Team notebook

CU 413 - University of Chittagong

March 20, 2020

Contents

1 bigint

```
//Copied from lightoj

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

#define optimize()
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl '\n'

struct Bigint {
    // representations and structures
    string a; // to store the digits
    int sign; // sign = -1 for negative numbers, sign = 1
        otherwise
    // constructors
    Bigint() {} // default constructor
    Bigint( string b ) { (*this) = b; } // constructor for
        string
    // some helpful methods
    int size() { // returns number of digits
        return a.size();
    }
    Bigint inverseSign() { // changes the sign
        sign *= -1;
        return (*this);
    }
    Bigint normalize( int newSign ) { // removes leading 0,
        fixes sign
        for( int i = a.size() - 1; i > 0 && a[i] == '0'; i-- )
            a.erase(a.begin() + i);
        sign = ( a.size() == 1 && a[0] == '0' ) ? 1 : newSign;
        return (*this);
    }
    // assignment operator
    void operator = ( string b ) { // assigns a string to Bigint
        a = b[0] == '-' ? b.substr(1) : b;
        reverse( a.begin(), a.end() );
        this->normalize( b[0] == '-' ? -1 : 1 );
    }
    // conditional operators
    bool operator < ( const Bigint &b ) const { // less than
        operator
        if( sign != b.sign ) return sign < b.sign;
        if( a.size() != b.a.size() )
            return sign == 1 ? a.size() < b.a.size() : a.size()
                > b.a.size();
        for( int i = a.size() - 1; i >= 0; i-- ) if( a[i] !=
            b.a[i] )
            return sign == 1 ? a[i] < b.a[i] : a[i] > b.a[i];
        return false;
    }
    bool operator == ( const Bigint &b ) const { // operator
        for equality
        return a == b.a && sign == b.sign;
    }
    // mathematical operators
    Bigint operator + ( Bigint b ) { // addition operator
        overloading
        if( sign != b.sign ) return (*this) - b.inverseSign();
        Bigint c;
        for(int i = 0, carry = 0; i<a.size() || i<b.size() ||
            carry; i++ ) {
            carry+=(i<a.size() ? a[i]-48 : 0)+(i<b.a.size() ?
                b.a[i]-48 : 0);
            c.a += (carry % 10 + 48);
            carry /= 10;
        }
```

```
    return c.normalize(sign);
}
Bigint operator - ( Bigint b ) { // subtraction operator
     overloading
    if( sign != b.sign ) return (*this) + b.inverseSign();
    int s = sign; sign = b.sign = 1;
    if( (*this) < b ) return ((b -
        (*this)).inverseSign()).normalize(-s);
    Bigint c;
    for( int i = 0, borrow = 0; i < a.size(); i++ ) {
        borrow = a[i] - borrow - (i < b.size() ? b.a[i] :
             48);
        c.a += borrow >= 0 ? borrow + 48 : borrow + 58;
        borrow = borrow >= 0 ? 0 : 1;
    }
    return c.normalize(s);
}
Bigint operator * ( Bigint b ) { // multiplication operator
     overloading
    Bigint c("0");
    for( int i = 0, k = a[i] - 48; i < a.size(); i++, k =
        a[i] - 48 ) {
        while(k--) c = c + b; // ith digit is k, so, we add
             k times
        b.a.insert(b.a.begin(), '0'); // multiplied by 10
    }
    return c.normalize(sign * b.sign);
}
Bigint operator / ( Bigint b ) { // division operator
     overloading
    if( b.size() == 1 && b.a[0] == '0' ) b.a[0] /= ( b.a[0]
        - 48 );
    Bigint c("0"), d;
    for( int j = 0; j < a.size(); j++ ) d.a += "0";
    int dSign = sign * b.sign; b.sign = 1;
    for( int i = a.size() - 1; i >= 0; i-- ) {
        c.a.insert( c.a.begin(), '0');
        c = c + a.substr( i, 1 );
        while( !( c < b ) ) c = c - b, d.a[i]++;
    }
    return d.normalize(dSign);
}
Bigint operator % ( Bigint b ) { // modulo operator
     overloading
    if( b.size() == 1 && b.a[0] == '0' ) b.a[0] /= ( b.a[0]
        - 48 );
    Bigint c("0");
    b.sign = 1;
    for( int i = a.size() - 1; i >= 0; i-- ) {
        c.a.insert( c.a.begin(), '0');
        c = c + a.substr( i, 1 );
        while( !( c < b ) ) c = c - b;
    }
    return c.normalize(sign);
}
// output method
void print() {
    if( sign == -1 ) cout << '-';
    for( int i = a.size() - 1; i >= 0; i-- ) cout << a[i];
}
};

int main() {
        optimize();
    Bigint a, b, c; // declared some Bigint variables
    ////////////////////////
    // taking Bigint input //
    ////////////////////////
    string input; // string to take input
    cin >> input; // take the Big integer as string
    a = input; // assign the string to Bigint a
    cin >> input; // take the Big integer as string
    b = input; // assign the string to Bigint b
    //////////////////////////////////
    // Using mathematical operators //
    //////////////////////////////////
    c = a + b; // adding a and b
    c.print(); // printing the Bigint
    cout << endl; // newline
    c = a - b; // subtracting b from a
    c.print(); // printing the Bigint
    cout << endl; // newline
    c = a * b; // multiplying a and b
    c.print(); // printing the Bigint
    cout << endl; // newline
    c = a / b; // dividing a by b
    c.print(); // printing the Bigint
    cout << endl; // newline
    c = a % b; // a modulo b
    c.print(); // printing the Bigint
    cout << endl; // newline
    /////////////////////////////////
    // Using conditional operators //
    /////////////////////////////////
    if( a == b ) puts("equal"); // checking equality
    else puts("not equal");
    if( a < b ) puts("a is smaller than b"); // checking less
         than operator
    return 0;
}
```

2 Combinatorics

2.1 combinatorics

```
nCr = nC(r-1) * ((n-r+1)/r)
nCr = (n-1)Cr + (n-1)C(r-1)
```

3 Data Structures

3.1 Disjoint Set Union Find

```
///DSU: Path compression + Union by size
///It turns out, that the final amortized time complexity is
     O((n)), where (n) is the inverse Ackermann function,
     which grows very slowly. In fact it grows so slowly, that
     it doesn't exceed 4 for all reasonable n (approximately
     n<10^600).


int par[10005];
int sz[10005];

int Find(int r) ///Path compression
{
        if( par[r] == r ) return r;
        par[r] = Find(par[r]);
        return par[r];
}

void Union(int a, int b) {   ///Union by size of subtrees.
    a = Find(a);
    b = Find(b);
    if (a != b) {
        if (sz[a] < sz[b]) swap(a, b);
        par[b] = a;
        sz[a] += sz[b];
    }
}

int main()
{
    for( int i = 0; i < 10005; ++i ) {
        par[i] = i;
        sz[i] = 1;
    }
}
```

3.2 mos

```
///mos algorithm
///Complexity: O(sqrt(N) * (N+M))

const int mx = 3e5+5;

const int block_sz = 550;    // N ~ 3e5
int freq[mx], mo_cnt = 0;
int ret[mx];

inline void add(int idx) {
        ++freq[a[idx]];
        if(freq[a[idx]] == 1) ++mo_cnt;
}

inline void del(int idx) {
        --freq[a[idx]];
        if(freq[a[idx]] == 0) --mo_cnt;
}

inline int get_ans() {
```

```
        return mo_cnt;
}

struct queries {
        int l, r, idx;
        queries() { }
        queries(int _l, int _r, int _i) : l(_l), r(_r), idx(_i)
             { }
        bool operator < (const queries &p) const {
                if(l/block_sz != p.l/block_sz) return l < p.l;
                return ((l/block_sz) & 1) ? r > p.r : r < p.r;
        }
};

void mo(vector<queries> &q) {
        sort(q.begin(), q.end());
        memset(ret, -1, sizeof ret);

        // l = 1, r = 0 if 1-indexed array
        int l = 0, r = -1;
        for(auto &qq : q) {
                while(qq.l < l) add(--l);
                while(qq.r > r) add(++r);
                while(qq.l > l) del(l++);
                while(qq.r < r) del(r--);
                ret[qq.idx] = get_ans();
        }
}
```

3.3 sparse table

```
///sparse table
///0-based indexing
//https://cp-algorithms.com/data_structures/sparse-table.html


///1. range sum query (copied from cp-algorithms)
const int MAXN = 1e5+5;
long long arr[MAXN];
const int LOG = log2(MAXN) + 1;
long long sp[MAXN][LOG+1];
int N; ///size of arr

void init()   ///O(NlogN)
{
        for( int i = 0; i < N; ++i ) sp[i][0] = arr[i];
        for( int j = 1; j <= LOG; ++j )
                for( int i = 0; i + (1 << j) <= N; ++i )
                        sp[i][j] = sp[i][j-1] + sp[i + (1 << (j -
                              1))][j - 1];
}

long long query( int L, int R ) ///O(logN)
{
        long long sum = 0;
        for( int j = LOG; j >= 0; --j ) {
                if((1 << j) <= R - L + 1) {
                        sum += sp[L][j];
                        L += 1 << j;
                }
        }
        return sum;
}


///2. rmq (copied from cp-algorithms)
const int MAXN = 1e5+5;
long long arr[MAXN];
const int LOG = log2(MAXN) + 1;
long long sp[MAXN][LOG+1];
int prelog[MAXN];
int N; ///size of arr

void init_log() ///O(MAXN)
{
        prelog[1] = 0;
        for( int i = 2; i < MAXN; ++i ) prelog[i] = prelog[i/2]
             + 1;
}

void init()   ///O(NlogN)
{
        for( int i = 0; i < N; ++i ) sp[i][0] = arr[i];
        for( int j = 1; j <= LOG; ++j )
                for( int i = 0; i + (1 << j) <= N; ++i )
                        sp[i][j] = min(sp[i][j-1], sp[i + (1 <<
                              (j - 1))][j - 1]);
}

long long query(int L, int R) ///O(1)
{
        int j = prelog[R - L + 1];
        long long minimum = min(sp[L][j], sp[R - (1 << j) +
             1][j]);
        return minimum;
}


///3. 2D rmq
const int MAXN = 1005;
const int MAXM = 1005;
long long arr[MAXN][MAXM];
const int LOGN = log2(MAXN) + 1;
const int LOGM = log2(MAXM) + 1;
long long sp[MAXN][MAXM][LOGN + 1][LOGM + 1];
int prelog[max(MAXN, MAXM)];
int N, M;     ///size of arr is N*M

void init_log() ///O(MAXN)
{
        prelog[1] = 0;
        for( int i = 2; i < max(MAXN, MAXM); ++i ) prelog[i] =
             prelog[i/2] + 1;
}

void init()   ///O(N*M*logN*logM)
{
        for( int i = 0; i < N; ++i )
                for( int j = 0; j < M; ++j )
                        sp[i][j][0][0] = arr[i][j];

        for( int k = 1; k <= LOGN; ++k ) {
        for( int i = 0; i + (1 << k) <= N; ++i ) {
           for( int j = 0; j < M; ++j ) {
              sp[i][j][k][0] = min(sp[i][j][k - 1][0] , sp[i +
                     (1 << (k - 1))][j][k - 1][0]);
           }
        }
    }

    for( int l = 1; l <= LOGM; ++l ) {
        for( int k = 0; k <= LOGN; ++k ) {
           for( int i = 0; i + (1 << k) <= N; ++i ) {
              for( int j = 0; j + (1 << l) <= M; ++j ) {
                 sp[i][j][k][l] = min(sp[i][j][k][l - 1] ,
                        sp[i][j + (1 << (l - 1))][k][l - 1]);
              }
           }
        }
    }
}

long long query( int r1, int c1, int r2, int c2 ) ///O(1)
{
        int a = prelog[(r2 - r1) + 1];
    int b = prelog[(c2 - c1) + 1];
    return min(min(sp[r1][c1][a][b], sp[r2 - (1 << a) +
           1][c1][a][b]), min(sp[r1][c2 - (1 << b) + 1][a][b],
           sp[r2 - (1 << a) + 1][c2 - (1 << b) + 1][a][b]));
}
```

4 Flow

4.1 Dinic with and without scaling

```
/**
Dinic algorithm with scaling and not scaling.
Complexity with scaling : O(VE * lg(U)). here, U is the maximum
     capacity
Complexity without scaling : O((V^2)E)

//////
Complexity in unit graph: O(E*sqrt(V)).
A unit network is a network in which all the edges have unit
     capacity, and for any vertex except s and t either
     incoming or outgoing edge is unique. That's exactly the
     case with the network we build to solve the maximum
     matching problem with flows.
So, Dinic gives similar performance to hopcroft karp algorithm
     incase of maximum bipartite matching as it has the same
     time complexity.
//////
```

```
3rd fastest max flow implementation

1. Works on directed graph
2. Works on undirected graph
3. Works on multi-edge(directed/undirected) graph
4. Works on self-loop(directed/undirected) graph

Can find the actual flow.
Can find the non-maxflow upto a certain value

//////
Implement it on your own. very easy. just a simple O(n) dfs.

Can find minimum cut sets(A and B).
A contains source itself and the nodes that are reachable from
     source using non-saturated edges.
B contains sink and the nodes that are not reachable from
     source using non-saturated edges.
Value of minimum cut capacity is summation of the per_cap of
     all edges u->v such that u belongs to set A, v belongs to
     set B.
Value of minimum cut flow is = (summation of the flow of all
     edges u->v such that u belongs to set A, v belongs to set
     B) - (summation of the flow of all edges v->u such that v
     belongs to set B, u belongs to set A)
Value of minimum cut capacity == maxflow.

To find the minimum cut sets(A and B) first find the maxflow.
     Then, we apply dfs from the source. This dfs will be such
     that- only
the nodes of set A will eventually be marked visited. Suppose
     we have reached in node u, that means u belongs to set A.
     Now,
If (flow of edge u->v) < (per_cap of edge u->v) then v belongs
     to set A.
else if(flow of edge u->v) == (per_cap of edge u->v) then v
     belongs to set B.

To find the minimum cut sets with minimum cardinality, first
     multiply (E+1) with all the edges. Then add 1 to all the
     edges. Then run the maxflow algorithm.
//////

Status: Tested and OK
**/


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

#define optimize()
     ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl '\n'


template <class flow_t> struct Dinic { ///int/long long;
   const static bool SCALING = true; /// non-scaling = V^2E,
        Scaling=VElog(U) with higher constant
   long long lim = 1;
   const flow_t INF = numeric_limits<flow_t>::max();
   flow_t K;
   bool K2 = false;
   struct edge {
      int to, rev;
      flow_t cap, flow, per_cap;
   };
   int s, t;
   vector<int> level, ptr;
   vector< vector<edge> > adj;
   Dinic(int NN) : s(NN-2), t(NN-1), level(NN), ptr(NN),
        adj(NN) {}

   void flow_limit( flow_t val ) {          ///non-maxflow upto
        K.
      K2 = true;
      K = val;
   }

   void addEdge(int a, int b, flow_t cap, bool isDirected =
        true) {
      adj[a].push_back({b, (int)adj[b].size(), cap, 0, cap});
      adj[b].push_back({a, (int)adj[a].size() - 1, isDirected
           ? 0 : cap, 0, isDirected ? 0 : cap});
   }
   bool bfs() {
      queue<int> q({s});
      fill( level.begin(), level.end(), -1 );
      level[s] = 0;
      while (!q.empty() && level[t] == -1) {
         int v = q.front();
         q.pop();
         for (auto e : adj[v]) {
            if (level[e.to] == -1 && e.flow < e.cap &&
                 (!SCALING || e.cap - e.flow >= lim)) {
               q.push(e.to);
               level[e.to] = level[v] + 1;
            }
         }
      }
      return level[t] != -1;
   }
   flow_t dfs(int v, flow_t flow) {
      if (v == t || !flow)
         return flow;
      for (; ptr[v] < adj[v].size(); ptr[v]++) {
         edge &e = adj[v][ptr[v]];
         if (level[e.to] != level[v] + 1)
            continue;
         if (flow_t pushed = dfs(e.to, min(flow, e.cap -
              e.flow))) {
            e.flow += pushed;
            adj[e.to][e.rev].flow -= pushed;
            return pushed;
         }
      }
      return 0;
   }
   flow_t max_flow(int source, int sink) {
      s = source, t = sink;
      long long flow = 0;
      for (lim = SCALING ? (1LL << 30) : 1; lim > 0; lim >>=
           1) { ///Here, lim = SCALING?(U):1 ; Here U is an
           int/long long strictly greater than the max
           capacity ;
         while (bfs()) {
            fill( ptr.begin(), ptr.end(), 0 );
            while (flow_t pushed = dfs(s,
                 ((K2==true)?K:INF))) {
                              flow += pushed;
                              if(K2) {
                                     K -= pushed;
                                     if( K == 0 ) break;
                              }
            }
            if( K2 && (K == 0) ) break;
         }
      }
      return flow;
   }
   vector<pair<pair<int,int>,long long>> getActualFlow()
   {
      vector<pair<pair<int,int>, long long>> vec;
      for( int i = 0; i < adj.size(); ++i ) {
                for( int j = 0; j < adj[i].size(); ++j ) {
                       if( adj[i][j].flow > 0 ) {
                              vec.push_back(
                                   make_pair(make_pair(i,
                                   adj[i][j].to),
                                   adj[i][j].flow) );
                       }
                }
      }
      return vec;
   }
};


int main()
{
      optimize();
      int T;
      cin >> T;
      for( int test = 1; test <= T; ++test ) {
             int N, M, s, t; /// no. of nodes; no. of edges;
                  source; sink;
             cin >> N >> M >> s >> t;
             Dinic<int> fl(N+1);   ///for long long change int
                  to long long; no. of nodes+1;
             for( int i = 1; i <= M; ++i ) {
                    int u, v, w;
                    cin >> u >> v >> w;
                    fl.addEdge(u, v, w);  ///Directed graph;
                    ///fl.addEdge(u, v, w, false);
                         ///Undirected graph;
             }
             ///fl.flow_limit(10); ///non-maxflow upto a
                  specific value;
             cout << fl.max_flow(s, t) << endl;
             vector<pair<pair<int,int>,long long>> vec =
                  fl.getActualFlow();
```

```
            cout << vec.size() << endl;
            for( auto xx : vec ) {
                cout << xx.first.first << " " <<
                    xx.first.second << " " << xx.second
                    << endl; ///node; node; flow;
            }
        }
        return 0;
}
```

4.2 Highest label preflow push

```
/**
Highest Label Preflow Push
Complexity : O(V^2 * sqrt(E))

Fastest max flow implementation

1. Works on directed graph
2. Works on undirected graph
3. Works on multi-edge(directed/undirected) graph
4. Works on self-loop(directed/undirected) graph

Can't find the actual flow.
Can't find the minimum cut sets.

Status: Tested and OK
**/


#include<bits/stdc++.h>
#define optimize()
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl '\n'
using namespace std;

template <class flow_t> ///int/long long;
struct HighestLabelPreflowPush {
    struct Edge {
        int v, rev;
        flow_t cap, tot;
        Edge(int a, flow_t b, int c) : v(a), rev(c), cap(b),
            tot(b) {}
    };

    const flow_t maxf = numeric_limits<flow_t>::max();
    int ht, S, T, N, H, labelcnt;

    vector<flow_t> exflow;
    vector< vector<Edge> > G;
    vector< vector<int> > hq, gap;
    vector<int> h, cnt;

    HighestLabelPreflowPush(int NN) : exflow(NN), G(NN),
        hq(NN), gap(NN) {}

    void addEdge(int u, int v, flow_t cap) {
        G[u].emplace_back(v, cap, G[v].size());
        G[v].emplace_back(u, 0, G[u].size() - 1);
    }

    void update(int u, int newh) {
        ++labelcnt;
        if (h[u] != H)
            --cnt[h[u]];
        h[u] = newh;
        if (newh == H)
            return;
        ++cnt[ht = newh];
        gap[newh].push_back(u);
        if (exflow[u] > 0)
            hq[newh].push_back(u);
    }

    void globalRelabel() {
        queue<int> q;
        for (int i = 0; i <= H; i++) hq[i].clear(),
            gap[i].clear();
        h.assign(H, H);
        cnt.assign(H, 0);
        q.push(T);
        labelcnt = ht = h[T] = 0;
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (Edge& e : G[u]) {
                if (h[e.v] == H && G[e.v][e.rev].cap) {
                    update(e.v, h[u] + 1);
                    q.push(e.v);
                }
            }
            ht = h[u];
        }
    }

    void push(int u, Edge& e) {
        if (exflow[e.v] == 0)
            hq[h[e.v]].push_back(e.v);
        flow_t df = min(exflow[u], e.cap);
        e.cap -= df;
        G[e.v][e.rev].cap += df;
        exflow[u] -= df;
        exflow[e.v] += df;
    }

    void discharge(int u) {
        int nxth = H;
        if (h[u] == H)
            return;
        for (Edge& e : G[u])
            if (e.cap) {
                if (h[u] == h[e.v] + 1) {
                    push(u, e);
                    if (exflow[u] <= 0)
                        return;
                } else if (nxth > h[e.v] + 1)
                    nxth = h[e.v] + 1;
            }
        if (cnt[h[u]] > 1)
            update(u, nxth);
        else
            for (; ht >= h[u]; gap[ht--].clear()) {
                for (int& j : gap[ht]) update(j, H);
            }
    }

    flow_t maxFlow(int s, int t, int n) {
        S = s, T = t, N = n, H = N + 1;
        fill( exflow.begin(), exflow.end(), 0 );
        exflow[S] = maxf;
        exflow[T] = -maxf;
        globalRelabel();
        for (Edge& e : G[S]) push(S, e);
        for (; ~ht; --ht) {
            while (!hq[ht].empty()) {
                int u = hq[ht].back();
                hq[ht].pop_back();
                discharge(u);
                if (labelcnt > (N << 2))
                    globalRelabel();
            }
        }
        return exflow[T] + maxf;
    }
};


int main() {
    optimize();
    int T;
    cin >> T;
    for( int test = 1; test <= T; ++test ) {
            int N, M, s, t; ///no. of nodes; no. of edges;
                source; sink;
        cin >> N >> M >> s >> t;
            HighestLabelPreflowPush<int> hlpp(N+2); ///int
                to long long for flow of long long; total
                no. of nodes+2(nodes+1 does not work);
            for( int i = 1; i <= M; ++i ) {
                int u, v, w;
                cin >> u >> v >> w;
                hlpp.addEdge(u, v, w); ///For directed
                    graph

                /**
                    For undirected graph:
                    hlpp.addEdge(u, v, w);
                    hlpp.addEdge(v, u, w);
                **/
            }
            cout << hlpp.maxFlow(s, t, N) << endl;
                ///source; sink; number of nodes;
    }
    return 0;
}
```

4.3 Hopcroft karp

```
//Hopcroft Karp
//Complexity: O(sqrt(V) * E), constant may be a bit high.
//Works on self loops.

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

struct Hopcroft_karp{

	int n;
	vector< vector<int> > edge;
	vector<int> dis, parent, L, R;
	vector<int> Q;

	Hopcroft_karp(int n_) : n(n_), edge(n_+1), dis(n_+1),
	     parent(n_+1), L(n_+1), R(n_+1), Q(n_+1) {};

	void add_edge( int u, int v )
	{
		edge[u].push_back(v);
	}

	bool dfs(int i)
	{
		int len = edge[i].size();
		for (int j = 0; j < len; j++) {
			int x = edge[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()
	{
		int x, f = 0, l = 0;
		fill( dis.begin(), dis.end(), -1 );
		for (int i = 1; i <= n; i++) {
			if (R[i] == -1) {
				Q[l++] = i;
				dis[i] = 0;
			}
		}
		while (f < l) {
			int i = Q[f++];
			int len = edge[i].size();
			for (int j = 0; j < len; j++) {
				x = edge[i][j];
				if (L[x] == -1) return true;
				else if (dis[L[x]] == -1) {
					parent[L[x]] = i;
					dis[L[x]] = dis[i] + 1;
					Q[l++] = L[x];
				}
			}
		}
		return false;
	}

	int matching()
	{
		int counter = 0;	///How many nodes are part
		     of the maximum matching?
		fill( L.begin(), L.end(), -1 );
		fill( R.begin(), R.end(), -1 );
		while (bfs()) {
			for (int i = 1; i <= n; i++) {
				if (R[i] == -1 && dfs(i))
				       counter++;
			}
		}
		return counter;
	}
};


int main()
{
	int n, m;	///no. of nodes; no. of edges;
	cin >> n >> m;
	Hopcroft_karp h(n);
	for( int i = 0; i < m; ++i ) {
		int u, v;
		cin >> u >> v;

		///Undirected:
		h.add_edge(u, v);
		h.add_edge(v, u);

		/**
		///Directed:
		h.add_edge(u, v);
		**/

	}

	int ans = h.matching(); ///How many nodes are part of
	     the maximum matching?
	cout << ans << endl;

	///print the actual maximum matching
	for( int i = 1; i <= n; ++i ) {
		if( h.L[i] != -1 ) cout << i << " " << h.L[i] <<
		     endl;
	}
}
```

4.4 minCostflow$_{b}y_{d}acin21_{1}$

```
/**
// Push-Relabel implementation of the cost-scaling algorithm
// Runs in O( <max_flow> * log(V * max_edge_cost)) = O( V^3 *
     log(V * C))
// Operates on integers
// Works on regular directed graphs.

// Can't operate on doubles. For this, use bellman
     ford/djikstra method.
//Whether or not it works on undirected or multi-edge or
     self-loop graphs is yet to be verified.

// To get the actual flow collect all the edges which have (f >
     0) - this works just like
   the actual flow finding in normal max flow algorithms;

// Can't find nonMaxflow value upto K. The code is too complex
     for me to add this function.
   Maybe its not possible at all and you can only do it with
        the bellmanFord/Djikstra method.

**/



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

#define optimize()
     ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl '\n'

template<typename flow_t = int, typename cost_t = int>
     ///int/long long; int/long long;
struct mcSFlow{
    struct Edge{
        cost_t c;
        flow_t f;
        int to, rev;
        Edge(int _to, cost_t _c, flow_t _f, int _rev):c(_c),
             f(_f), to(_to), rev(_rev){}
    };
    const cost_t INFCOST = numeric_limits<cost_t>::max()/2;
         ///divide by slightly bigger number if overflow;
    const cost_t INFFLOW = numeric_limits<flow_t>::max()/2;
         ///divide by slightly bigger number if overflow;
    cost_t epsilon;
    int N, S, T;
    vector<vector<Edge> > G;
    vector<unsigned int> isEnqueued, state;
    mcSFlow(int _N):epsilon(0), N(_N), G(_N){}
    void add_edge(int a, int b, cost_t cost, flow_t cap){
        if(a==b){assert(cost>=0); return;}
        cost*=N;// to preserve integer-values
        epsilon = max(epsilon, abs(cost));
        assert(a>=0&&a<N&&b>=0&&b<N);
        G[a].emplace_back(b, cost, cap, G[b].size());
        G[b].emplace_back(a, -cost, 0, G[a].size()-1);
    }
    flow_t calc_max_flow(){ // Dinic max-flow
```

```
    vector<flow_t> dist(N), state(N);
    vector<Edge*> path(N);
    auto cmp = [](Edge*a, Edge*b){return a->f < b->f;};
    flow_t addFlow, retflow=0;;
    do{
        fill(dist.begin(), dist.end(), -1);
        dist[S]=0;
        auto head = state.begin(), tail = state.begin();
        for(*tail++ = S;head!=tail;++head){
            for(Edge const&e:G[*head]){
                if(e.f && dist[e.to]==-1){
                    dist[e.to] = dist[*head]+1;
                    *tail++=e.to;
                }
            }
        }
        addFlow = 0;
        fill(state.begin(), state.end(), 0);
        auto top = path.begin();
        Edge dummy(S, 0, INFFLOW, -1);
        *top++ = &dummy;
        while(top != path.begin()){
            int n = (*prev(top))->to;
            if(n==T){
                auto next_top = min_element(path.begin(),
                     top, cmp);
                flow_t flow = (*next_top)->f;
                while(--top!=path.begin()){
                    Edge &e=**top, &f=G[e.to][e.rev];
                    e.f-=flow;
                    f.f+=flow;
                }
                addFlow=1;
                retflow+=flow;
                top = next_top;
                continue;
            }
            for(int &i=state[n], i_max = G[n].size(), need =
                 dist[n]+1;;++i){
                if(i==i_max){
                    dist[n]=-1;
                    --top;
                    break;
                }
                if(dist[G[n][i].to] == need && G[n][i].f){
                    *top++ = &G[n][i];
                    break;
                }
            }
        }
    }while(addFlow);
    return retflow;
}
vector<flow_t> excess;
vector<cost_t> h;
void push(Edge &e, flow_t amt){
    //cerr << "push: " << G[e.to][e.rev].to << " -> " <<
         e.to << " (" << e.f << "/" << e.c << ") : " << amt
         << "\n";
    if(e.f < amt) amt=e.f;
    e.f-=amt;
    excess[e.to]+=amt;
    G[e.to][e.rev].f+=amt;
    excess[G[e.to][e.rev].to]-=amt;
}
void relabel(int vertex){
    cost_t newHeight = -INFCOST;
    for(unsigned int i=0;i<G[vertex].size();++i){
        Edge const&e = G[vertex][i];
        if(e.f && newHeight < h[e.to]-e.c){
            newHeight = h[e.to] - e.c;
            state[vertex] = i;
        }
    }
    h[vertex] = newHeight - epsilon;
}
const int scale=2;
pair<flow_t, cost_t> minCostFlow(int _S, int _T){
    S = _S, T = _T;
    cost_t retCost = 0;
    for(int i=0;i<N;++i){
        for(Edge &e:G[i]){
            retCost += e.c*(e.f);
        }
    }
    //find feasible flow
    flow_t retFlow = calc_max_flow();
    excess.resize(N);h.resize(N);
    queue<int> q;
    isEnqueued.assign(N, 0); state.assign(N,0);
    for(;epsilon;epsilon>>=scale){
        //refine
        fill(state.begin(), state.end(), 0);
        for(int i=0;i<N;++i)
            for(auto &e:G[i])
                if(h[i] + e.c - h[e.to] < 0 && e.f) push(e,
                     e.f);
        for(int i=0;i<N;++i){
            if(excess[i]>0){
                q.push(i);
                isEnqueued[i]=1;
            }
        }
        while(!q.empty()){
            int cur=q.front();q.pop();
            isEnqueued[cur]=0;
            // discharge
            while(excess[cur]>0){
                if(state[cur] == G[cur].size()){
                    relabel(cur);
                }
                for(unsigned int &i=state[cur], max_i =
                     G[cur].size();i<max_i;++i){
                    Edge &e=G[cur][i];
                    if(h[cur] + e.c - h[e.to] < 0){
                        push(e, excess[cur]);
                        if(excess[e.to]>0 &&
                             isEnqueued[e.to]==0){
                            q.push(e.to);
                            isEnqueued[e.to]=1;
                        }
                        if(excess[cur]==0) break;
                    }
                }
            }
        }
        if(epsilon>1 && epsilon>>scale==0){
            epsilon = 1<<scale;
        }
    }
    for(int i=0;i<N;++i){
        for(Edge &e:G[i]){
            retCost -= e.c*(e.f);
        }
    }
    //cerr << " -> " << retFlow << " / " << retCost << "
         bzw. " << retCost/2/N << "\n";
    return make_pair(retFlow, retCost/2/N);
}
flow_t getFlow(Edge const &e){
    return G[e.to][e.rev].f;
}
};


int main()
{
        optimize();
        int T;
        cin >> T;
        for( int test = 1; test <= T; ++test ) {
                int N, M, S, T; /// Number of nodes; number of
                     edges; source; sink;
                cin >> N >> M >> S >> T;
                mcSFlow<> fl(N+1); ///Add long long, long long
                     if necessary;
                for( int i = 1; i <= M; ++i ) {
                        int u, v, c, w; /// node; node; cost;
                             capacity;
                        cin >> u >> v >> c >> w;
                        fl.add_edge(u, v, c, w);     ///Directed
                             graph;
                }
                cout << fl.minCostFlow(S, T).first << " " <<
                     fl.minCostFlow(S, T).second; ///flow; cost;
        }
}
```

4.5 minCostflow$_{b}y_{d}acin21_{2}$

```
/**
// Push-Relabel implementation of the cost-scaling algorithm
// Runs in O( <max_flow> * log(V * max_edge_cost)) = O( V^3 *
     log(V * C))
// Really fast in practice, 3e4 edges are fine.
// Operates on integers, costs are multiplied by N!!
// Works on regular directed graphs.
```

```
// Can't operate on doubles. For this, use bellman
     ford/djikstra method.
//Whether or not it works on undirected or multi-edge or
     self-loop graphs is yet to be verified.

// To get the actual flow collect all the edges which have (f >
     0) - this works just like
   the actual flow finding in normal max flow algorithms;

// Can't find nonMaxflow value upto K. The code is too complex
     for me to add this function.
   Maybe its not possible at all and you can only do it with
        the bellmanFord/Djikstra method.

**/



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

#define optimize()
     ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl '\n'

template<typename flow_t = int, typename cost_t = int>
     ///int/long long; int/long long;
struct mcSFlow{
   struct Edge{
      cost_t c;
      flow_t f;
      int to, rev;
      Edge(int _to, cost_t _c, flow_t _f, int _rev):c(_c),
           f(_f), to(_to), rev(_rev){}
   };
   static constexpr cost_t INFCOST =
        numeric_limits<cost_t>::max()/2; ///divide by slightly
        bigger number if overflow;
   cost_t eps;
   int N, S, T;
   vector<vector<Edge> > G;
   vector<unsigned int> isq, cur;
   vector<flow_t> ex;
   vector<cost_t> h;
   mcSFlow(int _N):eps(0), N(_N), G(_N){}
   void add_edge(int a, int b, cost_t cost, flow_t cap){
            assert(cap>=0);
      assert(a>=0&&a<N&&b>=0&&b<N);
      if(a==b){assert(cost>=0); return;}
      cost*=N;
      eps = max(eps, abs(cost));
      G[a].emplace_back(b, cost, cap, G[b].size());
      G[b].emplace_back(a, -cost, 0, G[a].size()-1);
   }
   void add_flow(Edge& e, flow_t f) {
      Edge &back = G[e.to][e.rev];
      if (!ex[e.to] && f)
         hs[h[e.to]].push_back(e.to);
      e.f -= f; ex[e.to] += f;
      back.f += f; ex[back.to] -= f;
   }
   vector<vector<int> > hs;
   vector<int> co;
   flow_t max_flow() {
      ex.assign(N, 0);
      h.assign(N, 0); hs.resize(2*N);
      co.assign(2*N, 0); cur.assign(N, 0);
      h[S] = N;
      ex[T] = 1;
      co[0] = N-1;
      for(auto &e:G[S]) add_flow(e, e.f);
      if(hs[0].size())
      for (int hi = 0;hi>=0;) {
         int u = hs[hi].back();
         hs[hi].pop_back();
         while (ex[u] > 0) { // discharge u
            if (cur[u] == G[u].size()) {
               h[u] = 1e9;
               for(unsigned int i=0;i<G[u].size();++i){
                  auto &e = G[u][i];
                  if (e.f && h[u] > h[e.to]+1){
                     h[u] = h[e.to]+1, cur[u] = i;
                  }
               }
               if (++co[h[u]], !--co[hi] && hi < N)
                  for(int i=0;i<N;++i)
                     if (hi < h[i] && h[i] < N){
                        --co[h[i]];
                        h[i] = N + 1;
                     }
               hi = h[u];
            } else if (G[u][cur[u]].f && h[u] ==
                 h[G[u][cur[u]].to]+1)
               add_flow(G[u][cur[u]], min(ex[u],
                    G[u][cur[u]].f));
            else ++cur[u];
                  }
         while (hi>=0 && hs[hi].empty()) --hi;
      }
      return -ex[S];
   }
   void push(Edge &e, flow_t amt){
      if(e.f < amt) amt=e.f;
      e.f-=amt; ex[e.to]+=amt;
      G[e.to][e.rev].f+=amt; ex[G[e.to][e.rev].to]-=amt;
   }
   void relabel(int vertex){
      cost_t newHeight = -INFCOST;
      for(unsigned int i=0;i<G[vertex].size();++i){
         Edge const&e = G[vertex][i];
         if(e.f && newHeight < h[e.to]-e.c){
            newHeight = h[e.to] - e.c;
            cur[vertex] = i;
         }
      }
      h[vertex] = newHeight - eps;
   }
   static constexpr int scale=2;
   pair<flow_t, cost_t> minCostMaxFlow(int _S, int _T){
      S = _S, T = _T;
      cost_t retCost = 0;
      for(int i=0;i<N;++i)
         for(Edge &e:G[i])
            retCost += e.c*(e.f);
      //find max-flow
      flow_t retFlow = max_flow();
      h.assign(N, 0); ex.assign(N, 0);
      isq.assign(N, 0); cur.assign(N,0);
      queue<int> q;
      for(;eps;eps>>=scale){
         //refine
         fill(cur.begin(), cur.end(), 0);
         for(int i=0;i<N;++i)
            for(auto &e:G[i])
               if(h[i] + e.c - h[e.to] < 0 && e.f) push(e,
                    e.f);
         for(int i=0;i<N;++i){
            if(ex[i]>0){
               q.push(i);
               isq[i]=1;
            }
         }
         // make flow feasible
         while(!q.empty()){
            int u=q.front();q.pop();
            isq[u]=0;
            while(ex[u]>0){
               if(cur[u] == G[u].size())
                  relabel(u);
               for(unsigned int &i=cur[u], max_i =
                    G[u].size();i<max_i;++i){
                  Edge &e=G[u][i];
                  if(h[u] + e.c - h[e.to] < 0){
                     push(e, ex[u]);
                     if(ex[e.to]>0 && isq[e.to]==0){
                        q.push(e.to);
                        isq[e.to]=1;
                     }
                     if(ex[u]==0) break;
                  }
               }
            }
         }
         if(eps>1 && eps>>scale==0){
            eps = 1<<scale;
         }
      }
      for(int i=0;i<N;++i){
         for(Edge &e:G[i]){
            retCost -= e.c*(e.f);
         }
      }
      return make_pair(retFlow, retCost/2/N);
   }
   flow_t getFlow(Edge const &e){
      return G[e.to][e.rev].f;
   }
};
```

```
int main()
{
	optimize();
	int T;
	cin >> T;
	for( int test = 1; test <= T; ++test ) {
		int N, M, S, T; /// Number of nodes; number of
		    edges; source; sink;
		cin >> N >> M >> S >> T;
		mcSFlow<> fl(N+1); ///Add long long, long long
		    if necessary;
		for( int i = 1; i <= M; ++i ) {
			int u, v, c, w; /// node; node; cost;
			    capacity;
			cin >> u >> v >> c >> w;
			fl.add_edge(u, v, c, w);	///Directed
			    graph;
		}
		cout << fl.minCostMaxFlow(S, T).first << " " <<
		    fl.minCostMaxFlow(S, T).second; ///flow;
		    cost;
	}
}
```

4.6 minCostflow$_by_dacin21_3$

```
/**
// Push-Relabel implementation of the cost-scaling algorithm
// Runs in O( <max_flow> * log(V * max_edge_cost)) = O( V^2 E *
    log(V * C))
// Really fast in practice, like O(V * E), so 3e4 edges are
    fine.
// Operates on integers, costs are multiplied by N!!
// Works on regular directed graphs.

// Can't operate on doubles. For this, use bellman
    ford/djikstra method.
//Whether or not it works on undirected or multi-edge or
    self-loop graphs is yet to be verified.

// To get the actual flow collect all the edges which have (f >
    0) - this works just like
  the actual flow finding in normal max flow algorithms;

// Can't find nonMaxflow value upto K. The code is too complex
    for me to add this function.
  Maybe its not possible at all and you can only do it with
      the bellmanFord/Djkstra method.

**/


#include<bits/stdc++.h>
using namespace std;
#define optimize()
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl '\n'

template<typename flow_t = long long, typename cost_t = long
    long> ///int/long long; int/long long;
struct mcSFlow{
    struct Edge{
        cost_t c;
        flow_t f;
        int to, rev;
        Edge(int _to, cost_t _c, flow_t _f, int _rev):c(_c),
            f(_f), to(_to), rev(_rev){}
    };
    static constexpr cost_t INFCOST =
        numeric_limits<cost_t>::max()/2; ///divide by slightly
        bigger number if overflow;
    cost_t eps;
    int N, S, T;
    vector<vector<Edge> > G;
    vector<unsigned int> isq, cur;
    vector<flow_t> ex;
    vector<cost_t> h;
    mcSFlow(int _N):eps(0), N(_N), G(_N){}
    Edge add_edge(int a, int b, cost_t cost, flow_t cap){
        assert(cap>=0);
        assert(a>=0&&a<N&&b>=0&&b<N);
        assert(a!=b);
        cost*=N;
        eps = max(eps, abs(cost));
        G[a].emplace_back(b, cost, cap, G[b].size());
        Edge ret = G[a].back();
        G[b].emplace_back(a, -cost, 0, G[a].size()-1);
        return ret;
    }
    void add_flow(Edge& e, flow_t f) {
        Edge &back = G[e.to][e.rev];
        if (!ex[e.to] && f)
        hs[h[e.to]].push_back(e.to);
        e.f -= f; ex[e.to] += f;
        back.f += f; ex[back.to] -= f;
    }
    vector<vector<int> > hs;
    vector<int> co;
    // fast max flow, lowest label version
    flow_t max_flow() {
        ex.assign(N, 0);
        h.assign(N, 0); hs.resize(2*N);
        co.assign(2*N, 0); cur.assign(N, 0);
        h[S] = N;
        ex[T] = 1;
        co[0] = N-1;
        for(auto &e:G[S]) add_flow(e, e.f);
        if(hs[0].size())
        for (int hi = 0;hi>=0;) {
            int u = hs[hi].back();
            hs[hi].pop_back();
            while (ex[u] > 0) { // discharge u
                if (cur[u] == G[u].size()) {
                    h[u] = 1e9;
                    for(unsigned int i=0;i<G[u].size();++i){
                        auto &e = G[u][i];
                        if (e.f && h[u] > h[e.to]+1){
                            h[u] = h[e.to]+1;
                            cur[u] = i;
                        }
                    }
                    if (++co[h[u]], !--co[hi] && hi < N)
                    for(int i=0;i<N;++i){
                        if (hi < h[i] && h[i] < N){
                            --co[h[i]];
                            h[i] = N + 1;
                        }
                    }
                    hi = h[u];
                } else if (G[u][cur[u]].f && h[u] ==
                    h[G[u][cur[u]].to]+1){
                    add_flow(G[u][cur[u]], min(ex[u],
                        G[u][cur[u]].f));
                } else ++cur[u];
            }
            while (hi>=0 && hs[hi].empty()) --hi;
        }
        return -ex[S];
    }
    // begin min cost flow
    bool look_ahead(int u){
        if(ex[u]) return false;
        cost_t newHeight = h[u]-N*eps;
        for(auto const&e:G[u]){
            if(e.f == 0) continue;
            if(h[u] + e.c - h[e.to] < 0) return false; //
                outgoing admissible arc
            else newHeight = max(newHeight, h[e.to] - e.c); //
                try to make arc admissible
        }
        h[u] = newHeight - eps;
        return true;
    }
    void push(Edge &e, flow_t amt){
        if(e.f < amt) amt=e.f;
        e.f-=amt; ex[e.to]+=amt;
        G[e.to][e.rev].f+=amt; ex[G[e.to][e.rev].to]-=amt;
    }
    void relabel(int vertex){
        cost_t newHeight = -INFCOST;
        for(unsigned int i=0;i<G[vertex].size();++i){
            Edge const&e = G[vertex][i];
            if(e.f && newHeight < h[e.to]-e.c){
                newHeight = h[e.to] - e.c;
                cur[vertex] = i;
            }
        }
        h[vertex] = newHeight - eps;
    }
    static constexpr int scale=2;
    template<bool use_look_ahead = true>
    pair<flow_t, cost_t> minCostMaxFlow(int _S, int _T){
        S = _S, T = _T;
        cost_t retCost = 0;
```

```
    for(int i=0;i<N;++i)
    for(Edge &e:G[i])
    retCost += e.c*(e.f);
    // remove this for circulation
    flow_t retFlow = max_flow();
    h.assign(N, 0); ex.assign(N, 0);
    isq.assign(N, 0); cur.assign(N,0);
    stack<int> q;
    for(;eps;eps>>=scale){
        fill(cur.begin(), cur.end(), 0);
        for(int i=0;i<N;++i)
        for(auto &e:G[i])
            if(h[i] + e.c - h[e.to] < 0 && e.f)
                push(e, e.f);
        for(int i=0;i<N;++i){
            if(ex[i]>0){
                q.push(i);
                isq[i]=1;
            }
        }
        while(!q.empty()){
            int u=q.top();q.pop();
            isq[u]=0;
            while(ex[u]>0){
                if(cur[u] == G[u].size())
                    relabel(u);
                for(unsigned int &i=cur[u], max_i =
                     G[u].size();i<max_i;++i){
                    Edge &e=G[u][i];
                    if(e.f == 0) continue;
                    if(h[u] + e.c - h[e.to] < 0){
                        if(use_look_ahead &&
                             look_ahead(e.to)){
                            --i;
                            continue;
                        }
                        push(e, ex[u]);
                        if(isq[e.to]==0){
                            q.push(e.to);
                            isq[e.to]=1;
                        }
                        if(ex[u]==0) break;
                    }
                }
            }
        }
        if(eps>1 && eps>>scale==0){
            eps = 1<<scale;
        }
    }
    for(int i=0;i<N;++i){
        for(Edge &e:G[i]){
            retCost -= e.c*(e.f);
        }
    }
    return make_pair(retFlow, retCost/2/N);
}
flow_t getFlow(Edge const &e){
    return G[e.to][e.rev].f;
}
};

int main()
{
        optimize();
        int T;
        cin >> T;
        for( int test = 1; test <= T; ++test ) {
                int N, M, S, T; /// Number of nodes; number of
                      edges; source; sink;
                cin >> N >> M >> S >> T;
                mcSFlow<> fl(N+1); ///Add long long, long long
                      if necessary;
                for( int i = 1; i <= M; ++i ) {
                        int u, v, c, w; /// node; node; cost;
                              capacity;
                        cin >> u >> v >> c >> w;
                        fl.add_edge(u, v, c, w);     ///Directed
                              graph;
                }
                cout << fl.minCostMaxFlow(S, T).first << " " <<
                      fl.minCostMaxFlow(S, T).second; ///flow;
                      cost;
        }
}
```

4.7 Push relabel with gap heuristic and highest labeling

```
/**
Push relabel with gap heuristic and highest labeling
Complexity : O(V^2 * sqrt(E))

2nd fastest max flow implementation

1. Works on directed graph
2. Works on undirected graph
3. Works on multi-edge(directed/undirected) graph
4. Works on self-loop(directed/undirected) graph

Can find the actual flow.

//////
Implement it on your own. very easy. just a simple O(n) dfs.

Can find minimum cut sets(A and B).
A contains source itself and the nodes that are reachable from
      source using non-saturated edges.
B contains sink and the nodes that are not reachable from
      source using non-saturated edges.
Value of minimum cut capacity is summation of the per_cap of
      all edges u->v such that u belongs to set A, v belongs to
      set B.
Value of minimum cut flow is = (summation of the flow of all
      edges u->v such that u belongs to set A, v belongs to set
      B) - (summation of the flow of all edges v->u such that v
      belongs to set B, u belongs to set A)
Value of minimum cut capacity == maxflow.

To find the minimum cut sets(A and B) first find the maxflow.
      Then, we apply dfs from the source. This dfs will be such
      that- only
the nodes of set A will eventually be marked visited. Suppose
      we have reached in node u, that means u belongs to set A.
      Now,
If (flow of edge u->v) < (per_cap of edge u->v) then v belongs
      to set A.
else if(flow of edge u->v) == (per_cap of edge u->v) then v
      belongs to set B.

To find the minimum cut sets with minimum cardinality, first
      multiply (E+1) with all the edges. Then add 1 to all the
      edges. Then run the maxflow algorithm.
//////

Status: Tested and OK
**/


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

#define optimize()
      ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl '\n'


template<typename flow_t = long long>
struct PushRelabel {
        struct Edge {
                int to, rev;
                flow_t f, c, per_cap;
        };
        vector<vector<Edge> > g;
        vector<flow_t> ec;
        vector<Edge*> cur;
        vector<vector<int> > hs;
        vector<int> H;
        PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n)
              {}
        void add_edge(int s, int t, flow_t cap, flow_t rcap=0) {
                if (s == t) return;
                Edge a = {t, (int)g[t].size(), 0, cap, cap};
                Edge b = {s, (int)g[s].size(), 0, rcap, rcap};
                g[s].push_back(a);
                g[t].push_back(b);
        }
        void add_flow(Edge& e, flow_t f) {
                Edge &back = g[e.to][e.rev];
                if (!ec[e.to] && f)
                        hs[H[e.to]].push_back(e.to);
                e.f += f; e.c -= f;
                ec[e.to] += f;
                back.f -= f; back.c += f;
                ec[back.to] -= f;
        }
```

```
flow_t max_flow(int s, int t) {
        int v = g.size();
        H[s] = v;
        ec[t] = 1;
        vector<int> co(2*v);
        co[0] = v-1;
        for(int i=0;i<v;++i) cur[i] = g[i].data();
        for(auto &e:g[s]) add_flow(e, e.c);
        if(hs[0].size())
        for (int hi = 0;hi>=0;) {
                int u = hs[hi].back();
                hs[hi].pop_back();
                while (ec[u] > 0) // discharge u
                        if (cur[u] == g[u].data() +
                              g[u].size()) {
                                H[u] = 1e9;
                                for(auto &e:g[u])
                                        if (e.c && H[u] >
                                              H[e.to]+1)
                                                H[u] =
                                                      H[e.to]+1,
                                                      cur[u]
                                                      = &e;
                                        if (++co[H[u]],
                                              !--co[hi] &&
                                              hi < v)
                                        for(int i=0;i<v;++i)
                                                if (hi <
                                                      H[i] &&
                                                      H[i] <
                                                      v){
                                                        --co[H[i]];
                                                        H[i]
                                                              =
                                                              v
                                                              +
                                                              1;
                                                }
                                hi = H[u];
                        } else if (cur[u]->c && H[u] ==
                              H[cur[u]->to]+1)
                                add_flow(*cur[u],
                                      min(ec[u],
                                      cur[u]->c));
                        else ++cur[u];
                while (hi>=0 && hs[hi].empty()) --hi;
        }
        return -ec[s];
}
vector<pair<pair<int,int>,long long>> getActualFlow()
{
        vector<pair<pair<int,int>, long long>> vec;
        for( int i = 0; i < g.size(); ++i ) {
                for( int j = 0; j < g[i].size(); ++j ) {
                        if( g[i][j].f > 0 ) {
                                vec.push_back(
                                      make_pair(make_pair(i,
                                      g[i][j].to),
                                      g[i][j].f) );
                        }
                }
        }
        return vec;
    }
};


int main()
{
        optimize();
        int T; ///no. of test cases;
        cin >> T;
        for( int test = 1; test <= T; ++test ) {
                int N, M, s, t; ///no. of nodes; no. of edges;
                      source; sink;
                cin >> N >> M >> s >> t;
                PushRelabel<> fl(N+1); ///total no. of nodes is
                      N;
                int u, v, w;
                for (int i = 1; i <= M; ++i) {
                        cin >> u >> v >> w;
                        fl.add_edge(u, v, w); /// Directed graph
                        ///fl.add_edge(u, v, w, w); ///
                              unDirected graph
                }
                cout << "Case " << test << ":" << endl;
                cout << fl.max_flow(s, t) << endl;  ///value of
                      maxFlow;
                vector<pair<pair<int,int>,long long>> vec =
                      fl.getActualFlow();  ///gets actual flow;
                for( auto xx : vec ) {
                        cout << xx.first.first << " " <<
                              xx.first.second << " " << xx.second
                              << endl; ///node; node; flow;
                }
        }
        return 0;
}
```

4.8 Push relabel with gap heuristic and lowest labeling

```
/**
Push relabel with gap heuristic and lowest labeling
Complexity : O(V^2 * sqrt(E))

Not so fast max flow implementation, but is said to work faster
      than highest labeling in very few worst cases(highly
      unlikely).

1. Works on directed graph
2. Works on undirected graph
3. Works on multi-edge(directed/undirected) graph
4. Works on self-loop(directed/undirected) graph

Can find the actual flow.

Does not work for unsigned types.

//////
Implement it on your own. very easy. just a simple O(n) dfs.

Can find minimum cut sets(A and B).
A contains source itself and the nodes that are reachable from
      source using non-saturated edges.
B contains sink and the nodes that are not reachable from
      source using non-saturated edges.
Value of minimum cut capacity is summation of the per_cap of
      all edges u->v such that u belongs to set A, v belongs to
      set B.
Value of minimum cut flow is = (summation of the flow of all
      edges u->v such that u belongs to set A, v belongs to set
      B) - (summation of the flow of all edges v->u such that v
      belongs to set B, u belongs to set A)
Value of minimum cut capacity == maxflow.

To find the minimum cut sets(A and B) first find the maxflow.
      Then, we apply dfs from the source. This dfs will be such
      that- only
the nodes of set A will eventually be marked visited. Suppose
      we have reached in node u, that means u belongs to set A.
      Now,
If (flow of edge u->v) < (per_cap of edge u->v) then v belongs
      to set A.
else if(flow of edge u->v) == (per_cap of edge u->v) then v
      belongs to set B.

To find the minimum cut sets with minimum cardinality, first
      multiply (E+1) with all the edges. Then add 1 to all the
      edges. Then run the maxflow algorithm.
//////

Status: Tested and OK
**/


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

#define optimize()
      ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl '\n'


long long push_count = 0;
long long arc_scans = 0;
template<typename flow_t = long long>
struct PushRelabel {
    struct Edge {
        int to, rev;
        flow_t f, c, per_cap;
    };
    vector<vector<Edge> > g;
    vector<flow_t> ec;
    vector<Edge*> cur;
    vector<vector<int> > hs;
```

```
    vector<int> H;
    PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}
    int add_edge(int s, int t, flow_t cap, flow_t rcap=0) {
        if (s == t) return -1;
        Edge a = {t, (int)g[t].size(), 0, cap, cap};
        Edge b = {s, (int)g[s].size(), 0, rcap, rcap};
        g[s].push_back(a);
        g[t].push_back(b);
        return b.rev;
    }
    void add_flow(Edge& e, flow_t f) {
        ++push_count;
        Edge &back = g[e.to][e.rev];
        if (!ec[e.to] && f)
            hs[H[e.to]].push_back(e.to);
        e.f += f; e.c -= f;
        ec[e.to] += f;
        back.f -= f; back.c += f;
        ec[back.to] -= f;
    }
    flow_t max_flow(int s, int t) {
        push_count = arc_scans = 0;
        int v = g.size();
        H[s] = v;
        ec[t] = 1;
        vector<int> co(2*v);
        co[0] = v-1;
        for(int i=0;i<v;++i) cur[i] = g[i].data();
        for(auto &e:g[s]) add_flow(e, e.c);
        if(hs[0].size())
        for (int hi = 0;hi<2*v;) {
            int u = hs[hi].back();
            hs[hi].pop_back();
            while (ec[u] > 0) // discharge u
                if (cur[u] == g[u].data() + g[u].size()) {
                    int hii = H[u];
                    H[u] = 1e9;
                    for(auto &e:g[u])
                        if (e.c && H[u] > H[e.to]+1)
                            H[u] = H[e.to]+1, cur[u] = &e;
                    if (++co[H[u]], !--co[hii] && hii < v)
                        for(int i=0;i<v;++i)
                            if (hii < H[i] && H[i] < v){
                                --co[H[i]];
                                H[i] = v + 1;
                            }
                    //hi = H[u];
                } else if (cur[u]->c && H[u] == H[cur[u]->to]+1)
                    add_flow(*cur[u], min(ec[u], cur[u]->c));
                else ++cur[u];
            if(hi) --hi;
            while (hi<2*v && hs[hi].empty()) ++hi;
        }
        return -ec[s];
    }
    vector<pair<pair<int,int>,long long>> getActualFlow()
    {
        vector<pair<pair<int,int>, long long>> vec;
        for( int i = 0; i < g.size(); ++i ) {
                for( int j = 0; j < g[i].size(); ++j ) {
                        if( g[i][j].f > 0 ) {
                                vec.push_back(
                                        make_pair(make_pair(i,
                                        g[i][j].to),
                                        g[i][j].f) );
                        }
                }
        }
        return vec;
    }
};

int main()
{
        optimize();
        int T; ///no. of test cases;
        cin >> T;
        for( int test = 1; test <= T; ++test ) {
                int N, M, s, t; ///no. of nodes; no. of edges;
                    source; sink;
                cin >> N >> M >> s >> t;
                PushRelabel<> fl(N+1); ///total no. of nodes is
                    N;
                int u, v, w;
                for (int i = 1; i <= M; ++i) {
                        cin >> u >> v >> w;
                        fl.add_edge(u, v, w); /// Directed graph
                        ///fl.add_edge(u, v, w, w); ///
                            unDirected graph
                }
                cout << "Case " << test << ":" << endl;
                cout << fl.max_flow(s, t) << endl;  ///value of
                    maxFlow;
                vector<pair<pair<int,int>,long long>> vec =
                    fl.getActualFlow();  ///gets actual flow;
                for( auto xx : vec ) {
                        cout << xx.first.first << " " <<
                            xx.first.second << " " << xx.second
                            << endl; ///node; node; flow;
                }
        }
        return 0;
}
```

5 Geometry

5.1 comparison of doubles

```
bool equalTo ( double a, double b ){ if ( fabs ( a - b ) <= eps
    ) return true; else return false; }

bool notEqual ( double a, double b ){if ( fabs ( a - b ) > eps
    ) return true; else return false; }

bool lessThan ( double a, double b ){ if ( a + eps < b ) return
    true; else return false; }

bool lessThanEqual ( double a, double b ){if ( a < b + eps )
    return true; else return false;}

bool greaterThan ( double a, double b ){if ( a > b + eps )
    return true;else return false;}

bool greaterThanEqual ( double a, double b ){if ( a + eps > b )
    return true;else return false;}
```

5.2 convex hull 2D

```
//Complexity : O(NlogN)
//From cp-algorithms
//Tested code. OK.

// returns points of convex hull in CW direction.

double EPS = 1e-12;

struct PT {
        double x, y;
        PT() {}
        PT(double x, double y) : x(x), y(y) {}
        PT(const PT &p) : x(p.x), y(p.y) {}
        PT operator + (const PT &p) const { return PT(x+p.x,
            y+p.y); }
        PT operator - (const PT &p) const { return PT(x-p.x,
            y-p.y); }
        PT operator * (double c)  const { return PT(x*c, y*c );
            }
        PT operator / (double c)  const { return PT(x/c, y/c );
            }
        bool operator <(const PT &p) const {
                return x < p.x || (x == p.x && y < p.y);
        }
};

ostream &operator<<(ostream &os, const PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}

// checks if a-b-c is CW or not.
bool isPointsCW(PT a, PT b, PT c) {
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y)+EPS < 0;
}
// checks if a-b-c is CCW or not.
bool isPointsCCW(PT a, PT b, PT c) {
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) > EPS;
}
// checks if a-b-c is collinear or not.
bool isPointsCollinear(PT a, PT b, PT c) {
    return abs(a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y)) <=
        EPS;
}
```

```
void convex_hull(vector<PT>& a) {
      int a_sz = a.size();
   if (a_sz == 1) return;
   sort(a.begin(), a.end());
   PT p1 = a[0], p2 = a.back();
   vector<PT> up, down; // up will store lower hull/upper
        envelope, down will store upper hull/lower envelope
   up.push_back(p1);
   down.push_back(p1);
   int up_sz = 1, down_sz = 1;
   for (int i = 1; i < a_sz; i++) {
      if (i == a_sz - 1 || isPointsCW(p1, a[i], p2)) {
         while (up_sz >= 2 && !isPointsCW(up[up_sz-2],
              up[up_sz-1], a[i])) {
            up.pop_back();
                        --up_sz;
         }
         up.push_back(a[i]);
         ++up_sz;
      }
      if (i == a_sz - 1 || isPointsCCW(p1, a[i], p2)) {
         while(down_sz >= 2 && !isPointsCCW(down[down_sz-2],
              down[down_sz-1], a[i])) {
            down.pop_back();
                        --down_sz;
         }
         down.push_back(a[i]);
         ++down_sz;
      }
   }

      // up has lower hull/upper envelope
      // down has upper hull/lower envelope
      // a has the overall convex hull

   a.clear();
   for (int i = 0; i < up_sz; i++)
      a.push_back(up[i]);
   for (int i = down_sz - 2; i > 0; i--)
      a.push_back(down[i]);
}
```

5.3 geo routine

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

#define optimize()
     ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl '\n'

typedef long long ll;
typedef vector<int> vi;
typedef vector<ll> vl;
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;
typedef vector<pii> vii;
typedef vector<pll> vll;
typedef long double ld;

#define F first
#define S second
#define MP make_pair
#define PB push_back

double INF = 1e100;
double EPS = 1e-12;

struct PT {
        double x, y;
        PT() {}
        PT(double x, double y) : x(x), y(y) {}
        PT(const PT &p) : x(p.x), y(p.y) {}
        PT operator + (const PT &p) const { return PT(x+p.x,
              y+p.y); }
        PT operator - (const PT &p) const { return PT(x-p.x,
              y-p.y); }
        PT operator * (double c)  const { return PT(x*c, y*c );
              }
        PT operator / (double c)  const { return PT(x/c, y/c );
              }
        bool operator <(const PT &p) const {
                return x < p.x || (x == p.x && y < p.y);
        }
        bool operator ==(const PT &p) const {
                return (fabs(x-p.x) <= EPS && fabs(y-p.y) <=
                      EPS);
        }
};

double dot(PT p, PT q)  { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q) { return dot(p-q,p-q); }
double cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}

// checks if a-b-c is CW or not.
bool isPointsCW(PT a, PT b, PT c) {
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y)+EPS < 0;
}
// checks if a-b-c is CCW or not.
bool isPointsCCW(PT a, PT b, PT c) {
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) > EPS;
}
// checks if a-b-c is collinear or not.
bool isPointsCollinear(PT a, PT b, PT c) {
    return abs(a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y)) <=
         EPS;
}

// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p) { return PT(-p.y,p.x); }
PT RotateCW90(PT p)  { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) { // rotate a point CCW t degrees
      around the origin
  return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

// rotate a point p CCW around another point q by angle
     t(radian)
//
     https://stackoverflow.com/questions/2259476/rotating-a-point-about-
PT RotateCCW_around_point(PT p, PT q, double t) {
  return PT((p.x-q.x)*cos(t)-(p.y-q.y)*sin(t)+q.x,
        (p.x-q.x)*sin(t)+(p.y-q.y)*cos(t)+q.y);
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
        return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
        double r = dot(b-a,b-a);
        if (fabs(r) < EPS) return a;
        r = dot(c-a, b-a)/r;
        if (r+EPS < 0) return a;
        if (r > 1+EPS) return b;
        return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
        return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
                          double a, double b, double c, double d)
{
        return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are parallel or
     collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
        return fabs(cross(b-a, c-d)) < EPS;
}

bool LinesCollinear(PT a, PT b, PT c, PT d) {
        return LinesParallel(a, b, c, d)
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
      dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
    if (dot(c-a, c-b) > EPS && dot(d-a, d-b) > EPS && dot(c-b,
          d-b) > EPS)
```

```
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > EPS) return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > EPS) return false;
  return true;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
        b=b-a; d=c-d; c=c-a;
        assert(dot(b, b) > EPS && dot(d, d) > EPS);
        return a + b*cross(c, d)/cross(b, d);
}

// shift the straight line passing through points a and b
// by distance Dist.
// If Dist is negative the line is shifted rightwards or
      upwards.
// If Dist is positive the line is shifted leftwards or
      downwards.
// The new line passes through points c and d
//
      https://math.stackexchange.com/questions/2593627/i-have-a-line-i-want-to-move-the-line-a-certain-distance-away-parallelly/2594547
pair<PT,PT> ShiftLineByDist(PT a, PT b, double Dist) {
        double r = sqrt( dist2(a, b) );
        double delx = (Dist*(a.y-b.y))/r;
        double dely = (Dist*(b.x-a.x))/r;
        PT c = PT(a.x+delx, a.y+dely);
        PT d = PT(b.x+delx, b.y+dely);
        return MP(c, d);
}

// This code computes the area or centroid of a (possibly
      nonconvex)
// polygon, assuming that the coordinates are listed in a
      clockwise or
// counterclockwise fashion. Note that the centroid is often
      known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
        double area = 0;
        for(int i = 0; i < p.size(); i++) {
                int j = (i+1) % p.size();
                area += p[i].x*p[j].y - p[j].x*p[i].y;
        }
        return area / 2.0;
}

double ComputeArea(const vector<PT> &p) {
        return fabs(ComputeSignedArea(p));
}

PT ComputeCentroid(const vector<PT> &p) {
        PT c(0,0);
        double scale = 6.0 * ComputeSignedArea(p);
        for (int i = 0; i < p.size(); i++){
                int j = (i+1) % p.size();
                c = c + (p[i]+p[j])*(p[i].x*p[j].y -
                    p[j].x*p[i].y);
        }
        return c / scale;
}

// angle from p2->p1 to p2->p3, returns -PI to PI
double angle(PT p1, PT p2, PT p3)
{
    PT va = p1-p2,vb=p3-p2;
    double x,y;
    x=dot(va,vb);
    y=cross(va,vb);
    return(atan2(y,x));
}

double angle_to_radian( double theta )
{
        return ((M_PI/180.0)*theta);
}

double radian_to_angle( double x )
{
        return ((180.0/M_PI)*x);
}
int main()
{
        // expected: (-5,2)
        cerr << RotateCCW90(PT(2,5)) << endl;

        // expected: (5,-2)
        cerr << RotateCW90(PT(2,5)) << endl;

        // expected: (-5,2)
        cerr << RotateCCW(PT(2,5),M_PI/2) << endl;

        // expected: (5,2)
        cerr << ProjectPointLine(PT(-5,-2), PT(10,4), PT(3,7))
             << endl;

        // expected: (5,2) (7.5,3) (2.5,1)
        cerr << ProjectPointSegment(PT(-5,-2), PT(10,4),
             PT(3,7)) << " "
               << ProjectPointSegment(PT(7.5,3), PT(10,4),
                    PT(3,7)) << " "
               << ProjectPointSegment(PT(-5,-2), PT(2.5,1),
                    PT(3,7)) << endl;

        // expected: 6.78903
        cerr << DistancePointPlane(4,-4,3,2,-2,5,-8) << endl;


        // expected: 1 0 1
        cerr << LinesParallel(PT(1,1), PT(3,5), PT(2,1),
             PT(4,5)) << " "
               << LinesParallel(PT(1,1), PT(3,5), PT(2,0),
                    PT(4,5)) << " "
               << LinesParallel(PT(1,1), PT(3,5), PT(5,9),
                    PT(7,13)) << endl;

        // expected: 0 0 1
        cerr << LinesCollinear(PT(1,1), PT(3,5), PT(2,1),
             PT(4,5)) << " "
               << LinesCollinear(PT(1,1), PT(3,5), PT(2,0),
                    PT(4,5)) << " "
               << LinesCollinear(PT(1,1), PT(3,5), PT(5,9),
                    PT(7,13)) << endl;

        // expected: (1,2)
        cerr << ComputeLineIntersection(PT(0,0), PT(2,4),
             PT(3,1), PT(-1,3)) << endl;

        // area should be 5.0
        // centroid should be (1.1666666, 1.166666)
        PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };
        vector<PT> p(pa, pa+4);
        PT c = ComputeCentroid(p);
        cerr << "Area: " << ComputeArea(p) << endl;
        cerr << "Centroid: " << c << endl;

        // expected: 0
        cerr << isPointsCCW( PT(5, 6), PT(10, 10), PT(11, 5) )
             << endl;
        // expected: 1
        cerr << isPointsCCW( PT(5, 6), PT(10, 2), PT(11, 5) )
             << endl;
        // expected: 1
        cerr << isPointsCW( PT(5, 6), PT(10, 10), PT(11, 5) )
             << endl;
        // expected: 0
        cerr << isPointsCW( PT(5, 6), PT(10, 2), PT(11, 5) ) <<
             endl;
        // expected: 0
        cerr << isPointsCollinear( PT(5, 6), PT(10, 2), PT(11,
             5) ) << endl;
        // expected: 1
        cerr << isPointsCollinear( PT(5, 6), PT(10, 6), PT(11,
             6) ) << endl;

        // expected: (-0.437602,12.6564) (2.5624,14.6564)
        cerr << ShiftLineByDist( PT(4, 6), PT(7, 8), 8 ).F << "
             " << ShiftLineByDist( PT(4, 6), PT(7, 8), 8 ).S <<
             endl;
        // expected: (8.4376,-0.656402) (11.4376,1.3436)
        cerr << ShiftLineByDist( PT(4, 6), PT(7, 8), -8 ).F <<
             " " << ShiftLineByDist( PT(4, 6), PT(7, 8), -8 ).S
             << endl;


}
```

5.4 half plane intersection Radewoosh style$_O FFLINE$

```
// OFFLINE
// Complexity: O(NlgN)
// very easy concept and implementation
// https://codeforces.com/blog/entry/61710

double INF = 1e100;
double EPS = 1e-12;

struct PT {
	double x, y;
	PT() {}
	PT(double x, double y) : x(x), y(y) {}
	PT(const PT &p) : x(p.x), y(p.y) {}
	PT operator + (const PT &p) const { return PT(x+p.x,
	     y+p.y); }
	PT operator - (const PT &p) const { return PT(x-p.x,
	     y-p.y); }
	PT operator * (double c)  const { return PT(x*c, y*c );
	     }
	PT operator / (double c)  const { return PT(x/c, y/c );
	     }
	bool operator <(const PT &p) const {
		return x < p.x || (x == p.x && y < p.y);
	}
};

ostream &operator<<(ostream &os, const PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}


int steps = 600;

vector<PT> lower_hull, upper_hull;
int lower_hull_sz, upper_hull_sz;

bool leBorder = 0, riBorder = 0;

double func( double xx, double val )
{
	double ans1 = INF, ans2 = -INF, ans;
	for( int i = 0; i < lower_hull_sz-1; ++i ) {
		if( leBorder && (i == 0) ) continue;
		PT a = lower_hull[i], b = lower_hull[i+1];
		          // straight line passes through
		     points a and b
		double m = (a.y-b.y)/(a.x-b.x);     // slope of
		     the straight line; if the TL is strict,
		     then better precalculate all the slopes and
		     store them beforehand
		double c = a.y - a.x*(m);           // intercept
		     of the straight line; if the TL is strict,
		     then better precalculate all the intercepts
		     and store them beforehand
		double aa = m*xx;
		double bb = c;
		double cc = aa+bb;
		ans1 = min( ans1, cc );
	}
	for( int i = 0; i < upper_hull_sz-1; ++i ) {
		if( riBorder && (i == upper_hull_sz-2) )
			continue;
		PT a = upper_hull[i], b = upper_hull[i+1];
		          // straight line passes through
		     points a and b
		double m = (a.y-b.y)/(a.x-b.x);     // slope of
		     the straight line; if the TL is strict,
		     then better precalculate all the slopes and
		     store them beforehand
		double c = a.y - a.x*(m);           // intercept
		     of the straight line; if the TL is strict,
		     then better precalculate all the intercepts
		     and store them beforehand
		double aa = m*xx;
		double bb = c;
		double cc = aa+bb;
		ans2 = max( ans2, cc );
	}
	ans = ans1-ans2;
	return ans;
}

bool Ternary_Search(double val)
{
	double lo = -INF, hi = INF, mid1, mid2;
	leBorder = 0, riBorder = 0;
	if( lower_hull[0].x == lower_hull[1].x ) lo =
	     lower_hull[0].x+val, leBorder = 1;
	if( upper_hull[upper_hull_sz-2].x ==
	     upper_hull[upper_hull_sz-1].x ) hi =
	     upper_hull[upper_hull_sz-1].x-val, riBorder = 1;
	if( lo > hi ) return 0;
	for( int i = 0; i < steps; ++i ) {
		mid1 = (lo*2.0 + hi)/3.0;
		mid2 = (lo + 2.0*hi)/3.0;
		double ff1 = func(mid1, val);
		double ff2 = func(mid2, val);
		if( ff1 >= 0 || ff2 >= 0 ) return 1;
		if( ff1 > ff2 ) hi = mid2;
		else lo = mid1;
	}
	if( func(lo, val) >= 0 ) return 1;
	return 0;
}
```

6 Graph

6.1 2-SAT

```
///2_SAT


///Complexity: O(N)
///https://cp-algorithms.com/graph/2SAT.html
///While practicing, its better to revise the concept of 2SAT
     from cp-algorithms and steven halim.
/**
	1. p xor q == (p or q) and (not_p or not_q)
	   not_(p xor q) == (not_p or q) and (p or not_q)

	2. For any or relation (p or b) we will add directed
	     edges:-
	   not_p -> b, not_b -> p

	3. The node for x is 2*x. Node for not_x is 2*x+1.

	4. If any node x and its inverse not_x is in same SCC
	     then the problem is not 2-satisfiable. Otherwise
	     it is.

	5. To find the actual assignment-
	   any node x will be given true if its component is
		situated to the left of the component of its
		inverse not_x- in the topological ordering.
**/

///After making the implication graph, if the graph is
     undirected, then instead of using the SCC method we can
     directly use the DSU method to find the strongly
     connected components which is a little bit simpler, and
     helps to solve faster in some problems. (in undirected
     graphs, if two nodes are in same component then they are
     said to be strongly connected)

int nds; ///no. of nodes (x and not_x). we will use [2,nds] for
     denoting nodes.
vector<vector<int>> g, gt; ///g is the implication graph. gt is
     the reverse implication graph
vector<bool> used;
vector<int> order, comp;
vector<bool> assignment;     ///contains the solution.

void dfs1(int u)
{
    used[u] = true;
    for (auto v : g[u]) {
        if (!used[v]) dfs1(v);
    }
    order.push_back(u);
}

void dfs2(int u, int cl)
{
    comp[u] = cl;
    for (auto v : gt[u]) {
        if (comp[v] == -1) dfs2(v, cl);
    }
}

bool solve_2SAT()
{
    used.assign(nds+1, false);
    for (int i = 2; i <= nds; ++i) {
        if (!used[i]) dfs1(i);
```

```
    }

    comp.assign(nds+1, -1);
    for (int i = 0, j = 0; i < nds-1; ++i) {
        int u = order[nds - i - 2];
        if (comp[u] == -1) dfs2(u, j++);
    }

    assignment.assign( (nds+1)/2, false );
    for (int i = 2; i <= nds; i += 2) {
        if (comp[i] == comp[i + 1])
            return false;
        assignment[i / 2] = comp[i] > comp[i + 1];
    }
    return true;
}
```

6.2 articulation bridges

```
//complexity: O(E+V)
//https://cp-algorithms.com/graph/bridge-searching.html
//nodes are 1-based indexed.

int n; // number of nodes
vector<vector<int>> adjlist; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int u, int par = -1) {
    visited[u] = true;
    tin[u] = low[u] = timer++;
    for (auto to : adjlist[u]) {
        if (to == par) continue;
        if (visited[to]) {
            low[u] = min(low[u], tin[to]);
        }
        else {
            dfs(to, u);
            low[u] = min(low[u], low[to]);
            if (low[to] > tin[u]) {
                IS_BRIDGE(u, to);
            }
        }
    }
}

void find_bridges() {
    timer = 0;
    visited.assign(n+1, false);
    tin.assign(n+1, -1);
    low.assign(n+1, -1);
    for (int i = 1; i <= n; ++i) {
        if (!visited[i]) dfs(i);
    }
}
```

6.3 dsu on trees

```
//https://codeforces.com/blog/entry/44351

Hi!

Most of the people know about dsu but what is the "dsu on tree"?

In Iran, we call this technique "Guni" (the word means "sack"
      in English), instead of "dsu on tree".

I will explain it and post ends with several problems in CF
      that can be solved by this technique.

What is the dsu on tree?
With dsu on tree we can answer queries of this type:

How many vertices in the subtree of vertex v has some property
      in time (for all of the queries)?

For example:

Given a tree, every vertex has color. Query is how many
      vertices in subtree of vertex v are colored with color c?

Let's see how we can solve this problem and similar problems.

First, we have to calculate the size of the subtree of every
      vertice. It can be done with simple dfs:

int sz[maxn];
void getsz(int v, int p){
    sz[v] = 1; // every vertex has itself in its subtree
    for(auto u : g[v])
        if(u != p){
            getsz(u, v);
            sz[v] += sz[u]; // add size of child u to its
                  parent(v)
        }
}
Now we have the size of the subtree of vertex v in sz[v].

The naive method for solving that problem is this code(that
      works in O(N ^ 2) time)

int cnt[maxn];
void add(int v, int p, int x){
    cnt[ col[v] ] += x;
    for(auto u: g[v])
        if(u != p)
            add(u, v, x)
}
void dfs(int v, int p){
    add(v, p, 1);
    //now cnt[c] is the number of vertices in subtree of vertex
          v that has color c. You can answer the queries easily.
    add(v, p, -1);
    for(auto u : g[v])
        if(u != p)
            dfs(u, v);
}
Now, how to improve it? There are several styles of coding for
      this technique.

1. easy to code but .
map<int, int> *cnt[maxn];
void dfs(int v, int p){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p){
           dfs(u, v);
           if(sz[u] > mx)
               mx = sz[u], bigChild = u;
       }
    if(bigChild != -1)
        cnt[v] = cnt[bigChild];
    else
        cnt[v] = new map<int, int> ();
    (*cnt[v])[ col[v] ] ++;
    for(auto u : g[v])
       if(u != p && u != bigChild){
           for(auto x : *cnt[u])
               (*cnt[v])[x.first] += x.second;
       }
    //now (*cnt[v])[c] is the number of vertices in subtree of
          vertex v that has color c. You can answer the queries
          easily.

}

2. easy to code and .
vector<int> *vec[maxn];
int cnt[maxn];
void dfs(int v, int p, bool keep){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p && sz[u] > mx)
           mx = sz[u], bigChild = u;
    for(auto u : g[v])
       if(u != p && u != bigChild)
           dfs(u, v, 0);
    if(bigChild != -1)
        dfs(bigChild, v, 1), vec[v] = vec[bigChild];
    else
        vec[v] = new vector<int> ();
    vec[v]->push_back(v);
    cnt[ col[v] ]++;
    for(auto u : g[v])
       if(u != p && u != bigChild)
           for(auto x : *vec[u]){
               cnt[ col[x] ]++;
               vec[v] -> push_back(x);
           }
    //now (*cnt[v])[c] is the number of vertices in subtree of
          vertex v that has color c. You can answer the queries
          easily.
    // note that in this step *vec[v] contains all of the
          subtree of vertex v.
    if(keep == 0)
```

```
        for(auto u : *vec[v])
            cnt[ col[u] ]--;
}

3. heavy-light decomposition style .
int cnt[maxn];
bool big[maxn];
void add(int v, int p, int x){
    cnt[ col[v] ] += x;
    for(auto u: g[v])
        if(u != p && !big[u])
            add(u, v, x)
}
void dfs(int v, int p, bool keep){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p && sz[u] > mx)
          mx = sz[u], bigChild = u;
    for(auto u : g[v])
        if(u != p && u != bigChild)
            dfs(u, v, 0); // run a dfs on small childs and clear
                  them from cnt
    if(bigChild != -1)
        dfs(bigChild, v, 1), big[bigChild] = 1; // bigChild
              marked as big and not cleared from cnt
    add(v, p, 1);
    //now cnt[c] is the number of vertices in subtree of vertex
         v that has color c. You can answer the queries easily.
    if(bigChild != -1)
        big[bigChild] = 0;
    if(keep == 0)
        add(v, p, -1);
}

4. My invented style .
This implementation for "Dsu on tree" technique is new and
      invented by me. This implementation is easier to code
      than others. Let st[v] dfs starting time of vertex v,
      ft[v] be it's finishing time and ver[time] is the vertex
      which it's starting time is equal to time.

int cnt[maxn];
void dfs(int v, int p, bool keep){
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p && sz[u] > mx)
          mx = sz[u], bigChild = u;
    for(auto u : g[v])
        if(u != p && u != bigChild)
            dfs(u, v, 0); // run a dfs on small childs and clear
                  them from cnt
    if(bigChild != -1)
        dfs(bigChild, v, 1); // bigChild marked as big and not
              cleared from cnt
    for(auto u : g[v])
        if(u != p && u != bigChild)
            for(int p = st[u]; p < ft[u]; p++)
                cnt[ col[ ver[p] ] ]++;
    cnt[ col[v] ]++;
    //now cnt[c] is the number of vertices in subtree of vertex
         v that has color c. You can answer the queries easily.
    if(keep == 0)
        for(int p = st[v]; p < ft[v]; p++)
            cnt[ col[ ver[p] ] ]--;
}

But why it is ? You know that why dsu has time (for q queries);
      the code uses the same method. Merge smaller to greater.

If you have heard heavy-light decomposition you will see that
      function add will go light edges only, because of this,
      code works in time.

Any problems of this type can be solved with same dfs function
      and just differs in add function.
```

6.4 euler$_{c}ircuit_{a}nd_{p}ath$

```
///The implementation for euler circuit and euler path is
      exactly same.
//http://www.graph-magics.com/articles/euler.php

/**
        1. Path traversing all the edges just once starting
              from one node and ending at that node is euler
              circuit.
        2. If the path does not end at the source node then
              that path is euler path.
**/

///euler circuit/euler path for undirected graph
/**
        1. The graph must be a single component. So, you have
              to check it.
        2. All the nodes should have even degree. So, check it.
              But if just 2 nodes have odd degree
        then there will an eulerian path only. Choose any of
              these 2 nodes as source node in that case.
**/
multiset<int> gset[10005];   //gset contains the graph info as
      adjacency list
vector<int> circuit;  //contains the path
void eularcircuit_U(int src)
{
        stack<int> curr_path;

        circuit.clear();
        if(gset[src].empty())return;
        int curr_v = src;

        while (1) {
                if (!gset[curr_v].empty()) {
                        curr_path.push(curr_v);
                        auto it=gset[curr_v].begin();
                        int next_v = *it;
                        gset[curr_v].erase(gset[curr_v].lower_bound(next_v));
                        gset[next_v].erase(gset[next_v].lower_bound(curr_v
                        curr_v = next_v;
                }
                else {
                        circuit.push_back(curr_v);
                        if(curr_path.empty()) break;
                        curr_v = curr_path.top();
                        curr_path.pop();
                }
        }

        reverse(circuit.begin(),circuit.end());
}

/// Euler circuit/euler path for directed graph
/**
        1. The graph -if converted to undirected must be a
              single component. So, you have to check it.
        2. in-degree and out-degree of all nodes should be
              equal. So, check it. But if just 2 nodes have
              unequal in-degree
        and out-degree then there will an eulerian path only if
              and only if one these node(source node to be
              exact) have
        out-degree==1+in-degree and the other have
              in-degree==1+out-degree. Choose any of these
        2 nodes as source node in that case.
**/
vector<int> gvec[10005]; //gvec contains the graph info as
      adjacency list
vector<int> circuit;  // contains the path
void eularcircuit_D(int src)
{
        stack<int> curr_path;

        circuit.clear();
        if(gvec[src].empty())return;
        int curr_v = src;

        while (1) {
                if (!gvec[curr_v].empty()) {
                        curr_path.push(curr_v);
                        int next_v = gvec[curr_v].back();
                        gvec[curr_v].pop_back();
                        curr_v = next_v;
                }
                else {
                        circuit.push_back(curr_v);
                        if(curr_path.empty()) break;
                        curr_v = curr_path.top();
                        curr_path.pop();
                }
        }

        reverse(circuit.begin(),circuit.end());
}
```

6.5 FloydWarshall

```
// order of nodes: k->i->j

for( int k = 1; k <= n; ++k ) {
        for( int i = 1; i <= n; ++i ) {
                for( int j = 1; j <= n; ++j ) {
                                adjmatrix[i][j] = min(
                                      adjmatrix[i][j],
                                      adjmatrix[i][k]+adjmatrix[k][j]
                                      );
                }
        }
}
```

6.6 lca

```
//https://cp-algorithms.com/graph/lca_binary_lifting.html

int n, l;
vector<vector<int>> adjlist;

int timer;
vector<int> tin, tout;
vector<vector<int>> up;

void dfs(int u, int par)
{
    tin[u] = ++timer;
    up[u][0] = par;
    for (int i = 1; i <= l; ++i) up[u][i] = up[up[u][i-1]][i-1];
    for (auto v : adjlist[u]) {
        if (v != par) dfs(v, u);
    }
    tout[u] = ++timer;
}

bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u, int v)
{
    if (is_ancestor(u, v)) return u;
    if (is_ancestor(v, u)) return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v)) u = up[u][i];
    }
    return up[u][0];
}

void preprocess(int root) {
    tin.resize(n+1);
    tout.resize(n+1);
    timer = 0;
    l = ceil(log2(n+1));
    up.assign(n+1, vector<int>(l + 1));
    dfs(root, root);
}
```

6.7 topological sort

```
///tarjan method.
///complexity: O(N)

vi adjlist[10005];
bool vis[10005];
vector<int> toposort;

void dfs( int u )
{
        vis[u] = 1;
        for( auto v : adjlist[u] ) {
                if( !vis[v] ) dfs(v);
        }
        toposort.PB(u);
}

int main()
{
        for( int i = 1; i <= n; ++i ) {
                if( !vis[i] ) dfs(i);
        }
        reverse( toposort.begin(), toposort.end() );
}
```

7 important$_C++_features$

```
///Unique vector
sort(vec.begin(), vec.end());
vec.erase( unique( vec.begin(), vec.end() ), vec.end() );

///bit manipulation
inline bool checkBit(ll n, int i) { return n&(1LL<<i); }
inline ll setBit(ll n, int i) { return n|(1LL<<i);; }
inline ll resetBit(ll n, int i) { return n&(~(1LL<<i)); }

///dir array
int dx[] = {0, 0, +1, -1};
int dy[] = {+1, -1, 0, 0};
//int dx[] = {+1, 0, -1, 0, +1, +1, -1, -1};
//int dy[] = {0, +1, 0, -1, +1, -1, +1, -1};


///constructor overwriting
struct Vertex {
    int next[K];
    bool leaf = false;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

///substring copy(Very fast. faster than hand-made custom. can
      avoid TLE)
string s;
s.substr(idx); //copy substring of range [idx, s.size()-1]
s.substr(idx, x); //copy substring of range [idx, idx+x-1]


///priority queue in ascending order
priority_queue<int, vector<int>, greater<int> > pq;

///random number generator
mt19937
      rng((unsigned)chrono::system_clock::now().time_since_epoch().count
      //mt199937_64 for ll (**use this above main function**)

shuffle( vec.begin(), vec.end(), rng ); ///shuffles a vector
int temp = rng();     ///generates a random number
```

8 Number Theory

8.1 bigmod

```
ll expo( ll b, ll p, ll m ) {
    ll res = 1LL, x = b%m;
    while(p) {
        if ( p&1LL ) res = ( res*x ) % m;
        x = ( x*x ) % m;
        p >>= 1LL;
    }
    return res;
}
```

8.2 factorial$_factorize$

```
//https://forthright48.com/prime-factorization-of-factorial


///How many times prime p occurs in n! (O(lgn))
long long factorialPrimePower ( long long n, long long p ) {
    long long freq = 0;
    long long x = n;
```

```
    while ( x ) {
        freq += x / p;
        x = x / p;
    }

    return freq;
}


///prime factorization of n! (O(nlgn))
void factFactorize ( long long n ) {
    for ( int i = 0; i < prime.size() && prime[i] <= n; i++ ) {
        ll x = n;
        ll freq = 0;

        while ( x ) {
            freq += x / prime[i];
            x = x / prime[i];
        }

                cout << prime[i] << '^' << freq << endl;
    }
}
```

8.3 mod$_{i}nverse_{f}rom_{1t}o_{N}$

```
///mod inverse from 1 to N
///Complexity: O(N)


const int mx = 100005;
ll inv[mx];

void generate_modinv()
{
        inv[1] = 1;
        for( int i = 2; i < mx; i++ ) {
                inv[i] = (-(MOD/i) * inv[MOD%i] ) % MOD;
                inv[i] = inv[i] + MOD;
        }
}
```

8.4 sieve

```
const int MAXN = 1e6+5;

vector<int> prime;
bool is_composite[MAXN];

void sieve () {
        for (int i = 2; i < MAXN; ++i) {
                if (!is_composite[i]) prime.push_back(i);
                for (int j = 0; j < prime.size() && i * prime[j]
                    < MAXN; ++j) {
                        is_composite[i * prime[j]] = true;
                        if (i % prime[j] == 0) break;
                }
        }
}
```

8.5 Tonelli-Shanks(for solving quadratic congruence equation)

```
//solves TIMUS 1132
//can't figure out the time complexity, but its fast

/**
for details-
1. https://www.codechef.com/wiki/tutorial-number-theory#
2. https://www.codechef.com/wiki/tutorial-quadratic-equations#
**/
//This function solves quadratic equation modulo prime p;
//returns -1 if there is no possible solution. i.e the
     congruence equation is not valid.
//returns an answer(if exists) as integer less than prime p.
//This algorithm may work for prime power moduli (according to
     wikipedia, but not tested)

//solves x*x == a (mod p);

/**
                                                  ***Important
                                                       notes***
**For even prime moduli(p = 2):
1. There is exactly 1 solution in range [1,p-1]. But when range
     is not concerned then there are infinite solutions. If a
     solution is x then the other solutions can be defined as
     k*p+x or k*p-x where k is any integer. you can prove it
     by showing that (k*p+x)^2 and x^2 are congruent modulo p
     or (k*p-x)^2 and x^2 are congruent modulo p.

**For odd prime moduli:
1.If the congruence equation is valid then there are always
     exactly 2 solutions in range [1, p-1]. if a solution is
     x(returned by the function) then the other solution is
     p-x. (Exception: if congruence equation is valid and x =
     0 is returned by the function, then p-x = 0. So, even
     though the congruence equation is valid- there are no
     solutions in range [1,p-1]. However, if range is not
     concerned then there are infinitly many solutions.)
2.But, if range is not concerned then there exists infinite
     solution if the congruence equation is valid. If a
     solution is x then the other solutions can be defined as
     k*p+x or k*p-x where k is any integer. you can prove it
     by showing that (k*p+x)^2 and x^2 are congruent modulo p
     or (k*p-x)^2 and x^2 are congruent modulo p.

**/

ll expo( ll b, ll power, ll m ) {
    ll res = 1LL, x = b%m;
    while(power) {
        if ( power&1LL ) res = ( res*x ) % m;
        x = ( x*x ) % m;
        power >>= 1LL;
    }
    return res;
}

//solves x*x == a (mod p); [Here, p is prime. We find a
     possible value of x.]
int solvequadratic(int a, int p)
{
        if(p == 2) {
                if(a&1) return 1;
                return 0;
        }
        if(a > p) a = a%p;
        while(a < 0) a = a+p;
        if(a == 0) return 0;
        if(expo(a,(p-1)/2,p) != 1) return -1;

        int n = 0, k = p-1;
        while(k % 2 == 0) k/=2, n++;
        int q = 2;
        while(expo(q,(p-1)/2,p) != (p-1)) q++;
        int t = expo(a,(k+1)/2,p);
        int r = expo(a,k,p);
        while(true) {
                int s = 1;
                int i = 0;
                while(expo(r,s,p) != 1) i+=1, s*=2;
                if(i == 0) return t;
                int e = 1<<(n-i-1);
                int u = expo(q, k*e,p);
                t = ((long long)t*u)%p;
                r = ((long long)r*u*u)%p;
    }
}
```

9 Policy based Data structures

```
/**SET STL Policy Based Data Structures (Usually Fenwik/Segment
     tree solutions are a bit faster).


References:
1) https://codeforces.com/blog/entry/15729
2) https://codeforces.com/blog/entry/5631

***The main code.
https://pastebin.com/WhuWcFj9

*/
```

```cpp
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;
using namespace std;

template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
      tree_order_statistics_node_update>; ///set in ascending
      order (the typical one)
template <typename T>
using ordered_set_of_pairs = tree<pair<T, size_t>, null_type,
      less<pair<T, size_t>>, rb_tree_tag,
      tree_order_statistics_node_update>; ///set of pairs in
      ascending order (the typical one)
template <typename T>
using ordered_set_desc = tree<T, null_type, greater<T>,
      rb_tree_tag, tree_order_statistics_node_update>; ///set
      in descending order
template <typename T>
using ordered_set_of_pairs_desc = tree<pair<T, size_t>,
      null_type, greater<pair<T, size_t>>, rb_tree_tag,
      tree_order_statistics_node_update>; ///set of pairs in
      descending order


#define optimize()
      ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
#define endl '\n'

#define MP make_pair
#define F first
#define S second
#define PB push_back

ordered_set<int> st1;
ordered_set_of_pairs<int> st2;
ordered_set_desc<int> st3;
ordered_set_of_pairs_desc<int> st4;

int main()
{
        optimize();
        for( int i = 0; i < 10; ++i ) st1.insert(i);
        cout << st1.order_of_key(2) << endl; ///how many
              elements in st1 less than 2? (output: 2)
        cout << *st1.find_by_order(5) << endl << endl; ///what
              is the 5th minimum element in st1?(0th based
              indexing) (output: 5)

        for( int i = 0; i < 10; ++i ) st2.insert(MP(i, i));
        cout << st2.order_of_key(MP(2, 3)) << endl; ///output: 3
        cout << st2.order_of_key(MP(2, 2)) << endl; ///output: 2
        cout << st2.order_of_key(MP(3, 2)) << endl; ///output: 3

        cout << st2.order_of_key(MP(3, -1)) << endl; ///output:
              4 (i know, you were expecting 3. but giving
              negative numbers as second element gives
              unexpected results.)

        cout << (*st2.find_by_order(5)).F << " " <<
              (*st2.find_by_order(5)).S << endl << endl;
              ///output: 5 5
}
```

10 Strings

10.1 manachers

```cpp
void manachers(string s, vector<int> &d1, vector<int> &d2)
{
        int n = s.size();
        /// odd length palindromes.
        d1.resize(n);
        for (int i = 0, l = 0, r = -1; i < n; i++) {
                int k = (i > r) ? 1 : min(d1[l + r - i], r - i +
                      1);
                while (0 <= i - k && i + k < n && s[i - k] ==
                      s[i + k]) {
                        k++;
                }
                d1[i] = k--;
                if (i + k > r) {
                        l = i - k;
                        r = i + k;
                }
        }

        /// even length palindromes. suppose "abba" is a
              palindrome. Here, 2nd index(o-based indexing) is
              the center.
        d2.resize(n);
        for (int i = 0, l = 0, r = -1; i < n; i++) {
                int k = (i > r) ? 0 : min(d2[l + r - i + 1], r -
                      i + 1);
                while (0 <= i - k - 1 && i + k < n && s[i - k -
                      1] == s[i + k]) {
                        k++;
                }
                d2[i] = k--;
                if (i + k > r) {
                        l = i - k - 1;
                        r = i + k ;
                }
        }
}
```

10.2 Suffix Array

```cpp
///Suffix array
//https://cp-algorithms.com/string/suffix-array.html

vector<int> sort_cyclic_shifts(string const& s) {
    int n = s.size();
    const int alphabet = 256;

    vector<int> p(n), c(n), cnt(max(alphabet, n), 0); /// p
          contains the sorted indexes. // c contains the
          equivalence classes of the indexes.(Smaller index
          means smaller suffix)
    for (int i = 0; i < n; i++)
        cnt[s[i]]++;
    for (int i = 1; i < alphabet; i++)
        cnt[i] += cnt[i-1];
    for (int i = 0; i < n; i++)
        p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i-1]])
            classes++;
        c[p[i]] = classes - 1;
    }

    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) %
                  n]};
            pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 <<
                  h)) % n]};
            if (cur != prev)
                ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
    }
    return p;
}

vector<int> suffix_array_construction(string s) {
    s += "$";
    vector<int> sorted_shifts = sort_cyclic_shifts(s);
    sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}
```

```
///compare two substrings of same string starting at i and j of
     same length l. Here, k is maximum value such that 2^k <= l
int compare(int i, int j, int l, int k) {
    pair<int, int> a = {c[k][i], c[k][(i+l-(1 << k))%n]};
    pair<int, int> b = {c[k][j], c[k][(j+l-(1 << k))%n]};
    return a == b ? 0 : a < b ? -1 : 1;
}

///lcp of any two arbitrary suffixes (p[i] and p[j]) - O(log(N))
int lcp(int i, int j) {
    int ans = 0;
    for (int k = log_n; k >= 0; k--) {
        if (c[k][i] == c[k][j]) {
            ans += 1 << k;
            i += 1 << k;
            j += 1 << k;
        }
    }
    return ans;
}

///Kasai - O(N)
///returns a vector lcp of size s.size()-2. lcp[i] is the lcp
     of p[i] and p[i+1] strings.
vector<int> lcp_construction(string const& s, vector<int>
     const& p) {
    int n = s.size();
    vector<int> rank(n, 0);
    for (int i = 0; i < n; i++)
        rank[p[i]] = i;

    int k = 0;
    vector<int> lcp(n-1, 0);
    for (int i = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = p[rank[i] + 1];
        while (i + k < n && j + k < n && s[i+k] == s[j+k])
            k++;
        lcp[rank[i]] = k;
        if (k)
            k--;
    }
    return lcp;
}
```

10.3 suffix$_array(betterversionbyAshiquul)$

```
/**

1. sa[i] = i'th suffix, i from 0 to n-1
2. everything is in 0'th base
3. lcp[i] = lcp of (i-1)th and ith suffix, i from 0 to n-1
4. adjust the alpha, usually for string ALPHA = 128 (max ascii
     value)
5. notice if clearing may cause tle
6. remove range_lcp_init() if not required
7. rev_sa[sa[i]] = i;

**/

const int MAXN = 1010; /// always take 10 extra.
const int ALPHA = 256, LOG = 12;    ///LOG is log2(MAXN)+3

struct SuffixArray {
    int sa[MAXN],data[MAXN],rnk[MAXN],height[MAXN],n;
    int wa[MAXN],wb[MAXN],wws[MAXN],wv[MAXN];
    int lg[MAXN], rmq[MAXN][LOG], rev_sa[MAXN];
    int cmp(int *r,int a,int b,int l){
        return (r[a]==r[b]) && (r[a+l]==r[b+l]);
    }
    void DA(int *r,int *sa,int n,int m){
        int i,j,p,*x=wa,*y=wb,*t;
        for(i=0;i<m;i++) wws[i]=0;
        for(i=0;i<n;i++) wws[x[i]=r[i]]++;
        for(i=1;i<m;i++) wws[i]+=wws[i-1];
        for(i=n-1;i>=0;i--) sa[--wws[x[i]]]=i;
        for(j=1,p=1;p<n;j*=2,m=p) {
            for(p=0,i=n-j;i<n;i++) y[p++]=i;
            for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j;
            for(i=0;i<n;i++) wv[i]=x[y[i]];
            for(i=0;i<m;i++) wws[i]=0;
            for(i=0;i<n;i++) wws[wv[i]]++;
            for(i=1;i<m;i++) wws[i]+=wws[i-1];
            for(i=n-1;i>=0;i--) sa[--wws[wv[i]]]=y[i];
            for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)
                x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
        }
    }
    void calheight(int *r,int *sa,int n){
        int i,j,k=0;
        for(i=1;i<=n;i++) rnk[sa[i]]=i;
        for(i=0;i<n;height[rnk[i++]]=k)
            for(k?k--:0,j=sa[rnk[i]-1];r[i+k]==r[j+k];k++);
    }
    void suffix_array (string &A) {
        n = A.size();
        for(int i=0;i<max(n+5,ALPHA);i++)
            sa[i]=data[i]=rnk[i]=height[i]=wa[i]=wb[i]=wws[i]=wv[i]=0;
        for (int i = 0; i < n; i++) data[i] = A[i];
        DA(data,sa,n+1,ALPHA);
        calheight(data,sa,n);
        for(int i = 0;i < n; i++) sa[i] = sa[i+1], height[i] =
            height[i+1], rev_sa[sa[i]] = i;
        range_lcp_init();
    }

    /** LCP for range : build of rmq table **/
    void range_lcp_init() {
        for(int i = 0; i < n; i++) rmq[i][0] = height[i];
        for(int j = 1; j < LOG; j++) {
            for(int i = 0; i < n; i++) {
                if (i+(1<<j)-1 < n) rmq[i][j] =
                    min(rmq[i][j-1],rmq[i+(1<<(j-1))][j-1]);
                else break;
            }
        }
        lg[0] = lg[1] = 0;
        for(int i = 2; i <= n; i++) {
            lg[i] = lg[i/2] + 1;
        }
    }

    /** lcp between l'th to r'th suffix **/
    int query_lcp(int l, int r) {
        assert(l <= r);
        assert(l>=0 && l<n && r>=0 && r<n);
        if(l == r) return n-sa[l];
        l++;
        int k = lg[r-l+1];
        return min(rmq[l][k],rmq[r-(1<<k)+1][k]);
    }
}SA;

///substring sort comparator function. (it is used to sort all
     possible substrings of a string)
bool cmp(pair<int,int> &lhs, pair<int,int> &rhs) {
    int l1 = lhs.first, r1 = lhs.second, l2 = rhs.first, r2 =
        rhs.second;
    bool f = 0;
    if(SA.rev_sa[l2] < SA.rev_sa[l1]) {
        swap(l1, l2);
        swap(r1, r2);
        f ^= 1;
    }
    int len1 = r1-l1+1, len2 = r2-l2+1;
    int com = SA.query_lcp(SA.rev_sa[l1], SA.rev_sa[l2]);
    if(com < min(len1, len2)) return f ^ 1;
    dbg(f);
    return (len1 < len2) ^ f;
}

int main()
{
        string s;
        cin >> s;
        SA.suffix_array(s);
        SA.range_lcp_init();
        vector<pair<int,int>> vec;
        for( int i = 0; i < s.size(); ++i ) {
            for( int j = i; j < s.size(); ++j ) {
                sub_string.PB(MP(i, j));
            }
        }
        sort(sub_string.begin(), sub_string.end(), cmp);
            ///substring sorted.
}
```

10.4 z-algorithm

```
//Z-algorithm
//https://cp-algorithms.com/string/z-function.html
//O(N)

vector<int> z_function(string s) {
    int n = (int) s.length();
    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;
}
```

11 template

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

using namespace std;

typedef long long ll;
typedef vector<int> vi;
typedef vector<ll> vl;
typedef vector<vi> vvi;
typedef vector<vl> vvl;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
typedef vector<pii> vii;
typedef vector<pll> vll;

#define endl '\n'
#define PB push_back
#define F first
#define S second
#define all(a) (a).begin(), (a).end()
#define rall(a) (a).rbegin(), (a).rend()
#define sz(x) (int) x.size()

const double PI = acos(-1);
const double eps = 1e-9;
const int inf = 2000000000;
const ll infLL = 9000000000000000000;
#define MOD 1000000007

#define mem(a, b) memset(a, b, sizeof(a))
#define sqr(a) ((a) * (a))

#define optimize() ios_base::sync_with_stdio(0); cin.tie(0);
      cout.tie(0);
#define fraction() cout.unsetf(ios::floatfield);
      cout.precision(10); cout.setf(ios::fixed,
      ios::floatfield);
#define file() freopen("input.txt", "r", stdin);
      freopen("output.txt", "w", stdout);

//debug
template<typename F,typename
      S>ostream&operator<<(ostream&os,const pair<F,S>&p){return
      os<<"("<<p.first<<", "<<p.second<<")";}
template<typename T>ostream&operator<<(ostream&os,const
      vector<T>&v){os<<"{";for(auto
      it=v.begin();it!=v.end();++it){if(it!=v.begin())os<<",
      ";os<<*it;}return os<<"}";}
template<typename T>ostream&operator<<(ostream&os,const
      set<T>&v){os<<"[";for(auto
      it=v.begin();it!=v.end();++it){if(it!=v.begin())os<<",";os<<*it;}return
      os<<"]";}
template<typename T>ostream&operator<<(ostream&os,const
      multiset<T>&v) {os<<"[";for(auto
      it=v.begin();it!=v.end();++it){if(it!=v.begin())os<<",
      ";os<<*it;}return os<<"]";}
template<typename F,typename
      S>ostream&operator<<(ostream&os,const
      map<F,S>&v){os<<"[";for(auto
      it=v.begin();it!=v.end();++it){if(it!=v.begin())os<<",
      ";os<<it->first<<" = "<<it->second;}return os<<"]";}
#define dbg(args...) do {cerr << #args << " : "; faltu(args); }
      while(0)
void faltu(){cerr << endl;}
template<typename T>void faltu(T a[],int n){for(int
      i=0;i<n;++i)cerr<<a[i]<<' ';cerr<<endl;}
template<typename T,typename...hello>void faltu(T arg,const
      hello&...rest){cerr<<arg<<' ';faltu(rest...);}

inline bool checkBit(ll n, int i) { return n & (1LL << i); }
inline ll setBit(ll n, int i) { return n | (1LL << i); }
inline ll resetBit(ll n, int i) { return n & (~(1LL << i)); }

inline void normal(ll &a) { a %= MOD; (a < 0) && (a += MOD); }
inline ll modMul(ll a, ll b) { a %= MOD; b %= MOD; normal(a);
      normal(b); return (a * b) % MOD; }
inline ll modAdd(ll a, ll b) { a %= MOD; b %= MOD; normal(a);
      normal(b); return (a + b) % MOD; }
inline ll modSub(ll a, ll b) { a %= MOD; b %= MOD; normal(a);
      normal(b); a -= b; normal(a); return a; }
inline ll modPow(ll b, ll p) { ll r = 1LL; while (p) { if (p &
      1) r = modMul(r, b); b = modMul(b, b); p >>= 1; } return
      r; }
inline ll modInverse(ll a) { return modPow(a, MOD - 2); }
inline ll modDiv(ll a, ll b) { return modMul(a, modInverse(b));
      }

int main() {
    optimize();
    // ...
    return 0;
}
```

12 ternary search

```
// complexity: O(lgN)
// checked. OK.
// https://codeforces.com/blog/entry/60702
//
      https://www.hackerearth.com/practice/algorithms/searching/ternary-s
// https://cp-algorithms.com/num_methods/ternary_search.html

// Be careful about choosing the boundary

/**
What is Unimodal function?
We are given a function f(x) which is unimodal on an interval
      [l,r]. By unimodal function, we mean one of two behaviors
      of the function:

 1. The function strictly increases first, reaches a maximum
      (at a single point or over an interval), and then
      strictly decreases.

 2. The function strictly decreases first, reaches a minimum,
      and then strictly increases.
**/

/// Ternary search on function f

/**
        This part is an unproven concept. Invented by me. Use
              it at your own risk.

        If the extrema of the unimodal function is a segment
              instead of a point
        then to get the leftmost or rightmost extremum return
              value of the
        function we may consider adding >= instead of > in the
              "if( )" part of
        the code only if necessary.
**/

// Finding the minimum function (doubles precision)

int steps = 200;

void ternary_search()
{
        double lo = -inf, hi = inf, mid1, mid2;
        for(int i = 0; i < steps; i++) {
                mid1 = (lo*2+hi) / 3.0;
                mid2 = (lo+2*hi) / 3.0;
                if(f(mid1) > f(mid2)) lo = mid1;
                else hi = mid2;
        }
    double ans = f(lo);
        return ans;
}


// Finding the maximum function (doubles precision)

void ternary_search()
{
        double lo = -inf, hi = inf, mid1, mid2;
```

```
        for(int i = 0; i < steps; i++) {
                mid1 = (lo*2+hi) / 3.0;
                mid2 = (lo+2*hi) / 3.0;
                if(f(mid1) > f(mid2)) hi = mid2;
                else lo = mid1;
        }
    double ans = f(lo);
        return ans;
}


// Finding the minimum function (search on integer range)

void ternary_search()
{
        int lo = -inf, hi = inf, mid1, mid2;
        while(hi - lo > 4) {
                mid1 = (lo + hi) / 2;
                mid2 = (lo + hi) / 2 + 1;
                if(f(mid1) > f(mid2)) lo = mid1;
                else hi = mid2;
        }
        ans = inf;
        for(int i = lo; i <= hi; i++) ans = min(ans , f(i));
}


// Finding the maximum function (search on integer range)

void ternary_search()
{
        int lo = -inf, hi = inf, mid1, mid2;
        while(hi - lo > 4) {
                mid1 = (lo + hi) / 2;
                mid2 = (lo + hi) / 2 + 1;
                if(f(mid1) > f(mid2)) hi = mid2;
                else lo = mid1;
        }
        ans = inf;
        for(int i = lo; i <= hi; i++) ans = min(ans , f(i));
}
```