

# Cache Flow

## Team Reference Document

21/11/2017

## CONTENTS

1.	Acieving AC on a solved problem	1
1.1.	WA	1
1.2.	TLE	1
1.3.	RTE	1
1.4.	MLE	1
2.	Ideas	1
2.1.	A TLE solution is obvious	1
2.2.	Try this on clueless problems	1
3.	Code Templates	1
3.1.	.bashrc	1
3.2.	.vimrc	1
3.3.	Java Template	1
3.4.	Python Template	1
3.5.	C++ Template	2
4.	Data Structures	2
4.1.	Binary Indexed Tree	2
4.2.	Segment Tree	2
4.3.	Lazy Segment Tree	3
4.4.	Union Find	4
4.5.	Monotone Queue	4
4.6.	Treap	4
5.	Graph Algorithms	6
5.1.	Djikstras algorithm	6
5.2.	Bipartite Graphs	6
5.3.	Network Flow	8
5.4.	Min Cost Max Flow	10
6.	Dynamic Programming	11
6.1.	Longest Increasing Subsequence	11
6.2.	String functions	11
6.3.	Josephus problem	11
6.4.	Floyd Warshall	12
7.	Etc	12
7.1.	System of Equations	12
7.2.	Convex Hull	12
7.3.	Number Theory	13
8.	NP tricks	15
8.1.	MaxClique	15

9.	Coordinate Geometry	15
9.1.	Area of a nonintersecting polygon	15
9.2.	Intersection of two lines	15
9.3.	Distance between line segment and point	16
9.4.	Picks theorem	16
9.5.	Implementations	16
10.	Practice Contest Checklist	19

## 1. ACIEVING AC ON A SOLVED PROBLEM

1.1. **WA.**

- Check that minimal input passes.
- Can an int overflow?
- Reread the problem statement.
- Start creating small testcases with python.
- Does cout print with high enough precision?
- Abstract the implementation.

1.2. **TLE.**

- Is the solution sanity checked?
- Use pypy instead of python.
- Rewrite in C++ or Java.
- Can we apply DP anywhere?
- To minimize penalty time you should create a worst case input (if easy) to test on.
- Binary Search over the answer?

1.3. **RTE.**

- Recursion limit in python?
- Arrayindex out of bounds?
- Division by 0?
- Modifying iterator while iterating over it?
- Not using a well defined operator for Collections.sort?
- If nothing makes sense and the end of the contest is approaching you can binary search over where the error is with try-except.

1.4. **MLE.**

- Create objects outside recursive function.
- Rewrite recursive solution to iterative with an own stack.

## 2. IDEAS

## 2.1. A TLE solution is obvious.

- If doing dp, drop parameter and recover from others.
- Use a sorted datastructure.
- Is there a hint in the statement saying that something more is bounded?

## 2.2. Try this on clueless problems.

- Try to interpret problem as a graph (D - NCPC2017).
- Can we apply maxflow, with mincost?
- How does it look for small examples, can we find a pattern?
- Binary search over solution.
- If problem is small, just brute force instead of solving it cleverly. Some times its enough to iterate over the entire domains instead of using xgcd.

## 3. CODE TEMPLATES

## 3.1. .bashrc. Aliases.

```
alias nv=vim
alias o="xdg-open ."
```

## 3.2. .vimrc. Tabs, linenumbers, wrapping

```
set nowrap
syntax on
set tabstop=8 softtabstop=0 shiftwidth=4
set expandtab smarttab
set autoindent smartindent
set rnu number
set scrolloff=8
language en_US
```

## 3.3. Java Template. A Java template.

```
import java.util.*;
import java.io.*;
public class A {
    void solve(BufferedReader in) throws Exception {

    }
    int toInt(String s) {return Integer.parseInt(s);}
    int[] toInts(String s) {
        String[] a = s.split(" ");
        int[] o = new int[a.length];
```

```
        for(int i = 0; i<a.length; i++)
            o[i] = toInt(a[i]);
        return o;
    }
    void e(Object o) {
        System.err.println(o);
    }
    public static void main(String[] args)
        throws Exception {
        BufferedReader in = new BufferedReader
            (new InputStreamReader(System.in));
        (new A()).solve(in);
    }
}
```

## 3.4. Python Template. A Python template

```
from collections import defaultdict
from collections import deque
from collections import Counter
from itertools import permutations #No repeated elements
import sys, bisect
sys.setrecursionlimit(1000000)
# q = deque([0])
# a = q.popleft()
# q.append(0)

# a = [1, 2, 3, 3, 4]
# bisect.bisect(a, 3) == 4
# bisect.bisect_left(a, 3) == 2

# reversed()
# sorted()
```

## 3.5. C++ Template. A C++ template

```
#include <stdio.h>
#include <iostream>
#include <algorithm>
#include <vector>
#include <math.h>
#include <cmath>
```

```
using namespace std;
int main() {
    cout.precision(9);
    int N;
    cin >> N;
    cout << 0 << endl;
}
```

#### 4. DATA STRUCTURES

4.1. **Binary Indexed Tree.** Also called a fenwick tree. Builds in  $\mathcal{O}(n \log n)$  from an array. Query sum from 0 to  $i$  in  $\mathcal{O}(\log n)$  and updates an element in  $\mathcal{O}(\log n)$ .

```
private static class BIT {
    long[] data;
    public BIT(int size) {
        data = new long[size+1];
    }
    public void update(int i, int delta) {
        while(i < data.length) {
            data[i] += delta;
            i += i & -i; // Integer.lowestOneBit(i);
        }
    }
    public long sum(int i) {
        long sum = 0;
        while(i > 0) {
            sum += data[i];
            i -= i & -i;
        }
        return sum;
    }
}
```

4.2. **Segment Tree.** More general than a fenwick tree. Can adapt other operations than sum, e.g. min and max.

```
private static class ST {
    int li, ri;
    int sum; //change to max/min
    ST lN;
    ST rN;
}
```

```
static ST makeSgmTree(int[] A, int l, int r) {
    if(l == r) {
        ST node = new ST();
        node.li = l;
        node.ri = r;
        node.sum = A[l]; //max/min
        return node;
    }
    int mid = (l+r)/2;
    ST lN = makeSgmTree(A, l, mid);
    ST rN = makeSgmTree(A, mid+1, r);
    ST root = new ST();
    root.li = lN.li;
    root.ri = rN.ri;
    root.sum = lN.sum + rN.sum; //max/min
    root.lN = lN;
    root.rN = rN;
    return root;
}
static int getSum(ST root, int l, int r) { //max/min
    if(root.li >= l && root.ri <= r)
        return root.sum; //max/min
    if(root.ri < l || root.li > r)
        return 0; //minInt/maxInt
    else //max/min
        return getSum(root.lN, l, r) + getSum(root.rN, l, r);
}
static int update(ST root, int i, int val) {
    int diff = 0;
    if(root.li == root.ri && i == root.li) {
        diff = val - root.sum; //max/min
        root.sum = val; //max/min
        return diff; //root.max
    }
    int mid = (root.li + root.ri) / 2;
    if(i <= mid) diff = update(root.lN, i, val);
    else diff = update(root.rN, i, val);
    root.sum += diff; //ask other child
    return diff; //and compute max/min
}
```

**4.3. Lazy Segment Tree.** More general implementation of a segment tree where its possible to increase whole segments by some diff, with lazy propagation. Implemented with arrays instead of nodes, which probably has less overhead to write during a competition.

```
class LazySegmentTree {
    private int n;
    private int[] lo, hi, sum, delta;
    public LazySegmentTree(int n) {
        this.n = n;
        lo = new int[4*n + 1];
        hi = new int[4*n + 1];
        sum = new int[4*n + 1];
        delta = new int[4*n + 1];
        init();
    }
    public int sum(int a, int b) {
        return sum(1, a, b);
    }
    private int sum(int i, int a, int b) {
        if(b < lo[i] || a > hi[i]) return 0;
        if(a <= lo[i] && hi[i] <= b) return sum(i);
        prop(i);
        int l = sum(2*i, a, b);
        int r = sum(2*i+1, a, b);
        update(i);
        return l + r;
    }
    public void inc(int a, int b, int v) {
        inc(1, a, b, v);
    }
    private void inc(int i, int a, int b, int v) {
        if(b < lo[i] || a > hi[i]) return;
        if(a <= lo[i] && hi[i] <= b) {
            delta[i] += v;
            return;
        }
        prop(i);
        inc(2*i, a, b, v);
        inc(2*i+1, a, b, v);
    }
}
```

```
    update(i);
}

private void init() {
    init(1, 0, n-1, new int[n]);
}
private void init(int i, int a, int b, int[] v) {
    lo[i] = a;
    hi[i] = b;
    if(a == b) {
        sum[i] = v[a];
        return;
    }
    int m = (a+b)/2;
    init(2*i, a, m, v);
    init(2*i+1, m+1, b, v);
    update(i);
}
private void update(int i) {
    sum[i] = sum(2*i) + sum(2*i+1);
}
private int range(int i) {
    return hi[i] - lo[i] + 1;
}
private int sum(int i) {
    return sum[i] + range(i)*delta[i];
}
private void prop(int i) {
    delta[2*i] += delta[i];
    delta[2*i+1] += delta[i];
    delta[i] = 0;
}
}
```

**4.4. Union Find.** This data structure is used in various algorithms, for example Kruskals algorithm for finding a Minimal Spanning Tree in a weighted graph. Also it can be used for backward simulation of dividing a set.

```
private class Node {
    Node parent;
    int h;
```

```

public Node() {
    parent = this;
    h = 0;
}
public Node find() {
    if(parent != this) parent = parent.find();
    return parent;
}
}
static void union(Node x, Node y) {
    Node xR = x.find(), yR = y.find();
    if(xR == yR) return;
    if(xR.h > yR.h)
        yR.parent = xR;
    else {
        if(yR.h == xR.h) yR.h++;
        xR.parent = yR;
    }
}
}

```

4.5. **Monotone Queue.** Used in sliding window algorithms where one would like to find the minimum in each interval of a given length. Amortized  $\mathcal{O}(n)$  to find min in each of these intervals in an array of length  $n$ . Can easily be used to find the maximum as well.

```

private static class MinMonQue {
    LinkedList<Integer> que = new LinkedList<>();
    public void add(int i) {
        while(!que.isEmpty() && que.getFirst() > i)
            que.removeFirst();
        que.addFirst(i);
    }
    public int last() {
        return que.getLast();
    }
    public void remove(int i) {
        if(que.getLast() == i) que.removeLast();
    }
}

```

4.6. **Treap.** Treap is a binary search tree that uses randomization to balance itself. It's easy to implement, and gives you access to the internal structures of a binary tree,

which can be used to find the  $k$ 'th element for example. Because of the randomness, the average height is about a factor 4 of a perfectly balanced tree.

```

class Treap{
    int sz;
    int v;
    double y;
    Treap L, R;

    static int sz(Treap t) {
        if(t == null) return 0;
        return t.sz;
    }
    static void update(Treap t) {
        if(t == null) return;
        t.sz = sz(t.L) + sz(t.R) + 1;
    }
    static Treap merge(Treap a, Treap b) {
        if (a == null) return b;
        if(b == null) return a;
        if (a.y < b.y) {
            a.R = merge(a.R, b);
            update(a);
            return a;
        } else {
            b.L = merge(a, b.L);
            update(b);
            return b;
        }
    }
    //inserts middle in left half
    static Treap[] split(Treap t, int x) {
        if (t == null) return new Treap[2];
        if (t.v <= x) {
            Treap[] p = split(t.R, x);
            t.R = p[0];
            p[0] = t;
            return p;
        } else {
            Treap[] p = split(t.L, x);
            t.L = p[1];

```

```

    p[1] = t;
    return p;
}
}
//use only with split
static Treap insert(Treap t, int x) {
    Treap m = new Treap();
    m.v = x;
    m.y = Math.random();
    m.sz = 1;
    Treap[] p = splitK(t, x-1);
    return merge(merge(p[0],m), p[1]);
}

//inserts middle in left half
static Treap[] splitK(Treap t, int x) {
    if (t == null) return new Treap[2];
    if (t.sz < x) return new Treap[]{t, null};
    if (sz(t.L) >= x) {
        Treap[] p = splitK(t.L, x);
        t.L = p[1];
        p[1] = t;
        update(p[0]);
        update(p[1]);
        return p;
    } else if (sz(t.L) + 1 == x){
        Treap r = t.R;
        t.R = null;
        Treap[] p = new Treap[]{t, r};
        update(p[0]);
        update(p[1]);
        return p;
    } else {
        Treap[] p = splitK(t.R, x - sz(t.L)-1);
        t.R = p[0];
        p[0] = t;
        update(p[0]);
        update(p[1]);
        return p;
    }
}

```

```

    }
}
//use only with splitK
static Treap insertK(Treap t, int w, int x) {
    Treap m = new Treap();
    m.v = x;
    m.y = Math.random();
    m.sz = 1;
    Treap[] p = splitK(t, w);
    t = merge(p[0], m);
    return merge(t, p[1]);
}
//use only with splitK
static Treap deleteK(Treap t, int w, int x) {
    Treap[] p = splitK(t, w);
    Treap[] q = splitK(p[0], w-1);
    return merge(q[0], p[1]);
}

static Treap Left(Treap t) {
    if (t == null) return null;
    if (t.L == null) return t;
    return Left(t.L);
}
static Treap Right(Treap t) {
    if (t == null) return null;
    if (t.R == null) return t;
    return Right(t.R);
}
}

```

## 5. GRAPH ALGORITHMS

5.1. **Djikstras algorithm.** Finds the shortest distance between two Nodes in a weighted graph in  $\mathcal{O}(|E| \log |V|)$  time.

```

//Requires java.util.LinkedList and java.util.TreeSet
private static class Node implements Comparable<Node>{
    LinkedList<Edge> edges = new LinkedList<>();
    int w;
}

```

```

    int id;
    public Node(int id) {
        w = Integer.MAX_VALUE;
        this.id = id;
    }
    public int compareTo(Node n) {
        if(w != n.w) return w - n.w;
        return id - n.id;
    }
    //Assumes all nodes have weight MAXINT.
    public int djikstra(Node x) {
        this.w = 0;
        TreeSet<Node> set = new TreeSet<>();
        set.add(this);
        while(!set.isEmpty()) {
            Node curr = set.pollFirst();
            if(x == curr) return x.w;
            for(Edge e: curr.edges) {
                Node other = e.u == curr? e.v : e.u;
                if(other.w > e.cost + curr.w) {
                    set.remove(other);
                    other.w = e.cost + curr.w;
                    set.add(other);
                }
            }
        }
        return -1;
    }
}

private static class Edge {
    Node u, v;
    int cost;
    public Edge(Node u, Node v, int c) {
        this.u = u; this.v = v;
        cost = c;
    }
}

```

**5.2. Bipartite Graphs.** The Hopcroft-Karp algorithm finds the maximal matching in a bipartite graph. Also, this matching can together with Königs theorem be used

to construct a minimal vertex-cover, which as we all know is the complement of a maximum independent set. Runs in  $\mathcal{O}(|E|\sqrt{|V|})$ .

```

import java.util.*;
class Node {
    int id;
    LinkedList<Node> ch = new LinkedList<>();
    public Node(int id) {
        this.id = id;
    }
}

public class BiGraph {
    private static int INF = Integer.MAX_VALUE;
    LinkedList<Node> L, R;
    int N, M;
    Node[] U;
    int[] Pair, Dist;
    int nild;
    public BiGraph(LinkedList<Node> L, LinkedList<Node> R){
        N = L.size(); M = R.size();
        this.L = L; this.R = R;
        U = new Node[N+M];
        for(Node n: L) U[n.id] = n;
        for(Node n: R) U[n.id] = n;
    }
    private boolean bfs() {
        LinkedList<Node> Q = new LinkedList<>();
        for(Node n: L)
            if(Pair[n.id] == -1) {
                Dist[n.id] = 0;
                Q.add(n);
            }else
                Dist[n.id] = INF;

        nild = INF;
        while(!Q.isEmpty()) {
            Node u = Q.removeFirst();
            if(Dist[u.id] < nild)
                for(Node v: u.ch) if(Dist[v.id] == INF){
                    if(Pair[v.id] == -1)
                        nild = Dist[u.id] + 1;

```

```

        else {
            Dist[Pair[v.id]] = Dist[u.id] + 1;
            Q.addLast(U[Pair[v.id]]);
        }
    }
    return nild != INF;
}
private int distp(Node v) {
    if(Pair[v.id] == -1) return nild;
    return Dist[Pair[v.id]];
}
private boolean dfs(Node u) {
    for(Node v: u.ch) if(distp(v) == Dist[u.id] + 1) {
        if(Pair[v.id] == -1 || dfs(U[Pair[v.id]])) {
            Pair[v.id] = u.id;
            Pair[u.id] = v.id;
            return true;
        }
    }
    Dist[u.id] = INF;
    return false;
}
public HashMap<Integer, Integer> maxMatch() {
    Pair = new int[M+N];
    Dist = new int[M+N];
    for(int i = 0; i < M+N; i++) {
        Pair[i] = -1;
        Dist[i] = INF;
    }
    HashMap<Integer, Integer> out = new HashMap<>();
    while(bfs()) {
        for(Node n: L) if(Pair[n.id] == -1)
            dfs(n);
    }
    for(Node n: L) if(Pair[n.id] != -1)
        out.put(n.id, Pair[n.id]);
    return out;
}
public HashSet<Integer> minVTC() {

```

```

    HashMap<Integer, Integer> Lm = maxMatch();
    HashMap<Integer, Integer> Rm = new HashMap<>();
    for(int x: Lm.keySet()) Rm.put(Lm.get(x), x);
    boolean[] Z = new boolean[M+N];
    LinkedList<Node> bfs = new LinkedList<>();
    for(Node n: L) {
        if(!Lm.containsKey(n.id)) {
            Z[n.id] = true;
            bfs.add(n);
        }
    }
    while(!bfs.isEmpty()) {
        Node x = bfs.removeFirst();
        int nono = -1;
        if(Lm.containsKey(x.id))
            nono = Lm.get(x.id);
        for(Node y: x.ch) {
            if(y.id == nono || Z[y.id]) continue;
            Z[y.id] = true;
            if(Rm.containsKey(y.id)){
                int xx = Rm.get(y.id);
                if(!Z[xx]) {
                    Z[xx] = true;
                    bfs.addLast(U[xx]);
                }
            }
        }
    }
    HashSet<Integer> K = new HashSet<>();
    for(Node n: L) if(!Z[n.id]) K.add(n.id);
    for(Node n: R) if(Z[n.id]) K.add(n.id);
    return K;
}
}

```

**5.3. Network Flow.** The Floyd Warshall algorithm for determining the maximum flow through a graph can be used for a lot of unexpected problems. Given a problem that can be formulated as a graph, where no ideas are found trying, it might help trying to apply network flow. The running time is  $\mathcal{O}(C \cdot m)$  where  $C$  is the maximum flow and  $m$  is the amount of edges in the graph. If  $C$  is very large we can change the



running time to  $\mathcal{O}(\log Cm^2)$  by only studying edges with a large enough capacity in the beginning.

```
import java.util.*;
class Node {
    LinkedList<Edge> edges = new LinkedList<>();
    int id;
    boolean visited = false;
    Edge last = null;
    public Node(int id) {
        this.id = id;
    }
    public void append(Edge e) {
        edges.add(e);
    }
}
class Edge {
    Node source, sink;
    int cap;
    int id;
    Edge redge;
    public Edge(Node u, Node v, int w, int id){
        source = u; sink = v;
        cap = w;
        this.id = id;
    }
}
class FlowNetwork {
    Node[] adj;
    int edgect = 0;
    HashMap<Integer,Integer> flow = new HashMap<>();
    ArrayList<Edge> real = new ArrayList<Edge>();
    public FlowNetwork(int size) {
        adj = new Node[size];
        for(int i = 0; i<size; i++) {
            adj[i] = new Node(i);
        }
    }
    void add_edge(int u, int v, int w, int id){
        Node Nu = adj[u], Nv = adj[v];
        Edge edge = new Edge(Nu, Nv, w, edgect++);
```

```
        Edge redge = new Edge(Nv, Nu, 0, edgect++);
        edge.redge = redge;
        redge.redge = edge;
        real.add(edge);
        adj[u].append(edge);
        adj[v].append(redge);
        flow.put(edge.id, 0);
        flow.put(redge.id, 0);
    }

    void reset() {
        for(int i = 0; i<adj.length; i++) {
            adj[i].visited = false; adj[i].last = null;
        }
    }

    LinkedList<Edge> find_path(Node s, Node t,
        List<Edge> path){
        reset();
        LinkedList<Node> active = new LinkedList<>();
        active.add(s);
        while(!active.isEmpty() && !t.visited) {
            Node now = active.pollFirst();
            for(Edge e: now.edges) {
                int residual = e.cap - flow.get(e.id);
                if(residual>0 && !e.sink.visited) {
                    e.sink.visited = true;
                    e.sink.last = e;
                    active.addLast(e.sink);
                }
            }
        }
        if(t.visited) {
            LinkedList<Edge> res = new LinkedList<>();
            Node curr = t;
            while(curr != s) {
                res.addFirst(curr.last);
                curr = curr.last.sink;
            }
            return res;
        }
    }
}
```

```

    } else return null;
}

int max_flow(int s, int t) {
    Node source = adj[s];
    Node sink = adj[t];
    LinkedList<Edge> path = find_path(source, sink,
        new LinkedList<Edge>());
    while (path != null) {
        int min = Integer.MAX_VALUE;
        for (Edge e : path) {
            min = Math.min(min, e.cap - flow.get(e.id));
        }
        for (Edge e : path) {
            flow.put(e.id, flow.get(e.id) + min);
            Edge r = e.redge;
            flow.put(r.id, flow.get(r.id) - min);
        }
        path = find_path(source, sink,
            new LinkedList<Edge>());
    }
    int sum = 0;
    for (Edge e : source.edges) {
        sum += flow.get(e.id);
    }
    return sum;
}

LinkedList<Edge> min_cut(int s, int t) {
    HashSet<Node> A = new HashSet<>();
    LinkedList<Node> bfs = new LinkedList<>();
    bfs.add(adj[s]);
    A.add(adj[s]);
    while (!bfs.isEmpty()) {
        Node i = bfs.removeFirst();
        for (Edge e : i.edges) {
            int c = e.cap - flow.get(e.id);
            if (c > 0 && !A.contains(e.sink)) {
                bfs.add(e.sink);
                A.add(e.sink);
            }
        }
    }
}

```

```

        if (e.sink.id == t) return null;
    }
}

LinkedList<Edge> out = new LinkedList<>();
for (Node n : A) for (Edge e : n.edges)
    if (!A.contains(e.sink) && e.cap != 0)
        out.add(e);
return out;
}
}

```

5.4. **Min Cost Max Flow.** Finds the minimal cost of a maximum flow through a graph. Can be used for some optimization problems where the optimal assignment needs to be a maximum flow.

```

class MinCostMaxFlow {
    boolean found[];
    int N, dad[];
    long cap[][][], flow[][][], cost[][][], dist[], pi[];

    static final long INF = Long.MAX_VALUE / 2 - 1;

    boolean search(int s, int t) {
        Arrays.fill(found, false);
        Arrays.fill(dist, INF);
        dist[s] = 0;

        while (s != N) {
            int best = N;
            found[s] = true;
            for (int k = 0; k < N; k++) {
                if (found[k]) continue;
                if (flow[k][s] != 0) {
                    long val = dist[s] + pi[s] - pi[k] - cost[k][s];
                    if (dist[k] > val) {
                        dist[k] = val;
                        dad[k] = s;
                    }
                }
            }
            if (flow[s][k] < cap[s][k]) {

```

```

    long val = dist[s] + pi[s] - pi[k] + cost[s][k];
    if (dist[k] > val) {
        dist[k] = val;
        dad[k] = s;
    }
}

if (dist[k] < dist[best]) best = k;
}
s = best;
}
for (int k = 0; k < N; k++)
    pi[k] = Math.min(pi[k] + dist[k], INF);
return found[t];
}

long[] mcmf(long c[][], long d[][], int s, int t) {
    cap = c;
    cost = d;

    N = cap.length;
    found = new boolean[N];
    flow = new long[N][N];
    dist = new long[N+1];
    dad = new int[N];
    pi = new long[N];

    long totflow = 0, totcost = 0;
    while (search(s, t)) {
        long amt = INF;
        for (int x = t; x != s; x = dad[x])
            amt = Math.min(amt, flow[x][dad[x]] != 0 ?
                flow[x][dad[x]] : cap[dad[x]][x] - flow[dad[x]][x]);
        for (int x = t; x != s; x = dad[x]) {
            if (flow[x][dad[x]] != 0) {
                flow[x][dad[x]] -= amt;
                totcost -= amt * cost[x][dad[x]];
            } else {
                flow[dad[x]][x] += amt;
                totcost += amt * cost[dad[x]][x];
            }
        }
    }
}

```

```

    }
}
totflow += amt;
}

return new long[]{ totflow, totcost };
}
}

```

## 6. DYNAMIC PROGRAMMING

**6.1. Longest Increasing Subsequence.** Finds the longest increasing subsequence in an array in  $\mathcal{O}(n \log n)$  time. Can easily be transformed to longest decreasing/nondecreasing/nonincreasing subsequence.

```

public static int lis(int[] X) {
    int n = X.length;
    int P[] = new int[n];
    int M[] = new int[n+1];
    int L = 0;
    for (int i = 0; i < n; i++) {
        int lo = 1;
        int hi = L;
        while (lo <= hi) {
            int mid = lo + (hi - lo + 1) / 2;
            if (X[M[mid]] < X[i])
                lo = mid + 1;
            else
                hi = mid - 1;
        }
        int newL = lo;
        P[i] = M[newL - 1];
        M[newL] = i;
        if (newL > L)
            L = newL;
    }
    int[] S = new int[L];
    int k = M[L];
    for (int i = L - 1; i >= 0; i--) {
        S[i] = k; //or X[k]
    }
}

```

```

    k = P[k];
}
return L; // or S
}

```

**6.2. String functions.** The z-function computes the longest common prefix of  $t$  and  $t[i:]$  for each  $i$  in  $\mathcal{O}(|t|)$ . The border function computes the longest common proper (smaller than whole string) prefix and suffix of string  $t[:i]$ .

```

def zfun(t):
    z = [0]*len(t)
    n = len(t)
    l, r = (0,0)
    for i in range(1,n):
        if i < r:
            z[i] = min(z[i-l], r-i+1)
        while z[i] + i < n and t[i+z[i]] == t[z[i]]:
            z[i]+=1
        if i + z[i] - 1 > r:
            l = i
            r = i + z[i] - 1
    return z

def matches(t, p):
    s = p + '#' + t
    return filter(lambda x: x[1] == len(p),
                  enumerate(zfun(s)))

def borders(s):
    b = [0]*len(s)
    for i in range(1, len(s)):
        k = b[i-1]
        while k>0 and s[k] != s[i]:
            k = b[k-1]
        if s[k] == s[i]:
            b[i] = k+1
    return b

```

**6.3. Josephus problem.** Who is the last one to get removed from a circle if the  $k$ 'th element is continuously removed?

```

# Rewritten from J(n, k) = (J(n-1, k) + k)%n
def J(n, k):
    r = 0
    for i in range(2, n+1):
        r = (r + k)%i
    return r

```

**6.4. Floyd Warshall.** Constructs a matrix with the distance between all pairs of nodes in  $\mathcal{O}(n^3)$  time. Works for negative edge weights, but not if there exists negative cycles. The next matrix is used to reconstruct a path. Can be skipped if we don't care about the path.

```

n, m = map(int, input().split())
INF = 1000000000
dist = [[INF]*n for _ in range(n)]
nxt = [[None]*n for _ in range(n)]
edges = [tuple(map(int, input().split())) for _ in range(m)]
for e in edges:
    dist[e[0]][e[1]] = e[2]
    nxt[e[0]][e[1]] = e[1]
for k in range(n):
    for i in range(n):
        for j in range(n):
            if dist[i][j] > dist[i][k] + dist[k][j]:
                dist[i][j] = dist[i][k] + dist[k][j]
                nxt[i][j] = nxt[i][k]

def path(i, j):
    if nxt[i][j] == None: return []
    path = [i]
    while i != j:
        i = nxt[i][j]
        path.append(i)
    return path

```

7. ETC

**7.1. System of Equations.** Solves the system of equations  $Ax = b$  by Gaussian elimination. This can for example be used to determine the expected value of each node in a markov chain. Runs in  $\mathcal{O}(N^3)$ .

```

//Computes A^-1 * b
static double[] solve(double[][] A, double[] b) {
    int N = b.length;

```

```

// Gaussian elimination with partial pivoting
for (int i = 0; i < N; i++) {
    // find pivot row and swap
    int max = i;
    for (int j = i + 1; j < N; j++)
        if (Math.abs(A[j][i]) > Math.abs(A[max][i]))
            max = j;
    double[] tmp = A[i];
    A[i] = A[max];
    A[max] = tmp;
    double tmp2 = b[i];
    b[i] = b[max];
    b[max] = tmp2;
    // A doesn't have full rank
    if (Math.abs(A[i][i]) < 0.00001) return null;
    // pivot within b
    for (int j = i + 1; j < N; j++)
        b[j] -= b[i] * A[j][i] / A[i][i];
    // pivot within A
    for (int j = i + 1; j < N; j++) {
        double m = A[j][i] / A[i][i];
        for (int k = i + 1; k < N; k++)
            A[j][k] -= A[i][k] * m;
        A[j][i] = 0.0;
    }
}
// back substitution
double[] x = new double[N];
for (int j = N - 1; j >= 0; j--) {
    double t = 0.0;
    for (int k = j + 1; k < N; k++)
        t += A[j][k] * x[k];
    x[j] = (b[j] - t) / A[j][j];
}
return x;
}

```

**7.2. Convex Hull.** From a collection of points in the plane the convex hull is often used to compute the largest distance or the area covered, or the length of a rope that

encloses the points. It can be found in  $\mathcal{O}(N \log N)$  time by sorting the points on angle and the sweeping over all of them.

```

import java.util.*;
public class ConvexHull {
    static class Point implements Comparable<Point> {
        static Point xmin;
        int x, y;
        public Point(int x, int y) {
            this.x = x; this.y = y;
        }
        public int compareTo(Point p) {
            int c = cross(this, xmin, p);
            if (c != 0) return c;
            double d = dist(this, xmin) - dist(p, xmin);
            return (int) Math.signum(d);
        }
    }
    static double dist(Point p1, Point p2) {
        return Math.hypot(p1.x - p2.x, p1.y - p2.y);
    }
    static int cross(Point a, Point b, Point c) {
        int dx1 = b.x - a.x;
        int dy1 = b.y - a.y;
        int dx2 = c.x - b.x;
        int dy2 = c.y - b.y;
        return dx1*dy2 - dx2*dy1;
    }
    Point[] convexHull(Point[] S) {
        int N = S.length;
        // find a point on the convex hull.
        Point xmin = S[0];
        int id = 0;
        for (int i = 0; i < N; i++) {
            Point p = S[i];
            if (xmin.x > p.x ||
                xmin.x == p.x && xmin.y > p.y) {
                xmin = p;
                id = i;
            }
        }
    }
}

```

```

S[id] = S[N-1];
S[N-1] = xmin;
Point.xmin = xmin;
// Sort on angle to xmin.
Arrays.sort(S, 0, N-1);
Point[] H = new Point[N+1];
H[0] = S[N-2];
H[1] = xmin;
for(int i = 0; i<N-1; i++)
    H[i+2] = S[i];
int M = 1;
// swipe over the points
for(int i = 2; i<=N; i++) {
    while(cross(H[M-1],H[M],H[i]) <= 0) {
        if(M>1)
            M--;
        else if (i == N)
            break;
        else
            i += 1;
    }
    M+=1;
    Point tmp = H[M];
    H[M] = H[i];
    H[i] = tmp;
}
Point[] Hull = new Point[M];
for(int i = 0; i<M; i++)
    Hull[i] = H[i];
return Hull;
}
}

```

### 7.3. Number Theory.

```

def gcd(a, b):
    return b if a%b == 0 else gcd(b, a%b)

# returns g = gcd(a, b), x0, y0,
# where g = x0*a + y0*b
def xgcd(a, b):

```

```

    x0, x1, y0, y1 = 1, 0, 0, 1
    while b != 0:
        q, a, b = (a // b, b, a % b)
        x0, x1 = (x1, x0 - q * x1)
        y0, y1 = (y1, y0 - q * y1)
    return (a, x0, y0)

def crt(la, ln):
    assert len(la) == len(ln)
    for i in range(len(la)):
        assert 0 <= la[i] < ln[i]
    prod = 1
    for n in ln:
        assert gcd(prod, n) == 1
        prod *= n
    lN = []
    for n in ln:
        lN.append(prod//n)
    x = 0
    for i, a in enumerate(la):
        print(lN[i], ln[i])
        _, Mi, mi = xgcd(lN[i], ln[i])
        x += a*Mi*lN[i]
    return x % prod

# finds x^e mod m
def modpow(x, m, e):
    res = 1
    while e:
        if e%2 == 1:
            res = (res*x) % m
        x = (x*x) % m
        e = e//2
    return res

# Divides a list of digits with an int.
# A lot faster than using bigint-division.
def div(L, d):
    r = [0]*(len(L) + 1)
    q = [0]*len(L)

```

```

for i in range(len(L)):
    x = int(L[i]) + r[i]*10
    q[i] = x//d
    r[i+1] = x-q[i]*d
s = []
for i in range(len(L) - 1, 0, -1):
    s.append(q[i]%10)
    q[i-1] += q[i]//10

while q[0]:
    s.append(q[0]%10)
    q[0] = q[0]//10
s = s[::-1]
i = 0
while s[i] == 0:
    i += 1
return s[i:]

```

*# Multiplies a list of digits with an int.  
# A lot faster than using bigint-multiplication.*

```

def mul(L, d):
    r = [d*x for x in L]
    s = []
    for i in range(len(r) - 1, 0, -1):
        s.append(r[i]%10)
        r[i-1] += r[i]//10
    while r[0]:
        s.append(r[0]%10)
        r[0] = r[0]//10
    return s[::-1]

```

```

large_primes = [
5915587277,
1500450271,
3267000013,
5754853343,
4093082899,
9576890767,
3628273133,

```

```

2860486313,
5463458053,
3367900313
]

```

## 8. NP TRICKS

**8.1. MaxClique.** The max clique problem is one of Karp's 21 NP-complete problems. The problem is to find the largest subset of an undirected graph that forms a clique - a complete graph. There is an obvious algorithm that just inspects every subset of the graph and determines if this subset is a clique. This algorithm runs in  $\mathcal{O}(n^2 2^n)$ . However one can use the meet in the middle trick (one step divide and conquer) and reduce the complexity to  $\mathcal{O}(n^2 2^{\frac{n}{2}})$ .

```

static int max_clique(int n, int[][] adj) {
    int fst = n/2;
    int snd = n - fst;
    int[] maxc = new int[1<<fst];
    int max = 1;
    for(int i = 0; i<(1<<fst); i++) {
        for(int a = 0; a<fst; a++) {
            if((i&1<<a) != 0)
                maxc[i] = Math.max(maxc[i], maxc[i^(1<<a)]);
        }
        boolean ok = true;
        for(int a = 0; a<fst; a++) if((i&1<<a) != 0) {
            for(int b = a+1; b<fst; b++) {
                if((i&1<<b) != 0 && adj[a][b] == 0)
                    ok = false;
            }
        }
        if(ok) {
            maxc[i] = Integer.bitCount(i);
            max = Math.max(max, maxc[i]);
        }
    }
    for(int i = 0; i<(1<<snd); i++) {
        boolean ok = true;
        for(int a = 0; a<snd; a++) if((i&1<<a) != 0) {
            for(int b = a+1; b<snd; b++) {
                if((i&1<<b) != 0)

```

```

        if(adj[a+fst][b+fst] == 0)
            ok = false;
    }
}
if(!ok) continue;
int mask = 0;
for(int a = 0; a<fst; a++) {
    ok = true;
    for(int b = 0; b<snd; b++) {
        if((i&1<<b) != 0) {
            if(adj[a][b+fst] == 0) ok = false;
        }
    }
    if(ok) mask |= (1<<a);
}
max = Math.max(Integer.bitCount(i) + maxc[mask],
                max);
}
return max;
}

```

## 9. COORDINATE GEOMETRY

**9.1. Area of a nonintersecting polygon.** The signed area of a polygon with  $n$  vertices is given by

$$A = \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i)$$

**9.2. Intersection of two lines.** Two lines defined by

$$a_1 x + b_1 y + c_1 = 0$$

$$a_2 x + b_2 y + c_2 = 0$$

Intersects in the point

$$P = \left( \frac{b_1 c_2 - b_2 c_1}{w}, \frac{a_2 c_1 - a_1 c_2}{w} \right),$$

where  $w = a_1 b_2 - a_2 b_1$ . If  $w = 0$  the lines are parallell.

**9.3. Distance between line segment and point.** Given a linesegment between point  $P, Q$ , the distance  $D$  to point  $R$  is given by:

$$\begin{aligned}
 a &= Q_y - P_y \\
 b &= Q_x - P_x \\
 c &= P_x Q_y - P_y Q_x \\
 R_P &= \left( \frac{b(bR_x - aR_y) - ac}{a^2 + b^2}, \frac{a(aR_y - bR_x) - bc}{a^2 + b^2} \right) \\
 D &= \begin{cases} \frac{|aR_x + bR_y + c|}{\sqrt{a^2 + b^2}} & \text{if } (R_{P_x} - P_x)(R_{P_x} - Q_x) < 0, \\ \min |P - R|, |Q - R| & \text{otherwise} \end{cases}
 \end{aligned}$$

**9.4. Picks theorem.** Find the amount of internal integer coordinates  $i$  inside a polygon with picks theorem  $A = \frac{b}{2} + i - 1$ , where  $A$  is the area of the polygon and  $b$  is the amount of coordinates on the boundary.

**9.5. Implementations.**

```
import math
```

```
# Distance between two points
```

```
def dist(p, q):
    return math.hypot(p[0]-q[0], p[1] - q[1])
```

```
# Converts two points to a line (a, b, c),
```

```
# ax + by + c = 0
```

```
# if p == q, a = b = c = 0
```

```
def pts2line(p, q):
```

```
    return (-q[1] + p[1],
            q[0] - p[0],
            p[0]*q[1] - p[1]*q[0])
```

```
# Distance from a point to a line,
```

```
# given that a != 0 or b != 0
```

```
def distl(l, p):
    return (abs(l[0]*p[0] + l[1]*p[1] + l[2])
            /math.hypot(l[0], l[1]))
```

```
# intersects two lines.
```

```
# if parallell, returns False.
```

```
def inters(l1, l2):
```



```

a1,b1,c1 = l1
a2,b2,c2 = l2
cp = a1*b2 - a2*b1
if cp != 0:
    return ((b1*c2 - b2*c1)/cp, (a2*c1 - a1*c2)/cp)
else:
    return False

# projects a point on a line
def project(l, p):
    a, b, c = l
    return ((b*(b*p[0] - a*p[1]) - a*c)/(a*a + b*b),
            (a*(a*p[1] - b*p[0]) - b*c)/(a*a + b*b))

# Finds the distance between a point, and
# the Segment AB, the Ray AB and the line AB.
# (distSeg, distHalfinf, distLine)
def distSegP(P, Q, R):
    a, b, c = pts2line(P, Q)
    Rpx, Rpy = project((a,b,c), R)
    dp = min(dist(P, R), dist(Q, R))
    dl = distl((a,b,c), R)
    if (inside(Rpx, P[0], Q[0]) and
        inside(Rpy, P[1], Q[1])):
        return (dl, dl, dl)
    if insideH((Rpx, Rpy), P, Q):
        return (dp, dl, dl)
    return (dp, dp, dl)

# Finds if A <= i <= B.
def inside(i, A, B):
    return (i-A)*(i-B) <= 0

# Finds if a point i on the line AB
# is on the segment AB.
def insideS(i, A, B):
    return (inside(i[0], A[0], B[0])
            and inside(i[1], A[1], B[1]))

# Finds if a point i on the line AB

```

```

# is on the ray AB.
def insideH(i, A, B):
    return ((i[0] - A[0])*(A[0] - B[0]) <= 0
            and (i[1] - A[1])*(A[1] - B[1]) <= 0)

# Finds if segment AB and CD overlaps.
def overlap(A, B, C, D):
    if A[0] == B[0]:
        return __overlap(A[1], B[1], C[1], D[1])
    else:
        return __overlap(A[0], B[0], C[0], D[0])

# Helper functions
def __overlap(x1, x2, x3, x4):
    x1, x2 = (min(x1,x2), max(x1, x2))
    x3, x4 = (min(x3,x4), max(x3, x4))
    return x2 >= x3 and x1 <= x4

# prints a point
def p(P):
    print(str(P[0]) + ' ' + str(P[1]))

# prints common points between
# two segments AB and CD.
def SegSeg(A, B, C, D):
    eqa = A == B
    eqc = C == D
    if eqa and eqc:
        if A == C:
            p(A)
            return True
        return False
    if eqc:
        eqa, A, B, C, D = (eqc, C, D, A, B)
    if eqa:
        l = pts2line(C, D)
        if (l[0]*A[0] + l[1]*A[1] + l[2] == 0 and
            inside(A[0], C[0], D[0]) and
            inside(A[1], C[1], D[1])):
            p(A)
            return True

```

```

    return False

A, B = tuple(sorted([A,B]))
C, D = tuple(sorted([C,D]))
l1 = pts2line(A, B)
l2 = pts2line(C, D)
if l1[0]*l2[1] == l1[1]*l2[0]:
    if l1[0]*l2[2] == l1[2]*l2[0]:
        if overlap(A, B, C, D):
            if B == C:
                p(B)
                return True
            if D == A:
                p(A)
                return True
            s = sorted([A,B,C,D])
            p(s[1])
            p(s[2])
            return True
        else:
            return False
    else:
        return False
ix, iy = inters(l1, l2)
if (inside(ix, A[0], B[0]) and
    inside(iy, A[1], B[1]) and
    inside(ix, C[0], D[0]) and
    inside(iy, C[1], D[1])):
    p((ix, iy))
    return True
return False

# Intersections between circles
def intersections(c1, c2):
    x1, y1, r1 = c1
    x2, y2, r2 = c2
    if x1 == x2 and y1 == y2 and r1 == r2:
        return False
    if r1 > r2:
        x1, y1, r1, x2, y2, r2 = (x2, y2, r2, x1, y1, r1)

```

```

    dist2 = (x1 - x2)*(x1-x2) + (y1 - y2)*(y1 - y2)
    rsq = (r1 + r2)*(r1 + r2)
    if dist2 > rsq or dist2 < (r1-r2)*(r1-r2):
        return []
    elif dist2 == rsq:
        cx = x1 + (x2-x1)*r1/(r1+r2)
        cy = y1 + (y2-y1)*r1/(r1+r2)
        return [(cx, cy)]
    elif dist2 == (r1-r2)*(r1-r2):
        cx = x1 - (x2-x1)*r1/(r2-r1)
        cy = y1 - (y2-y1)*r1/(r2-r1)
        return [(cx, cy)]

    d = math.sqrt(dist2)
    f = (r1*r1 - r2*r2 + dist2)/(2*dist2)
    xf = x1 + f*(x2-x1)
    yf = y1 + f*(y2-y1)
    dx = xf-x1
    dy = yf-y1
    h = math.sqrt(r1*r1 - dx*dx - dy*dy)
    norm = abs(math.hypot(dx, dy))
    p1 = (xf + h*(-dy)/norm, yf + h*(dx)/norm)
    p2 = (xf + h*(dy)/norm, yf + h*(-dx)/norm)
    return sorted([p1, p2])

```

*# Finds the bisector through origo  
# between two points by normalizing.*

```

def bisector(p1, p2):
    d1 = math.hypot(p1[0], p2[1])
    d2 = math.hypot(p2[0], p2[1])
    return ((p1[0]/d1 + p2[0]/d2),
            (p1[1]/d1 + p2[1]/d2))

```

*# Distance from P to origo*

```

def norm(P):
    return (P[0]**2 + P[1]**2 + P[2]**2)**(0.5)

```

*# Finds distance between point p  
# and line A + t\*u in 3D*

```

def dist3D(A, u, p):

```

---

```
AP = tuple(A[i] - p[i] for i in range(3))
cross = tuple(AP[i]*u[(i+1)%3] - AP[(i+1)%3]*u[i]
             for i in range(3))
return norm(cross)/norm(u)
```

---

10. PRACTICE CONTEST CHECKLIST

- Operations per second in py2
  - Operations per second in py3
  - Operations per second in java
  - Operations per second in c++
  - Operations per second on local machine
  - Is MLE called MLE or RTE?
  - What happens if extra output is added? What about one extra new line or space?
  - Look at documentation on judge.
  - Submit a clar.
  - Print a file.
  - Directory with test cases.
-