      1 ? a : 1);
}
Mint inv(Mint a) {
  assert(a.v != 0);
  return pow(a, MOD - 2);
}
Mint operator/(Mint a, Mint b) { return a *
    inv(b); }
```

4 string

4.1 bin$_trie$

```
struct bin_trie {
    struct node {
           int nxt[2];
        int cnt;
        node() {
            cnt = 0;
                       nxt[0] = nxt[1] = -1;
        }
    };
        int maxbit = 40; // change this if ai
            <= 10^18
    int sz;
    vector<node> b;
    bin_trie() {
        b.assign(1, node());
        sz = 0;
    }
    void add(long long s) {
            int u = 0;
            for (int k = maxbit; k >= 0;
                k--) {
                   bool id = (1ll << k) & s;
                   if (b[u].nxt[id] == -1) {
                           b[u].nxt[id] =
                               ++sz;
                           b.push_back(node());
                   }
                   u = b[u].nxt[id];
                   b[u].cnt++;
            }
    }
};
```

4.2 hashing

```
struct hashing {
    //create MOD outside
    long long base = 311;
    int n;
    vector<long long> hs;
    vector<long long> invbase;

    hashing(string s) {
        n = s.size();
        hs.assign(n, 0); invbase.assign(n, 0);
        long long iv = inv(base);

        hs[0] = s[0]; invbase[0] = 1;
        long long p = 1;
        for (int i = 1; i < n; i++) {
            p = mul(p, base);
            hs[i] = add(hs[i - 1], mul(s[i],
                p));
            invbase[i] = mul(invbase[i - 1],
                iv);
        }
    }

    ll get(ll l, ll r) {
        ll res = hs[r];
        if (l) {
            res = add(res, -hs[l - 1]);
            res = mul(res, invbase[l]);
        }
        return res;
    }
};
```

4.3 prefix-function

```
vector<int> prefix_function (const string& s)
    {
  int n = (int) s.length();
  vector<int> prefix(n, 0);
  for (int i = 1; i < n; i++) {
    int j = prefix[i - 1];
    while (j > 0 && s[i] != s[j]) j = prefix[j
        - 1];
    if (s[i] == s[j]) j++;
    prefix[i] = j;
  }
  return prefix;
}
```

4.4 rolling$_{h}ash$

```
// rolling hash with one-based indexing
//
// usage Hash(s, BASE)
//
// query
//   hash.fwd(l, r) //get forward hash
//   hash.bwd(l, r) //get backward hash

struct Hash {
  int hash[N], hashR[N], pow[N];
  Hash(string &s, int BASE) {
    int n = s.size() - 1;
    hash[0] = 0;
    hashR[0] = 0;
    pow[0] = 1;
    for(int i = 1; i <= n; i++) {
      pow[i] = (pow[i - 1] * BASE) % MOD;
    }
    for(int i = 1; i <= n; i++) {
      hash[i] = (hash[i - 1] * BASE + s[i] -
          'a' + 1) % MOD;
    }
    for(int i = n; i >= 1; i--) {
      hashR[i] = (hashR[i + 1] * BASE + s[i] -
          'a' + 1) % MOD;
    }
  }
  int fwd(int i, int j) {
    return (hash[j] - hash[i - 1] * pow[j - i
        + 1] + MOD * MOD) % MOD;
  }
  int bwd(int i, int j) {
    return (hashR[i] - hashR[j + 1] * pow[j -
        i + 1] + MOD * MOD) % MOD;
  }
};
```

4.5 str$_{t}rie$

```
struct str_trie {
    struct node {
        vector<int> nxt;
        int end;

        node(int n) {
            end = 0;
            nxt.assign(n, -1);
        }
    };

    int sz;
    vector<node> b;

    str_trie(int maxb = 0) {
        b.assign(maxb + 5, node(26));
        sz = 0;
    }

    void add(string s) {
        int u = 0;
        for (char c : s) {
            if (b[u].nxt[c - 'a'] == -1)
                b[u].nxt[c - 'a'] = ++sz;
            u = b[u].nxt[c - 'a'];
        }
        b[u].end++;
    }
};
```