Contents

[Template] Duc.cpp

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
#define sorted_array tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update>
#define llong long long
#define long llong

int n_test;

void enter()
{
}

void solve()
{
}

void cleanup()
{
}

int main()
{
    ios::sync_with_stdio(false);
            if (ifstream("test.inp")) cin.rdbuf((new ifstream("test.inp"))->rdbuf());

            cin >> n_test;
            while (n_test--)
            {
                        enter();
                        solve();
                        cleanup();
            }
}
```

[Template] Duc.java

```
import java.util.*;
import java.awt.geom.*;
import java.io.*;
import java.math.*;

class Main
{
   static Scanner fi = new Scanner (System.in);
   static PrintWriter fo = new PrintWriter (System.out);
   static int n;

   static void enter()
   {
   }

   static void solve()
   {
   }

   public static void main(String[] args)
   {
      try {fi = new Scanner (new File("test.inp"));}
      catch (FileNotFoundException e) {}

      enter ();
      solve ();

      fi.close(); fo.close();
   }
}
```

Template [Viet].cpp

```
#include <bits/stdc++.h>
#define N
#define INF
#define rep(i, a, b) for(int i = a; i <= b; i++)
#define per(i, b, a) for(int i = b; i >= a; i--)
#define bitcount(S) __builtin_popcount(S)
#define ll (long long)
#define db (double)
#define dbg(x) std::cout << #x << " " << x << std::endl
#define sz(x) x.size()
#define fi first
#define se second
using namespace std;
int main() {
   //freopen("input.txt", "r", stdin);
}
```

BufferedReader.cpp

```
#include <bits/stdc++.h>
using namespace std;
#define long int64_t

struct BufferedReader
{
   char Buf[1 << 16 | 1];
   int cur;

   void FillBuffer ()
   {
      Buf[fread(Buf, 1, 1 << 16, stdin)] = 0;
      cur = 0;
   }

   BufferedReader ()
   {
      //freopen("test.inp","r",stdin);
      FillBuffer();
   }

   void nextchar ()
   {
      ++cur;
      if (!Buf[cur]) FillBuffer();
   }
```

```
    BufferedReader& operator >> (int &t)
    {
        t = 0;
        while (!isdigit(Buf[cur])) nextchar();
        while (isdigit(Buf[cur])) t = t * 10 + Buf[cur] - '0', nextchar();

        return *this;
    }
};
```

push-relabel.cpp

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

const int n_max = 1010;

bool in_queue[n_max];
long E[n_max];
int G[n_max][n_max], F[n_max][n_max];
int H[n_max];
int n, xs, xt;

void push_flow(int x, int y)
{
	int flow = min((long)G[x][y] - F[x][y], E[x]);
	F[x][y] += flow;
	F[y][x] -= flow;
	E[x] -= flow;
	E[y] += flow;
}

void enter()
{
	int m, x, y;
	cin >> n >> m >> xs >> xt;

	while (m--)
	{
		cin >> x >> y;
		cin >> G[x][y];
	}
}

void solve()
{
	list<int> P;

	H[xs] = n;
	E[xs] = LLONG_MAX;
	for (int y = 1; y <= n; ++y)
	if (G[xs][y] > 0)
	{
		push_flow(xs, y);
		P.push_back(y);
		in_queue[y] = true;
	}

	while (!P.empty())
	{
		int x = P.front();
		int h_min = INT_MAX;

		for (int y = 1; y <= n; ++y)
		if (G[x][y] > F[x][y])
		{
			if (H[y] >= H[x]) h_min = min(h_min, H[y]);
			else
			{
				push_flow(x, y);
				if (!in_queue[y] && y != xs && y != xt)
P.push_back(y), in_queue[y] = true;
			}
		}

		if (E[x] > 0) H[x] = h_min + 1;
		else
		{
			P.pop_front();
			in_queue[x] = false;
		}
```

```
            }

            cout << E[xt] << "\n";
}

int main()
{
   ios::sync_with_stdio(false); cin.tie(nullptr);
            if (ifstream("test.inp")) cin.rdbuf((new ifstream("test.inp"))->rdbuf());

            enter();
            solve();
}
```

QTREE (HLD).cpp

```
#include <bits/stdc++.h>
using namespace std;
#define llong long long
#define long llong
#define ii pair<int,int>

const int n_max = 10010;

struct edge { int x, y, c; } E[n_max];

list<ii> G[n_max];
int M[n_max], H[n_max], I[n_max], IE[n_max], W[n_max], top_node[n_max],
heavy_child[n_max], T[n_max * 4], P[n_max][16];
int n, m, n_test;

inline bool is_heavy (int x) { return M[x] * 2 > M[P[x][0]]; }

void dfs (int x,int h)
{
   H[x] = h; M[x] = 1;

   for (list<ii>::iterator i = G[x].begin(); i != G[x].end(); ++i)
   if (i->first != P[x][0])
   {
      P[i->first][0] = x;
      W[i->first] = i->second;
      dfs(i->first, h + 1);
      M[x] += M[i->first];
   }
}

void load_path(int x)
{
   while (M[x] * 2 > M[P[x][0]]) I[x] = ++m, x = P[x][0];

   int p = x;
   while ((x = heavy_child[x]) > 0) IE[x] = m, top_node[x] = p;
}

void upd_edge (int u,int v,int x = 1,int l = 1,int r = m)
{
   if (l == r) { T[x] = v; return; }

   int k = x * 2, mid = (l + r) / 2;
   if (u <= mid)
      upd_edge(u, v, k, l, mid);
   else
      upd_edge(u, v, k + 1, mid + 1, r);

   T[x] = max(T[k], T[k + 1]);
}

int get_max (int u,int v,int x = 1,int l = 1,int r = m)
{
   if (v < l || r < u) return 0;
   if (u <= l && r <= v) return T[x];

   int k = x * 2, mid = (l + r) / 2;
   return max(get_max(u, v, k, l, mid),
           get_max(u, v, k + 1, mid + 1, r));
}

int lca (int x,int y)
{
   if (H[x] < H[y]) swap(x, y);

   for (int i = 15; i >= 0; --i)
```

```
    if (H[P[x][i]] >= H[y]) x = P[x][i];

    if (x == y) return x;

    for (int i = 15; i >= 0; --i)
    if (P[x][i] != P[y][i]) x = P[x][i], y = P[y][i];

    return P[x][0];
}

int get_max_edge (int x,int p) // p is x's ancestor
{
    int s = 0;

    while (x != p)
        if (is_heavy(x))
            if (H[top_node[x]] >= H[p]) // p is not in heavy path
            {
                s = max(s, get_max(I[x], IE[x]));
                x = top_node[x];
            } else // p is in heavy path
            {
                s = max(s, get_max(I[x], I[p] - 1));
                x = p;
            }
        else // advance normally
        {
            s = max(s, W[x]);
            x = P[x][0];
        }

    return s;
}

namespace query
{
    void change (int u,int v)
    {
        int x = (H[E[u].x] > H[E[u].y] ? E[u].x : E[u].y);

        W[x] = v;
        if (is_heavy(x)) upd_edge(I[x], v);
    }

    void query (int x,int y)
    {
        int p = lca(x, y);
        cout << max(get_max_edge(x, p), get_max_edge(y, p)) << "\n";
    }
}

void enter ()
{
    cin >> n;
    for (int i = 1; i < n; ++i)
    {
        cin >> E[i].x >> E[i].y >> E[i].c;
        G[E[i].x].push_back(ii (E[i].y, E[i].c));
        G[E[i].y].push_back(ii (E[i].x, E[i].c));
    }
}

void init ()
{
    dfs(1, 1);
    for (int j = 1; j < 16; ++j)
    for (int i = 1; i <= n; ++i)
        P[i][j] = P[P[i][j - 1]][j - 1];

    M[0] = 3 * n;
    for (int x = 1; x <= n; ++x)
    if (is_heavy(x)) heavy_child[P[x][0]] = x;

    for (int x = 1; x <= n; ++x)
    if (is_heavy(x) && heavy_child[x] == 0) load_path(x);

    for (int x = 1; x <= n; ++x)
    if (is_heavy(x)) upd_edge(I[x], W[x]);
}

void solve ()
{
    string s;
    int x, y;
```

```
  while (true)
  {
    cin >> s;
    if (s == "DONE") return;
    cin >> x >> y;

    if (s == "CHANGE") query::change(x, y);
    if (s == "QUERY") query::query(x, y);
  }
}

void clean_up ()
{
  fill_n(heavy_child + 1, n, 0);
  fill_n(top_node + 1, n, 0);
  fill_n(IE + 1, n, 0);
  fill_n(I + 1, n, 0);
  for (int i = 1; i <= n; ++i) G[i].clear();
  m = 0;
}

int main ()
{
  ios::sync_with_stdio(false); cin.tie(NULL);
  if (ifstream("test.inp")) cin.rdbuf((new ifstream("test.inp"))->rdbuf());

  cin >> n_test;
  while (n_test--)
  {
    enter();
    init();
    solve();
    clean_up();
  }
}
```

convex-hull-trick.java

```
public class ConvexHullOptimization {
  long[] A = new long[1000000];
  long[] B = new long[1000000];
  int len;
  int ptr;

  // a descends
  public void addLine(long a, long b) {
    // intersection of (A[len-2],B[len-2]) with (A[len-1],B[len-1]) must lie to the left of
intersection of (A[len-1],B[len-1]) with (a,b)
    while (len >= 2 && (B[len - 2] - B[len - 1]) * (a - A[len - 1]) >= (B[len - 1] - b) *
(A[len - 1] - A[len - 2])) {
      --len;
    }
    A[len] = a;
    B[len] = b;
    ++len;
  }

  // x ascends
  public long minValue(long x) {
    ptr = Math.min(ptr, len - 1);
    while (ptr + 1 < len && A[ptr + 1] * x + B[ptr + 1] <= A[ptr] * x + B[ptr]) {
      ++ptr;
    }
    return A[ptr] * x + B[ptr];
  }

  // Usage example
  public static void main(String[] args) {
    ConvexHullOptimization h = new ConvexHullOptimization();
    h.addLine(3, 0);
    h.addLine(2, 1);
    h.addLine(3, 2);
    h.addLine(0, 6);
    System.out.println(h.minValue(0));
    System.out.println(h.minValue(1));
    System.out.println(h.minValue(2));
    System.out.println(h.minValue(3));
  }
}
```

Edmond Blossom.java

```
import java.util.*;

public class MaxMatchingEdmonds {

  static int lca(int[] match, int[] base, int[] p, int a, int b) {
    boolean[] used = new boolean[match.length];
    while (true) {
      a = base[a];
      used[a] = true;
      if (match[a] == -1) break;
      a = p[match[a]];
    }
    while (true) {
      b = base[b];
      if (used[b]) return b;
      b = p[match[b]];
    }
  }

  static void markPath(int[] match, int[] base, boolean[] blossom, int[] p, int v, int b,
int children) {
    for (; base[v] != b; v = p[match[v]]) {
      blossom[base[v]] = blossom[base[match[v]]] = true;
      p[v] = children;
      children = match[v];
    }
  }

  static int findPath(List<Integer>[] graph, int[] match, int[] p, int root) {
    int n = graph.length;
    boolean[] used = new boolean[n];
    Arrays.fill(p, -1);
    int[] base = new int[n];
    for (int i = 0; i < n; ++i)
      base[i] = i;

    used[root] = true;
    int qh = 0;
    int qt = 0;
    int[] q = new int[n];
    q[qt++] = root;
    while (qh < qt) {
      int v = q[qh++];

      for (int to : graph[v]) {
        if (base[v] == base[to] || match[v] == to) continue;
        if (to == root || match[to] != -1 && p[match[to]] != -1) {
          int curbase = lca(match, base, p, v, to);
          boolean[] blossom = new boolean[n];
          markPath(match, base, blossom, p, v, curbase, to);
          markPath(match, base, blossom, p, to, curbase, v);
          for (int i = 0; i < n; ++i)
            if (blossom[base[i]]) {
              base[i] = curbase;
              if (!used[i]) {
                used[i] = true;
                q[qt++] = i;
              }
            }
        } else if (p[to] == -1) {
          p[to] = v;
          if (match[to] == -1)
            return to;
          to = match[to];
          used[to] = true;
          q[qt++] = to;
        }
      }
    }
    return -1;
  }

  public static int maxMatching(List<Integer>[] graph) {
    int n = graph.length;
    int[] match = new int[n];
    Arrays.fill(match, -1);
    int[] p = new int[n];
    for (int i = 0; i < n; ++i) {
      if (match[i] == -1) {
        int v = findPath(graph, match, p, i);
        while (v != -1) {
          int pv = p[v];
        int ppv = match[pv];
        match[v] = pv;
        match[pv] = v;
        v = ppv;
      }
    }
  }

  int matches = 0;
  for (int i = 0; i < n; ++i)
    if (match[i] != -1)
      ++matches;
  return matches / 2;
}

// Usage example
public static void main(String[] args) {
  int n = 4;
  List<Integer>[] g = new List[n];
  for (int i = 0; i < n; i++) {
    g[i] = new ArrayList<>();
  }
  g[0].add(1);
  g[1].add(0);
  g[1].add(2);
  g[2].add(1);
  g[2].add(3);
  g[3].add(2);
  g[0].add(3);
  g[3].add(0);
  System.out.println(2 == maxMatching(g));
}
}
```

mincost-matching.cpp

```
#include <algorithm>
#include <cstdio>
#include <cmath>
#include <vector>

using namespace std;

typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate) {
  int n = int(cost.size());

  // construct dual feasible solution
  VD u(n);
  VD v(n);
  for (int i = 0; i < n; i++) {
    u[i] = cost[i][0];
    for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
  }
  for (int j = 0; j < n; j++) {
    v[j] = cost[0][j] - u[0];
    for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);
  }

  // construct primal solution satisfying complementary slackness
  Lmate = VI(n, -1);
  Rmate = VI(n, -1);
  int mated = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if (Rmate[j] != -1) continue;
      if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
           Lmate[i] = j;
           Rmate[j] = i;
           mated++;
           break;
      }
    }
  }

  VD dist(n);
  VI dad(n);
  VI seen(n);

  // repeat until primal solution is feasible
while (mated < n) {

  // find an unmatched left node
  int s = 0;
  while (Lmate[s] != -1) s++;

  // initialize Dijkstra
  fill(dad.begin(), dad.end(), -1);
  fill(seen.begin(), seen.end(), 0);
  for (int k = 0; k < n; k++)
    dist[k] = cost[s][k] - u[s] - v[k];

  int j = 0;
  while (true) {

    // find closest
    j = -1;
    for (int k = 0; k < n; k++) {
          if (seen[k]) continue;
          if (j == -1 || dist[k] < dist[j]) j = k;
    }
    seen[j] = 1;

    // termination condition
    if (Rmate[j] == -1) break;

    // relax neighbors
    const int i = Rmate[j];
    for (int k = 0; k < n; k++) {
          if (seen[k]) continue;
          const double new_dist = dist[j] + cost[i][k] - u[i] - v[k];
          if (dist[k] > new_dist) {
            dist[k] = new_dist;
            dad[k] = j;
          }
    }
  }

  // update dual variables
  for (int k = 0; k < n; k++) {
    if (k == j || !seen[k]) continue;
    const int i = Rmate[k];
    v[k] += dist[k] - dist[j];
    u[i] -= dist[k] - dist[j];
  }
  u[s] += dist[j];

  // augment along path
  while (dad[j] >= 0) {
    const int d = dad[j];
    Rmate[j] = Rmate[d];
    Lmate[Rmate[j]] = j;
    j = d;
  }
  Rmate[j] = s;
  Lmate[s] = j;

  mated++;
}

double value = 0;
for (int i = 0; i < n; i++)
  value += cost[i][Lmate[i]];

return value;
}
```

edmondskarp [Viet].cpp

```
#include <bits/stdc++.h>
#define N 1001
#define INF (int) 1e6 + 1
#define rep(i, a, b) for(int i = a; i <= b; i++)
#define per(i, b, a) for(int i = b; i >= a; i--)
#define bitcount(S) __builtin_popcount(S)
#define ll (long long)
#define db (double)
#define dbg(x) std::cout << x << std::endl
#define sz(x) x.size()
#define fi first
#define se second
using namespace std;
int n, m, s, t;
```

```
vector<int> g[N];
int c[N][N], d[N], f[N][N];
long long res;
bool findPath() {
   queue<int> q;
   rep(i, 1, n)
      d[i] = 0;
   d[s] = -1;
   q.push(s);
   while (!q.empty()) {
      int u = q.front();
      q.pop();
      if (u == t)
         return true;
      for(auto v: g[u])
         if (!d[v] && c[u][v] > f[u][v]) {
            d[v] = u;
            q.push(v);
         }
   }
   return false;
}
void enlarge() {
   int u, v, delta = INF;
   v = t;
   while (v != s) {
      u = d[v];
      delta = min(delta, c[u][v] - f[u][v]);
      v = u;
   }
   v = t;
   while (v != s) {
      u = d[v];
      f[u][v] += delta;
      f[v][u] -= delta;
      v = u;
   }
   res += delta;
}
int main() {
   //freopen("input.txt", "r", stdin);
   scanf("%d%d%d%d", &n, &m, &s, &t);
   rep(i, 1, m) {
      int u, v, w;
      scanf("%d%d%d", &u, &v, &w);
      g[u].push_back(v);
      g[v].push_back(u);
      c[u][v] = w;
   }
   while (findPath())
      enlarge();
   printf("%lld", res);
}
```

convexHull [Viet].cpp

```
#include <iostream>
#include <stack>
#include <stdlib.h>
using namespace std;

struct Point
{
   int x, y;
};

// A globle point needed for  sorting points with reference
// to  the first point Used in compare function of qsort()
Point p0;

// A utility function to find next to top in a stack
Point nextToTop(stack<Point> &S)
{
   Point p = S.top();
   S.pop();
   Point res = S.top();
   S.push(p);
   return res;
}

// A utility function to swap two points
int swap(Point &p1, Point &p2)
{
    Point temp = p1;
    p1 = p2;
    p2 = temp;
}

// A utility function to return square of distance
// between p1 and p2
int distSq(Point p1, Point p2)
{
    return (p1.x - p2.x)*(p1.x - p2.x) +
          (p1.y - p2.y)*(p1.y - p2.y);
}

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise
int orientation(Point p, Point q, Point r)
{
    int val = (q.y - p.y) * (r.x - q.x) -
              (q.x - p.x) * (r.y - q.y);

    if (val == 0) return 0;  // colinear
    return (val > 0)? 1: 2; // clock or counterclock wise
}

// A function used by library function qsort() to sort an array of
// points with respect to the first point
int compare(const void *vp1, const void *vp2)
{
   Point *p1 = (Point *)vp1;
   Point *p2 = (Point *)vp2;

   // Find orientation
   int o = orientation(p0, *p1, *p2);
   if (o == 0)
     return (distSq(p0, *p2) >= distSq(p0, *p1))? -1 : 1;

   return (o == 2)? -1: 1;
}
```

```
// Prints convex hull of a set of n points.
void convexHull(Point points[], int n)
{
   // Find the bottommost point
   int ymin = points[0].y, min = 0;
   for (int i = 1; i < n; i++)
   {
     int y = points[i].y;

     // Pick the bottom-most or chose the left
     // most point in case of tie
     if ((y < ymin) || (ymin == y &&
         points[i].x < points[min].x))
        ymin = points[i].y, min = i;
   }

   // Place the bottom-most point at first position
   swap(points[0], points[min]);

   // Sort n-1 points with respect to the first point.
   // A point p1 comes before p2 in sorted ouput if p2
   // has larger polar angle (in counterclockwise
   // direction) than p1
   p0 = points[0];
   qsort(&points[1], n-1, sizeof(Point), compare);

   // If two or more points make same angle with p0,
   // Remove all but the one that is farthest from p0
   // Remember that, in above sorting, our criteria was
   // to keep the farthest point at the end when more than
   // one points have same angle.
   int m = 1; // Initialize size of modified array
   for (int i=1; i<n; i++)
   {
      // Keep removing i while angle of i and i+1 is same
      // with respect to p0
      while (i < n-1 && orientation(p0, points[i],
                              points[i+1]) == 0)
         i++;


      points[m] = points[i];
```

```
    m++;  // Update size of modified array
  }

  // If modified array of points has less than 3 points,
  // convex hull is not possible
  if (m < 3) return;

  // Create an empty stack and push first three points
  // to it.
  stack<Point> S;
  S.push(points[0]);
  S.push(points[1]);
  S.push(points[2]);

  // Process remaining n-3 points
  for (int i = 3; i < m; i++)
  {
    // Keep removing top while the angle formed by
    // points next-to-top, top, and points[i] makes
    // a non-left turn
    while (orientation(nextToTop(S), S.top(), points[i]) != 2)
      S.pop();
    S.push(points[i]);
  }

  // Now stack has the output points, print contents of stack
  while (!S.empty())
  {
     Point p = S.top();
     cout << "(" << p.x << ", " << p.y <<")" << endl;
     S.pop();
  }
}

// Driver program to test above functions
int main()
{
   Point points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4},
                     {0, 0}, {1, 2}, {3, 1}, {3, 3}};
   int n = sizeof(points)/sizeof(points[0]);
   convexHull(points, n);
   return 0;
}
```

mincost [Viet].cpp

```
#include <bits/stdc++.h>
#define N 502
#define INF 1000001
#define rep(i, a, b) for(int i = a; i <= b; i++)
#define per(i, b, a) for(int i = b; i >= a; i--)
#define bitcount(S) __builtin_popcount(S)
#define ll (long long)
#define db (double)
#define dbg(x) std::cout << x << std::endl
#define sz(x) x.size()
#define fi first
#define se second
using namespace std;
int n, m, k, p[N], res, s, t, d[N];
string st, a[N];
struct edge {
   int v, cap, f, cost, i;
   edge(int _v, int _cap, int _f, int _cost, int _i) {
      v = _v;cap = _cap;f = _f;cost = _cost; i = _i;
   }
};
vector<edge> g[N];
pair<int, int> pre[N];
bool in[N];

void addEdge(int u, int v, int cap, int cost) {
   edge a = edge(v, cap, 0, cost, g[v].size());
   edge b = edge(u, 0, 0, -cost, g[u].size());
   g[u].push_back(a);
   g[v].push_back(b);
}

void buildGraph() {
   rep(i, 0, n)
      addEdge(i, i + 1, k, 0);
   rep(i, 1, m)
      rep(j, 0, n - a[i].length())
```

```
            if (st.substr(j, a[i].length()) == a[i])
                addEdge(j, j + a[i].length(), 1, -p[i]);
    s = 0; t = n + 1;
}

bool fordBellman() {
    queue<int> q;
    rep(i, 1, n + 1) {
        d[i] = INF;
        in[i] = false;
    }
    d[s] = 0;
    q.push(s);
    in[s] = true;
    while (!q.empty()) {
        int u = q.front(); q.pop(); in[u] = false;
        rep(i, 0, g[u].size() - 1) {
            edge e = g[u][i];
            if (e.cap > e.f && d[e.v] > d[u] + e.cost) {
                d[e.v] = d[u] + e.cost;
                pre[e.v] = make_pair(u, i);
                if (!in[e.v]) {
                    q.push(e.v);
                    in[e.v] = true;
                }
            }
        }
    }
    return (d[t] < INF);
}

void enlarge() {
    int v = t, delta = INF;
    int u, i, j;
    while (v != s) {
        u = pre[v].fi;
        i = pre[v].se;
        delta = min(delta, g[u][i].cap - g[u][i].f);
        v = u;
    }
    v = t;
    while (v != s) {
        u = pre[v].fi;
        i = pre[v].se;
        j = g[u][i].i;
        g[u][i].f += delta;
        g[v][j].f -= delta;
        v = u;
    }
    res -= d[t] * delta;
}

int main() {
    //freopen("input.txt", "r", stdin);
    cin >> n >> st >> m;
    rep(i, 1, m)
        cin >> a[i] >> p[i];
    cin >> k;
    buildGraph();
    while (fordBellman())
        enlarge();
    cout << res;
}
```

Treap.cpp

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

struct Treap
{
    #define S(t) (t ? t->Sum : 0)
    #define W(t) (t ? t->Weight : 0)

    struct Node
    {
        long Weight, Sum, Value, AssignValue, IncStart, IncRate;
        int Priority;
        Node *l, *r;

        Node (long x)
        {
            Value = x;
```

```
      Priority = (rand() << 15) | rand();
      l = r = NULL;
      IncRate = IncStart = 0;
      AssignValue = -1;
      Weight = 1;
      Sum = 0;
   }
} *root;

void PushQuery (Node *t)
{
   Node *l = t->l, *r = t->r;

   if (t->AssignValue > -1)
   {
      if (l)
      {
         l->IncRate = l->IncStart = 0;
         l->Value = l->AssignValue = t->AssignValue;
         l->Sum = l->Value * l->Weight;
      }

      if (r)
      {
         r->IncRate = r->IncStart = 0;
         r->Value = r->AssignValue = t->AssignValue;
         r->Sum = r->Value * r->Weight;
      }

      t->AssignValue = -1;
   }

   if (t->IncStart || t->IncRate)
   {
      if (l)
      {
         l->Sum += t->IncStart * l->Weight;
         l->Sum += t->IncRate * ((l->Weight * (l->Weight + 1)) >> 1);
         l->Value += t->IncStart + t->IncRate * (W(l->l) + 1);

         l->IncStart += t->IncStart;
         l->IncRate += t->IncRate;
      }

      if (r)
      {
         long k = t->IncStart + t->IncRate * (W(l) + 1);
         r->Sum += k * r->Weight;
         r->Sum += t->IncRate * ((r->Weight * (r->Weight + 1)) >> 1);
         r->Value += k + t->IncRate * (W(r->l) + 1);

         r->IncStart += k;
         r->IncRate += t->IncRate;
      }

      t->IncRate = t->IncStart = 0;
   }
}

void PropertyUpdate (Node *t)
{
   if (t)
   {
      t->Weight = W(t->l) + W(t->r) + 1;
      t->Sum = S(t->l) + S(t->r) + t->Value;
   }
}

void Split (Node *t, Node *&l, Node *&r, int p)
{
   if (!t) l = r = NULL;
   else
   {
      PushQuery(t);

      if (p <= W(t->l))
      {
         r = t;
         Split(t->l, l, r->l, p);
      } else
      {
         l = t;
         Split(t->r, l->r, r, p - W(t->l) - 1);
```

```
        }
    }

    PropertyUpdate(t);
}

void Merge (Node *&t, Node *l, Node *r)
{
    if (l) PushQuery(l);
    if (r) PushQuery(r);

    if (!l) t = r;
    else
    if (!r) t = l;
    else
    if (l->Priority > r->Priority)
    {
        t = l;
        Merge(t->r, l->r, r);
    } else
    {
        t = r;
        Merge(t->l, l, r->l);
    }

    PropertyUpdate(t);
}

void Assign (int l,int r,long value)
{
    Node *t1, *t = root, *t2;
    Split(t, t, t2, r);
    Split(t, t1, t, l - 1);

    t->IncRate = t->IncStart = 0;
    t->AssignValue = t->Value = value;
    t->Sum = value * t->Weight;

    Merge(t, t1, t);
    Merge(t, t, t2);
    root = t;
}
```

```
    void IncRange (int l,int r,long IncRate)
    {
        Node *t1, *t = root, *t2;
        Split(t, t, t2, r);
        Split(t, t1, t, l - 1);

        t->IncRate += IncRate;
        t->Sum += IncRate * ((t->Weight * (t->Weight + 1)) >> 1);
        t->Value += IncRate * (W(t->l) + 1);

        Merge(t, t1, t);
        Merge(t, t, t2);
        root = t;
    }

    void insert (int p,int v)
    {
        Node *t1, *t2;
        Split(root, t1, t2, p - 1);
        Merge(t1, t1, new Node(v));
        Merge(root, t1, t2);
    }

    long GetSum (int l,int r)
    {
        Node *t1, *t = root, *t2;
        Split(t, t, t2, r);
        Split(t, t1, t, l - 1);

        long s = t->Sum;
        Merge(t, t1, t);
        Merge(t, t, t2);
        root = t;

        return s;
    }

    #undef S
    #undef W
};
```

Math.cpp

```
#include "bits/stdc++.h"

const double PI = acos(-1);

struct point
{
	long x, y;
	point(long x, long y): x(x), y(y) {}
};

// = 0 -> parallel
// > 0 -> left
// < 0 -> right
inline long ccw(point a, point b, point c)
{
	return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
}

long euclid_extended(int nr, int r, int nt, int t)
{
	if (nr == 0) return t;
	return euclid_extended(r - r / nr * nr, nr, t - r / nr * nt, nt);
}

long inverse_mod(int x, int module)
{
	return euclid_extended(x, module, 1, 0);
}
```

Stirling number of the second kind

Number of ways to partition a set of n objects into k non-empty subsets:

$\{\frac{n}{n}\}$ = 1

$\{\frac{n}{1}\}$ = 1

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

Bell numbers

The sum over the values for k of the Stirling numbers of the second kind

$$B_n = \sum_{k=0}^{n}\{\frac{n}{k}\}$$

$$B_{n+1} = \sum_{k=0}^{n}\binom{n}{k}B_k$$

Let $(x)_n = \mathrm{x}(\mathrm{x}-1)(\mathrm{x}-2)\ldots(\mathrm{x}-\mathrm{n}+1)$

$\sum_{k=0}^{n}\{\frac{n}{k}\}(x)_n = x^n$

Recursion:

{(n+1)/k} = k{n/k} + {n/(k-1)}

For k>0: {0/0] = 1 and {n/0} = {0/n} = 0

Catalan number

$$C_n = \frac{(2n)!}{n!\,(n+1)!}$$

Motzkin number

$$M_n = \frac{2n+1}{n+2}M_{n-1} + \frac{3n-3}{n+2}M_{n-2}$$