# ACM-ICPC Team Reference Document

## University of Massachusetts Boston

## Contents

# 1 Graphs

## 1.1 Max Flow

```
int res[MAX_V][MAX_V], mf, f, s, t; // global variables
      vi p; // p stores the BFS spanning tree from s
      void augment(int v, int minEdge) { // traverse BFS spanning tree from s->t
            if (v == s) {
                  f = minEdge;
                  return;
            } // record minEdge in a global var f
            else if (p[v] != -1) {
                  augment(p[v], min(minEdge, res[p[v]][v]));
                  res[p[v]][v] -= f; res[v][p[v]] += f;
            }
      }
      // inside int main(): set up 'res', 's', and 't' with appropriate values
      mf = 0; // mf stands for max_flow
      while (1) { // O(VE^2) (actually O(V^3 E) Edmonds Karp's algorithm
            f = 0;
            // run BFS, compare with the original BFS shown in Section 4.2.2
            vi dist(MAX_V, INF); dist[s] = 0; queue<int> q; q.push(s);
            p.assign(MAX_V, -1); // record the BFS spanning tree, from s to t!
            while (!q.empty()) {
                  int u = q.front(); q.pop();
                  if (u == t) break; // immediately stop BFS if we already reach sink t
                  for (int v = 0; v < MAX_V; v++) // note: this part is slow
                        if (res[u][v] > 0 && dist[v] == INF)
                  dist[v] = dist[u] + 1, q.push(v), p[v] = u; // 3 lines in 1!
            }
            augment(t, INF); // find the min edge weight 'f' in this path, if any
            if (f == 0) break; // we cannot send any more flow ('f' = 0), terminate
            mf += f; // we can still send a flow, increase the max flow!
      }
printf("%d\n", mf); // this is the max flow value
```

## 1.2 Dfs Algos

```
#include <algorithm>
#include <cstdio>
#include <vector>
using namespace std;

typedef pair<int, int> ii; // In this chapter, we will frequently use these
typedef vector<ii> vii; // three data type shortcuts. They may look cryptic
typedef vector<int> vi; // but shortcuts are useful in competitive programming
```

```
#define DFS_WHITE -1 // normal DFS, do not change this with other values (other th
#define DFS_BLACK 1

vector<vii> AdjList;

void printThis(char* message) {
  printf("==================================\n");
  printf("%s\n", message);
  printf("==================================\n");
}

vi dfs_num; // this variable has to be global, we cannot put it in recursion
int numCC;

void dfs(int u) { // DFS for normal usage: as graph traversal algorithm
  printf(" %d", u); // this vertex is visited
  dfs_num[u] = DFS_BLACK; // important step: we mark this vertex as visited
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j]; // v is a (neighbor, weight) pair
    if (dfs_num[v.first] == DFS_WHITE) // important check to avoid cycle
      dfs(v.first); // recursively visits unvisited neighbors v of vertex u
} }

// note: this is not the version on implicit graph
void floodfill(int u, int color) {
  dfs_num[u] = color; // not just a generic DFS_BLACK
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];
    if (dfs_num[v.first] == DFS_WHITE)
      floodfill(v.first, color);
} }

vi topoSort; // global vector to store the toposort in reverse order

void dfs2(int u) { // change function name to differentiate with original dfs
  dfs_num[u] = DFS_BLACK;
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];
    if (dfs_num[v.first] == DFS_WHITE)
      dfs2(v.first);
  }
  topoSort.push_back(u); } // that is, this is the only change

#define DFS_GRAY 2 // one more color for graph edges property check
vi dfs_parent; // to differentiate real back edge versus bidirectional edge

void graphCheck(int u) { // DFS for checking graph edge properties
  dfs_num[u] = DFS_GRAY; // color this as DFS_GRAY (temp) instead of DFS_BLACK
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];
    if (dfs_num[v.first] == DFS_WHITE) { // Tree Edge, DFS_GRAY to DFS_WHITE
      dfs_parent[v.first] = u; // parent of this children is me
      graphCheck(v.first);
    }
    else if (dfs_num[v.first] == DFS_GRAY) { // DFS_GRAY to DFS_GRAY
```

```
      if (v.first == dfs_parent[u]) // to differentiate these two cases
        printf(" Bidirectional (%d, %d) - (%d, %d)\n", u, v.first, v.first, u);
      else // the most frequent application: check if the given graph is cyclic
        printf(" Back Edge (%d, %d) (Cycle)\n", u, v.first);
    }
    else if (dfs_num[v.first] == DFS_BLACK) // DFS_GRAY to DFS_BLACK
      printf(" Forward/Cross Edge (%d, %d)\n", u, v.first);
  }
  dfs_num[u] = DFS_BLACK; // after recursion, color this as DFS_BLACK (DONE)
}

vi dfs_low; // additional information for articulation points/bridges/SCCs
vi articulation_vertex;
int dfsNumberCounter, dfsRoot, rootChildren;

void articulationPointAndBridge(int u) {
  dfs_low[u] = dfs_num[u] = dfsNumberCounter++; // dfs_low[u] <= dfs_num[u]
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];
    if (dfs_num[v.first] == DFS_WHITE) { // a tree edge
      dfs_parent[v.first] = u;
      if (u == dfsRoot) rootChildren++; // special case, count children of root

      articulationPointAndBridge(v.first);

      if (dfs_low[v.first] >= dfs_num[u]) // for articulation point
        articulation_vertex[u] = true; // store this information first
      if (dfs_low[v.first] > dfs_num[u]) // for bridge
        printf(" Edge (%d, %d) is a bridge\n", u, v.first);
      dfs_low[u] = min(dfs_low[u], dfs_low[v.first]); // update dfs_low[u]
    }
    else if (v.first != dfs_parent[u]) // a back edge and not direct cycle
      dfs_low[u] = min(dfs_low[u], dfs_num[v.first]); // update dfs_low[u]
} }

vi S, visited; // additional global variables
int numSCC;

void tarjanSCC(int u) {
  dfs_low[u] = dfs_num[u] = dfsNumberCounter++; // dfs_low[u] <= dfs_num[u]
  S.push_back(u); // stores u in a vector based on order of visitation
  visited[u] = 1;
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
    ii v = AdjList[u][j];
    if (dfs_num[v.first] == DFS_WHITE)
      tarjanSCC(v.first);
    if (visited[v.first]) // condition for update
      dfs_low[u] = min(dfs_low[u], dfs_low[v.first]);
  }

  if (dfs_low[u] == dfs_num[u]) { // if this is a root (start) of an SCC
    printf("SCC %d:", ++numSCC); // this part is done after recursion
    while (1) {
      int v = S.back(); S.pop_back(); visited[v] = 0;
      printf(" %d", v);
      if (u == v) break;
    }
    printf("\n");
} }

int main() {
  int V, total_neighbors, id, weight;

  /*
  // Use the following input:
  // Graph in Figure 4.1
  9
  1 1 0
  3 0 0 2 0 3 0
  2 1 0 3 0
  3 1 0 2 0 4 0
  1 3 0
  0
  2 7 0 8 0
  1 6 0
  1 6 0

  // Example of directed acyclic graph in Figure 4.4 (for toposort)
  8
  2 1 0 2 0
  2 2 0 3 0
  2 3 0 5 0
  1 4 0
  0
  0
  0
  1 6 0

  // Example of directed graph with back edges
  3
  1 1 0
  1 2 0
  1 0 0

  // Left graph in Figure 4.6/4.7/4.8
  6
  1 1 0
  3 0 0 2 0 4 0
  1 1 0
  1 4 0
  3 1 0 3 0 5 0
  1 4 0

  // Right graph in Figure 4.6/4.7/4.8
  6
  1 1 0
```

```
5 0 0 2 0 3 0 4 0 5 0
1 1 0
1 1 0
2 1 0 5 0
2 1 0 4 0

// Directed graph in Figure 4.9
8
1 1 0
1 3 0
1 1 0
2 2 0 4 0
1 5 0
1 7 0
1 4 0
1 6 0
*/

freopen("in_01.txt", "r", stdin);

scanf("%d", &V);
AdjList.assign(V, vii()); // assign blank vectors of pair<int, int>s to AdjList
for (int i = 0; i < V; i++) {
  scanf("%d", &total_neighbors);
  for (int j = 0; j < total_neighbors; j++) {
    scanf("%d %d", &id, &weight);
    AdjList[i].push_back(ii(id, weight));
  }
}

printThis("Standard DFS Demo (the input graph must be UNDIRECTED)");
numCC = 0;
dfs_num.assign(V, DFS_WHITE); // this sets all vertices' state to DFS_WHITE
for (int i = 0; i < V; i++) // for each vertex i in [0..V-1]
  if (dfs_num[i] == DFS_WHITE) // if that vertex is not visited yet
    printf("Component %d:", ++numCC), dfs(i), printf("\n"); // 3 lines here!
printf("There are %d connected components\n", numCC);

printThis("Flood Fill Demo (the input graph must be UNDIRECTED)");
numCC = 0;
dfs_num.assign(V, DFS_WHITE);
for (int i = 0; i < V; i++)
  if (dfs_num[i] == DFS_WHITE)
    floodfill(i, ++numCC);
for (int i = 0; i < V; i++)
  printf("Vertex %d has color %d\n", i, dfs_num[i]);

// make sure that the given graph is DAG
printThis("Topological Sort (the input graph must be DAG)");
topoSort.clear();
dfs_num.assign(V, DFS_WHITE);
for (int i = 0; i < V; i++) // this part is the same as finding CCs
  if (dfs_num[i] == DFS_WHITE)
    dfs2(i);
reverse(topoSort.begin(), topoSort.end()); // reverse topoSort
for (int i = 0; i < (int)topoSort.size(); i++) // or you can simply read
  printf(" %d", topoSort[i]); // the content of 'topoSort' backwards
printf("\n");

printThis("Graph Edges Property Check");
numCC = 0;
dfs_num.assign(V, DFS_WHITE); dfs_parent.assign(V, -1);
for (int i = 0; i < V; i++)
  if (dfs_num[i] == DFS_WHITE)
    printf("Component %d:\n", ++numCC), graphCheck(i); // 2 lines in one

printThis("Articulation Points & Bridges (the input graph must be UNDIRECTED)");
dfsNumberCounter = 0; dfs_num.assign(V, DFS_WHITE); dfs_low.assign(V, 0);
dfs_parent.assign(V, -1); articulation_vertex.assign(V, 0);
printf("Bridges:\n");
for (int i = 0; i < V; i++)
  if (dfs_num[i] == DFS_WHITE) {
    dfsRoot = i; rootChildren = 0;
    articulationPointAndBridge(i);
    articulation_vertex[dfsRoot] = (rootChildren > 1); } // special case
printf("Articulation Points:\n");
for (int i = 0; i < V; i++)
  if (articulation_vertex[i])
    printf(" Vertex %d\n", i);

printThis("Strongly Connected Components (the input graph must be DIRECTED)");
dfs_num.assign(V, DFS_WHITE); dfs_low.assign(V, 0); visited.assign(V, 0);
dfsNumberCounter = numSCC = 0;
for (int i = 0; i < V; i++)
  if (dfs_num[i] == DFS_WHITE)
    tarjanSCC(i);

return 0;
}
```

## 1.3 Bridges

```
#define SZ 100
bool M[SZ][SZ];
int N, colour[SZ], dfsNum[SZ], num, pos[SZ], leastAncestor[SZ], parent[SZ];

void dfs(int u) {
  int v;
  stack<int> S;
  S.push(u);

  while (!S.empty()) {
```

```
    v = S.top();
    if (colour[v] == 0) {
      colour[v] = 1;
      dfsNum[v] = num++;
      leastAncestor[v] = num;
    }

    for (; pos[v] < N; ++pos[v]) {
      if (M[v][pos[v]] && pos[v] != parent[v]) {
        if (colour[pos[v]] == 0) {
          parent[pos[v]] = v;
          S.push(pos[v]);
          break;
        } else
          leastAncestor[v] < ? = dfsNum[pos[v]];
      }
    }

    if (pos[v] == N) {
      colour[v] = 2;
      S.pop();

      if (v != u) leastAncestor[parent[v]] < ? = leastAncestor[v];
    }
  }
}

void Bridge_detection() {
  memset(colour, 0, sizeof(colour));
  memset(pos, 0, sizeof(pos));
  memset(parent, -1, sizeof(parent));
  num = 0;

  int ans = 0;

  for (int i = 0; i < N; i++)
    if (colour[i] == 0) dfs(i);

  for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
      if (parent[j] == i && leastAncestor[j] > dfsNum[i]) {
        printf("%d - %d\n", i, j);
        ++ans;
      }

  printf("%d bridges\n", ans);
}
```

## 1.4 Strongly Connected Components

```
vector<vector<int> > g, gt;
stack<int> S;
int n;
vi scc;

void scc_dfs(const vector<vector<int> > &g, int u, bool addToStack = false) {
  for (int i = 0; i < (int)g[u].size(); ++i) {
    int v = g[u][i];
    if (scc[v] == inf) {
      scc[v] = scc[u];
      scc_dfs(g, v, addToStack);
    }
  }
  if (addToStack) S.push(u);
}

int kosaraju() {
  const int inf = int(1e9 + 7);
  int ans = 0;

  scc.assign(n, inf);
  for (int u = 0; u < n; ++u) {
    if (scc[u] != inf) continue;
    scc[u] = true;
    scc_dfs(g, u, true);
  }
  scc.assign(n, inf);
  while (!S.empty()) {
    int u = S.top();
    S.pop();
    if (scc[u] != inf) continue;
    scc[u] = ans++;
    scc_dfs(gt, u);
  }

  return ans;
}
```

## 1.5 Articulation

```
#include <iostream>
#include <string>
#include <sstream>
#include <stack>

using namespace std;

#define SZ 100
```

```
bool M[SZ][SZ];
int N,colour[SZ],dfsNum[SZ],pos[SZ],leastAncestor[SZ],parent[SZ];

int Articulation_points(int u){
   int ans=0,cont=0,num=0,v;

   memset(colour,0,sizeof(colour));
   memset(pos,0,sizeof(pos));
   memset(parent,-1,sizeof(parent));

   stack<int> S;
   S.push(u);

   while(!S.empty()){
      v=S.top();
      if(colour[v]==0){
         colour[v]=1;
         dfsNum[v]=num++;
         leastAncestor[v]=num;
      }

      for(;pos[v]<N;pos[v]++){
         if(M[v][pos[v]] && pos[v]!=parent[v]){
            if(colour[pos[v]]==0){
               parent[pos[v]]=v;
               S.push(pos[v]);
               if(v==u) cont++;
               break;
            }else leastAncestor[v]<?=dfsNum[pos[v]];
         }
      }

      if(pos[v]==N){
         colour[v]=2;
         S.pop();

         if(v!=u) leastAncestor[parent[v]]<?=leastAncestor[v];
      }
   }

   if(cont>1) ans++;

   for(int i=0;i<N;i++){
      if(i==u) continue;
      for(int j=0;j<N;j++)
         if(M[i][j] && parent[j]==i && leastAncestor[j]>=dfsNum[i]){
            ans++;
            break;
         }
   }

   return ans;
}
```

```
int main(){
   int u,v;
   string s;

   while(1){
      scanf("%d\n",&N);
      if(N==0) break;

      memset(M,false,sizeof(M));

      while(1){
         getline(cin,s);
         istringstream is(s);

         is>>u;
         if(u==0) break;
         u--;

         while(is>>v){
            v--;
            M[u][v]=M[v][u]=true;
         }
      }
      printf("%d\n",Articulation_points(0));
   }
   return 0;
}
```

## 1.6 Bfs Algos

```
#include <algorithm>
#include <cstdio>
#include <vector>
#include <queue>
using namespace std;

typedef pair<int, int> ii; // In this chapter, we will frequently use these
typedef vector<ii> vii; // three data type shortcuts. They may look cryptic
typedef vector<int> vi; // but shortcuts are useful in competitive programming

int V, E, a, b, s;
vector<vii> AdjList;
vi p; // addition: the predecessor/parent vector

void printPath(int u) { // simple function to extract information from 'vi p'
  if (u == s) { printf("%d", u); return; }
  printPath(p[u]); // recursive call: to make the output format: s -> ... -> t
  printf(" %d", u); }
```

```
int main() {
 /*
 // Graph in Figure 4.3, format: list of unweighted edges
 // This example shows another form of reading graph input
 13 16
 0 1 1 2 2 3 0 4 1 5 2 6 3 7 5 6
 4 8 8 9 5 10 6 11 7 12 9 10 10 11 11 12
 */

 freopen("in_04.txt", "r", stdin);

 scanf("%d %d", &V, &E);

 AdjList.assign(V, vii()); // assign blank vectors of pair<int, int>s to AdjList
 for (int i = 0; i < E; i++) {
  scanf("%d %d", &a, &b);
  AdjList[a].push_back(ii(b, 0));
  AdjList[b].push_back(ii(a, 0));
 }

 // as an example, we start from this source, see Figure 4.3
 s = 5;

 // BFS routine
 // inside int main() -- we do not use recursion, thus we do not need to create separate function!
 vi dist(V, 1000000000); dist[s] = 0; // distance to source is 0 (default)
 queue<int> q; q.push(s); // start from source
 p.assign(V, -1); // to store parent information (p must be a global variable!)
 int layer = -1; // for our output printing purpose
 bool isBipartite = true; // addition of one boolean flag, initially true

 while (!q.empty()) {
  int u = q.front(); q.pop(); // queue: layer by layer!
  if (dist[u] != layer) printf("\nLayer %d: ", dist[u]);
  layer = dist[u];
  printf("visit %d, ", u);
  for (int j = 0; j < (int)AdjList[u].size(); j++) {
   ii v = AdjList[u][j]; // for each neighbors of u
   if (dist[v.first] == 1000000000) {
    dist[v.first] = dist[u] + 1; // v unvisited + reachable
    p[v.first] = u; // addition: the parent of vertex v->first is u
    q.push(v.first); // enqueue v for next step
   }
   else if ((dist[v.first] % 2) == (dist[u] % 2)) // same parity
    isBipartite = false;
 } }

 printf("\nShortest path: ");
 printPath(7), printf("\n");
 printf("isBipartite? %d\n", isBipartite);

 return 0;
}
```

## 1.7 Bfs

```
// Program to print BFS traversal from a given
// source vertex. BFS(int s) traverses vertices
// reachable from s.
#include<iostream>
#include <list>

using namespace std;

// This class represents a directed graph using
// adjacency list representation
class Graph
{
   int V; // No. of vertices

   // Pointer to an array containing adjacency
   // lists
   list<int> *adj;
public:
   Graph(int V); // Constructor

   // function to add an edge to graph
   void addEdge(int v, int w);

   // prints BFS traversal from a given source s
   void BFS(int s);
};

Graph::Graph(int V)
{
   this->V = V;
   adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
   adj[v].push_back(w); // Add w to v's list.
}

void Graph::BFS(int s)
{
   // Mark all the vertices as not visited
   bool *visited = new bool[V];
   for(int i = 0; i < V; i++)
      visited[i] = false;
```

```
    // Create a queue for BFS
    list<int> queue;

    // Mark the current node as visited and enqueue it
    visited[s] = true;
    queue.push_back(s);

    // 'i' will be used to get all adjacent
    // vertices of a vertex
    list<int>::iterator i;

    while(!queue.empty())
    {
        // Dequeue a vertex from queue and print it
        s = queue.front();
        cout << s << " ";
        queue.pop_front();

        // Get all adjacent vertices of the dequeued
        // vertex s. If a adjacent has not been visited,
        // then mark it visited and enqueue it
        for (i = adj[s].begin(); i != adj[s].end(); ++i)
        {
            if (!visited[*i])
            {
                visited[*i] = true;
                queue.push_back(*i);
            }
        }
    }
}

// Driver program to test methods of graph class
int main()
{
    // Create a graph given in the above diagram
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);
    cout << "Following is Breadth First Traversal "
        << "(starting from vertex 2) \n";
    g.BFS(2);
    return 0;
}
```

## 1.8 Dfs

```
// C++ program to print DFS traversal from
// a given vertex in a given graph
#include<iostream>
#include<list>
using namespace std;

// Graph class represents a directed graph
// using adjacency list representation
class Graph
{
      int V; // No. of vertices

      // Pointer to an array containing
      // adjacency lists
      list<int> *adj;

      // A recursive function used by DFS
      void DFSUtil(int v, bool visited[]);
public:
      Graph(int V); // Constructor

      // function to add an edge to graph
      void addEdge(int v, int w);

      // DFS traversal of the vertices
      // reachable from v
      void DFS(int v);
};

Graph::Graph(int V)
{
      this->V = V;
      adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
      adj[v].push_back(w); // Add w to v's list.
}

void Graph::DFSUtil(int v, bool visited[])
{
      // Mark the current node as visited and
      // print it
      visited[v] = true;
      cout << v << " ";

      // Recur for all the vertices adjacent
      // to this vertex
```

```
	list<int>::iterator i;
	for (i = adj[v].begin(); i != adj[v].end(); ++i)
		if (!visited[*i])
			DFSUtil(*i, visited);
}

// DFS traversal of the vertices reachable from v.
// It uses recursive DFSUtil()
void Graph::DFS(int v)
{
	// Mark all the vertices as not visited
	bool *visited = new bool[V];
	for (int i = 0; i < V; i++)
		visited[i] = false;

	// Call the recursive helper function
	// to print DFS traversal
	DFSUtil(v, visited);
}

int main()
{
	// Create a graph given in the above diagram
	Graph g(4);
	g.addEdge(0, 1);
	g.addEdge(0, 2);
	g.addEdge(1, 2);
	g.addEdge(2, 0);
	g.addEdge(2, 3);
	g.addEdge(3, 3);

	cout << "Following is Depth First Traversal"
			" (starting from vertex 2) \n";
	g.DFS(2);

	return 0;
}
```

# 2 Java

## 2.1 Fast Input Output Template

```
import java.io.*;
import java.math.*;
import java.util.*;
import java.lang.*;

public class Main {
 public static void main(String[] args) {
  InputReader in = new InputReader(System.in);
  OutputWriter out = new OutputWriter(System.out);
  // Do your thing
  out.close();
 }
}

class InputReader {
 private InputStream stream;
 private byte[] buf = new byte[1024];
 private int curChar;
 private int numChars;
 private SpaceCharFilter filter;

 public InputReader(InputStream stream) {
  this.stream = stream;
 }

 public int read() {
  if (numChars == -1) {
   throw new InputMismatchException();
  }

  if (curChar >= numChars) {
   curChar = 0;
   try {
    numChars = stream.read(buf);
   } catch (IOException e) {
    throw new InputMismatchException();
   }
   if (numChars <= 0) {
    return -1;
   }
  }

  return buf[curChar++];
 }

 public int readInt() {
  int c = read();
  while (isSpaceChar(c)) {
   c = read();
  }

  int sgn = 1;
  if (c == '-') {
   sgn = -1;
   c = read();
  }

  int res = 0;
```

```
    do {
      if (c < '0' || c > '9') {
        throw new InputMismatchException();
      }
      res *= 10;
      res += c - '0';
      c = read();
    } while (!isSpaceChar(c));

    return res * sgn;
  }

  public String readString() {
    int c = read();
    while (isSpaceChar(c)) {
      c = read();
    }

    StringBuilder res = new StringBuilder();
    do {
      res.appendCodePoint(c);
      c = read();
    } while (!isSpaceChar(c));
    return res.toString();
  }

  public boolean isSpaceChar(int c) {
    if (filter != null) {
      return filter.isSpaceChar(c);
    }
    return c == ' ' || c == '\n' || c == '\r' || c == '\t' || c == -1;
  }

  public String next() {
    return readString();
  }

  public interface SpaceCharFilter {
    public boolean isSpaceChar(int ch);
  }
}

class OutputWriter {
  private final PrintWriter writer;

  public OutputWriter(OutputStream outputStream) {
    writer = new PrintWriter(
      new BufferedWriter(new OutputStreamWriter(outputStream))
    );
  }

  public OutputWriter(Writer writer) {
    this.writer = new PrintWriter(writer);
  }

  public void print(Object... objects) {
    for (int i = 0; i < objects.length; i++) {
      if (i != 0) {
        writer.print(' ');
      }
      writer.print(objects[i]);
    }
  }

  public void printLine(Object... objects) {
    print(objects);
    writer.println();
  }

  public void close() {
    writer.close();
  }

  public void flush() {
    writer.flush();
  }
}

class IOUtils {
  public static int[] readIntArray(InputReader in, int size) {
    int[] array = new int[size];
    for (int i = 0; i < size; i++) {
      array[i] = in.readInt();
    }
    return array;
  }
}
```

# 3 Geometry

## 3.1 Geometry

```
#define EPS 1e-8
#define PI acos(-1)
#define Vector Point

struct Point {
  double x, y;
  Point() {}
  Point(double a, double b) {
```

```
    x = a;
    y = b;
  }
  double mod2() { return x * x + y * y; }
  double mod() { return sqrt(x * x + y * y); }
  double arg() { return atan2(y, x); }
  Point ort() { return Point(-y, x); }
  Point unit() {
    double k = mod();
    return Point(x / k, y / k);
  }
};

Point operator+(const Point &a, const Point &b) {
  return Point(a.x + b.x, a.y + b.y);
}
Point operator-(const Point &a, const Point &b) {
  return Point(a.x - b.x, a.y - b.y);
}
Point operator/(const Point &a, double k) { return Point(a.x / k, a.y / k); }
Point operator*(const Point &a, double k) { return Point(a.x * k, a.y * k); }

bool operator==(const Point &a, const Point &b) {
  return abs(a.x - b.x) < EPS && abs(a.y - b.y) < EPS;
}
bool operator!=(const Point &a, const Point &b) { return !(a == b); }
bool operator<(const Point &a, const Point &b) {
  if (abs(a.x - b.x) > EPS) return a.x < b.x;
  return a.y + EPS < b.y;
}

//### FUNCIONES BASICAS
//############################################################

double dist(const Point &A, const Point &B) {
  return hypot(A.x - B.x, A.y - B.y);
}
double cross(const Vector &A, const Vector &B) { return A.x * B.y - A.y * B.x; }
double dot(const Vector &A, const Vector &B) { return A.x * B.x + A.y * B.y; }
double area(const Point &A, const Point &B, const Point &C) {
  return cross(B - A, C - A);
}

// Heron triangulo y cuadrilatero ciclico
// http://mathworld.wolfram.com/CyclicQuadrilateral.html
// http://www.spoj.pl/problems/QUADAREA/

double areaHeron(double a, double b, double c) {
  double s = (a + b + c) / 2;
  return sqrt(s * (s - a) * (s - b) * (s - c));
}

double circumradius(double a, double b, double c) {
  return a * b * c / (4 * areaHeron(a, b, c));
}

double areaHeron(double a, double b, double c, double d) {
  double s = (a + b + c + d) / 2;
  return sqrt((s - a) * (s - b) * (s - c) * (s - d));
}

double circumradius(double a, double b, double c, double d) {
  return sqrt((a * b + c * d) * (a * c + b * d) * (a * d + b * c)) /
       (4 * areaHeron(a, b, c, d));
}

//### DETERMINA SI P PERTENECE AL SEGMENTO AB
//###########################################
bool between(const Point &A, const Point &B, const Point &P) {
  return P.x + EPS >= min(A.x, B.x) && P.x <= max(A.x, B.x) + EPS &&
       P.y + EPS >= min(A.y, B.y) && P.y <= max(A.y, B.y) + EPS;
}

bool onSegment(const Point &A, const Point &B, const Point &P) {
  return abs(area(A, B, P)) < EPS && between(A, B, P);
}

//### DETERMINA SI EL SEGMENTO P1Q1 SE INTERSECTA CON EL SEGMENTO P2Q2
//#####################
// funciona para cualquiera P1, P2, P3, P4
bool intersects(const Point &P1, const Point &P2, const Point &P3,
        const Point &P4) {
  double A1 = area(P3, P4, P1);
  double A2 = area(P3, P4, P2);
  double A3 = area(P1, P2, P3);
  double A4 = area(P1, P2, P4);

  if (((A1 > 0 && A2 < 0) || (A1 < 0 && A2 > 0)) &&
     ((A3 > 0 && A4 < 0) || (A3 < 0 && A4 > 0)))
    return true;

  else if (A1 == 0 && onSegment(P3, P4, P1))
    return true;
  else if (A2 == 0 && onSegment(P3, P4, P2))
    return true;
  else if (A3 == 0 && onSegment(P1, P2, P3))
    return true;
  else if (A4 == 0 && onSegment(P1, P2, P4))
    return true;
  else
    return false;
}

//### DETERMINA SI A, B, M, N PERTENECEN A LA MISMA RECTA
//##############################
bool sameLine(Point P1, Point P2, Point P3, Point P4) {
```

```
  return area(P1, P2, P3) == 0 && area(P1, P2, P4) == 0;
}
//### SI DOS SEGMENTOS O RECTAS SON PARALELOS
//##################################################
bool isParallel(const Point &P1, const Point &P2, const Point &P3,
                const Point &P4) {
  return cross(P2 - P1, P4 - P3) == 0;
}

//### PUNTO DE INTERSECCION DE DOS RECTAS NO PARALELAS
//#################################
Point lineIntersection(const Point &A, const Point &B, const Point &C,
                       const Point &D) {
  return A + (B - A) * (cross(C - A, D - C) / cross(B - A, D - C));
}

Point circumcenter(const Point &A, const Point &B, const Point &C) {
  return (A + B + (A - B).ort() * dot(C - B, A - C) / cross(A - B, A - C)) / 2;
}

//### FUNCIONES BASICAS DE POLIGONOS
//###############################################
bool isConvex(const vector<Point> &P) {
  int n = P.size(), pos = 0, neg = 0;
  for (int i = 0; i < n; i++) {
    double A = area(P[i], P[(i + 1) % n], P[(i + 2) % n]);
    if (A < 0)
      neg++;
    else if (A > 0)
      pos++;
  }
  return neg == 0 || pos == 0;
}

double area(const vector<Point> &P) {
  int n = P.size();
  double A = 0;
  for (int i = 1; i <= n - 2; i++) A += area(P[0], P[i], P[i + 1]);
  return abs(A / 2);
}

bool pointInPoly(const vector<Point> &P, const Point &A) {
  int n = P.size(), cnt = 0;
  for (int i = 0; i < n; i++) {
    int inf = i, sup = (i + 1) % n;
    if (P[inf].y > P[sup].y) swap(inf, sup);
    if (P[inf].y <= A.y && A.y < P[sup].y)
      if (area(A, P[inf], P[sup]) > 0) cnt++;
  }
  return (cnt % 2) == 1;
}

//### CONVEX HULL
//####################################################################
// O(nh)
vector<Point> ConvexHull(vector<Point> S) {
  sort(all(S));

  int it = 0;
  Point primero = S[it], ultimo = primero;

  int n = S.size();

  vector<Point> convex;
  do {
    convex.push_back(S[it]);
    it = (it + 1) % n;

    for (int i = 0; i < S.size(); i++) {
      if (S[i] != ultimo && S[i] != S[it]) {
        if (area(ultimo, S[it], S[i]) < EPS) it = i;
      }
    }

    ultimo = S[it];
  } while (ultimo != primero);

  return convex;
}

// O(n log n)
vector<Point> ConvexHull(vector<Point> P) {
  sort(P.begin(), P.end());
  int n = P.size(), k = 0;
  Point H[2 * n];

  for (int i = 0; i < n; ++i) {
    while (k >= 2 && area(H[k - 2], H[k - 1], P[i]) <= 0) --k;
    H[k++] = P[i];
  }

  for (int i = n - 2, t = k; i >= 0; --i) {
    while (k > t && area(H[k - 2], H[k - 1], P[i]) <= 0) --k;
    H[k++] = P[i];
  }

  return vector<Point>(H, H + k - 1);
}

//### DETERMINA SI P ESTA EN EL INTERIOR DEL POLIGONO CONVEXO A
//########################

// O (log n)
bool isInConvex(vector<Point> &A, const Point &P) {
  int n = A.size(), lo = 1, hi = A.size() - 1;
```

```
  if (area(A[0], A[1], P) <= 0) return 0;
  if (area(A[n - 1], A[0], P) <= 0) return 0;

  while (hi - lo > 1) {
    int mid = (lo + hi) / 2;

    if (area(A[0], A[mid], P) > 0)
      lo = mid;
    else
      hi = mid;
  }

  return area(A[lo], A[hi], P) > 0;
}

// O(n)
Point norm(const Point &A, const Point &O) {
  Vector V = A - O;
  V = V * 10000000000.0 / V.mod();
  return O + V;
}

bool isInConvex(vector<Point> &A, vector<Point> &B) {
  if (!isInConvex(A, B[0]))
    return 0;
  else {
    int n = A.size(), p = 0;

    for (int i = 1; i < B.size(); i++) {
      while (!intersects(A[p], A[(p + 1) % n], norm(B[i], B[0]), B[0]))
        p = (p + 1) % n;

      if (area(A[p], A[(p + 1) % n], B[i]) <= 0) return 0;
    }

    return 1;
  }
}

//##### SMALLEST ENCLOSING CIRCLE O(n)
//###############################################
// http://www.cs.uu.nl/docs/vakken/ga/slides4b.pdf
// http://www.spoj.pl/problems/ALIENS/

pair<Point, double> enclosingCircle(vector<Point> P) {
  random_shuffle(P.begin(), P.end());

  Point O(0, 0);
  double R2 = 0;

  for (int i = 0; i < P.size(); i++) {
    if ((P[i] - O).mod2() > R2 + EPS) {
      O = P[i], R2 = 0;
      for (int j = 0; j < i; j++) {
        if ((P[j] - O).mod2() > R2 + EPS) {
          O = (P[i] + P[j]) / 2, R2 = (P[i] - P[j]).mod2() / 4;
          for (int k = 0; k < j; k++)
            if ((P[k] - O).mod2() > R2 + EPS)
              O = circumcenter(P[i], P[j], P[k]), R2 = (P[k] - O).mod2();
        }
      }
    }
  }
  return make_pair(O, sqrt(R2));
}

//##### CLOSEST PAIR OF POINTS
//#########################################################
bool XYorder(Point P1, Point P2) {
  if (P1.x != P2.x) return P1.x < P2.x;
  return P1.y < P2.y;
}
bool YXorder(Point P1, Point P2) {
  if (P1.y != P2.y) return P1.y < P2.y;
  return P1.x < P2.x;
}
double closest_recursive(vector<Point> vx, vector<Point> vy) {
  if (vx.size() == 1) return 1e20;
  if (vx.size() == 2) return dist(vx[0], vx[1]);

  Point cut = vx[vx.size() / 2];

  vector<Point> vxL, vxR;
  for (int i = 0; i < vx.size(); i++)
    if (vx[i].x < cut.x || (vx[i].x == cut.x && vx[i].y <= cut.y))
      vxL.push_back(vx[i]);
    else
      vxR.push_back(vx[i]);

  vector<Point> vyL, vyR;
  for (int i = 0; i < vy.size(); i++)
    if (vy[i].x < cut.x || (vy[i].x == cut.x && vy[i].y <= cut.y))
      vyL.push_back(vy[i]);
    else
      vyR.push_back(vy[i]);

  double dL = closest_recursive(vxL, vyL);
  double dR = closest_recursive(vxR, vyR);
  double d = min(dL, dR);

  vector<Point> b;
  for (int i = 0; i < vy.size(); i++)
    if (abs(vy[i].x - cut.x) <= d) b.push_back(vy[i]);

  for (int i = 0; i < b.size(); i++)
    for (int j = i + 1; j < b.size() && (b[j].y - b[i].y) <= d; j++)
```

```
    d = min(d, dist(b[i], b[j]));

  return d;
}
double closest(vector<Point> points) {
  vector<Point> vx = points, vy = points;
  sort(vx.begin(), vx.end(), XYorder);
  sort(vy.begin(), vy.end(), YXorder);

  for (int i = 0; i + 1 < vx.size(); i++)
    if (vx[i] == vx[i + 1]) return 0.0;

  return closest_recursive(vx, vy);
}

// INTERSECCION DE CIRCULOS
vector<Point> circleCircleIntersection(Point O1, double r1, Point O2,
                                       double r2) {
  vector<Point> X;

  double d = dist(O1, O2);

  if (d > r1 + r2 || d < max(r2, r1) - min(r2, r1))
    return X;
  else {
    double a = (r1 * r1 - r2 * r2 + d * d) / (2.0 * d);
    double b = d - a;
    double c = sqrt(abs(r1 * r1 - a * a));

    Vector V = (O2 - O1).unit();
    Point H = O1 + V * a;

    X.push_back(H + V.ort() * c);

    if (c > EPS) X.push_back(H - V.ort() * c);
  }

  return X;
}

// LINEA AB vs CIRCULO (O, r)
// 1. Mucha perdida de precision, reemplazar por resultados de formula.
// 2. Considerar line o segment

vector<Point> lineCircleIntersection(Point A, Point B, Point O, long double r) {
  vector<Point> X;

  Point H1 = O + (B - A).ort() * cross(O - A, B - A) / (B - A).mod2();
  long double d2 = cross(O - A, B - A) * cross(O - A, B - A) / (B - A).mod2();

  if (d2 <= r * r + EPS) {
    long double k = sqrt(abs(r * r - d2));

    Point P1 = H1 + (B - A) * k / (B - A).mod();
    Point P2 = H1 - (B - A) * k / (B - A).mod();

    if (between(A, B, P1)) X.push_back(P1);

    if (k > EPS && between(A, B, P2)) X.push_back(P2);
  }

  return X;
}

//### PROBLEMAS BASICOS
//#############################################################
void CircumscribedCircle() {
  int x1, y1, x2, y2, x3, y3;
  scanf("%d %d %d %d %d %d", &x1, &y1, &x2, &y2, &x3, &y3);

  Point A(x1, y1), B(x2, y2), C(x3, y3);

  Point P1 = (A + B) / 2.0;
  Point P2 = P1 + (B - A).ort();
  Point P3 = (A + C) / 2.0;
  Point P4 = P3 + (C - A).ort();

  Point CC = lineIntersection(P1, P2, P3, P4);
  double r = dist(A, CC);

  printf("(%.6lf,%.6lf,%.6lf)\n", CC.x, CC.y, r);
}

void InscribedCircle() {
  int x1, y1, x2, y2, x3, y3;
  scanf("%d %d %d %d %d %d", &x1, &y1, &x2, &y2, &x3, &y3);

  Point A(x1, y1), B(x2, y2), C(x3, y3);

  Point AX = A + (B - A).unit() + (C - A).unit();
  Point BX = B + (A - B).unit() + (C - B).unit();

  Point CC = lineIntersection(A, AX, B, BX);
  double r = abs(area(A, B, CC) / dist(A, B));

  printf("(%.6lf,%.6lf,%.6lf)\n", CC.x, CC.y, r);
}

vector<Point> TangentLineThroughPoint(Point P, Point C, long double r) {
  vector<Point> X;

  long double h2 = (C - P).mod2();
  if (h2 < r * r)
    return X;
  else {
    long double d = sqrt(h2 - r * r);
```

```
    long double m1 = (r * (P.x - C.x) + d * (P.y - C.y)) / h2;
    long double n1 = (P.y - C.y - d * m1) / r;

    long double n2 = (d * (P.x - C.x) + r * (P.y - C.y)) / h2;
    long double m2 = (P.x - C.x - d * n2) / r;

    X.push_back(C + Point(m1, n1) * r);
    if (d != 0) X.push_back(C + Point(m2, n2) * r);

    return X;
  }
}

void TangentLineThroughPoint() {
  int xc, yc, r, xp, yp;
  scanf("%d %d %d %d %d", &xc, &yc, &r, &xp, &yp);

  Point C(xc, yc), P(xp, yp);

  double hyp = dist(C, P);
  if (hyp < r)
    printf("[]\n");
  else {
    double d = sqrt(hyp * hyp - r * r);

    double m1 = (r * (P.x - C.x) + d * (P.y - C.y)) / (r * r + d * d);
    double n1 = (P.y - C.y - d * m1) / r;
    double ang1 = 180 * atan(-m1 / n1) / PI + EPS;
    if (ang1 < 0) ang1 += 180.0;

    double n2 = (d * (P.x - C.x) + r * (P.y - C.y)) / (r * r + d * d);
    double m2 = (P.x - C.x - d * n2) / r;
    double ang2 = 180 * atan(-m2 / n2) / PI + EPS;
    if (ang2 < 0) ang2 += 180.0;

    if (ang1 > ang2) swap(ang1, ang2);

    if (d == 0)
      printf("[%.6lf]\n", ang1);
    else
      printf("[%.6lf,%.6lf]\n", ang1, ang2);
  }
}

void CircleThroughAPointAndTangentToALineWithRadius() {
  int xp, yp, x1, y1, x2, y2, r;
  scanf("%d %d %d %d %d %d %d", &xp, &yp, &x1, &y1, &x2, &y2, &r);

  Point P(xp, yp), A(x1, y1), B(x2, y2);

  Vector V = (B - A).ort() * r / (B - A).mod();

  Point X[2];
  int cnt = 0;

  Point H1 = P + (B - A).ort() * cross(P - A, B - A) / (B - A).mod2() + V;
  double d1 = abs(r + cross(P - A, B - A) / (B - A).mod());

  if (d1 - EPS <= r) {
    double k = sqrt(abs(r * r - d1 * d1));

    X[cnt++] = Point(H1 + (B - A).unit() * k);

    if (k > EPS) X[cnt++] = Point(H1 - (B - A).unit() * k);
  }

  Point H2 = P + (B - A).ort() * cross(P - A, B - A) / (B - A).mod2() - V;
  double d2 = abs(r - cross(P - A, B - A) / (B - A).mod());

  if (d2 - EPS <= r) {
    double k = sqrt(abs(r * r - d2 * d2));

    X[cnt++] = Point(H2 + (B - A).unit() * k);

    if (k > EPS) X[cnt++] = Point(H2 - (B - A).unit() * k);
  }

  sort(X, X + cnt);

  if (cnt == 0)
    printf("[]\n");
  else if (cnt == 1)
    printf("[(%.6lf,%.6lf)]\n", X[0].x, X[0].y);
  else if (cnt == 2)
    printf("[(%.6lf,%.6lf),(%.6lf,%.6lf)]\n", X[0].x, X[0].y, X[1].x, X[1].y);
}

void CircleTangentToTwoLinesWithRadius() {
  int x1, y1, x2, y2, x3, y3, x4, y4, r;
  scanf("%d %d %d %d %d %d %d %d %d", &x1, &y1, &x2, &y2, &x3, &y3, &x4, &y4,
      &r);

  Point A1(x1, y1), B1(x2, y2), A2(x3, y3), B2(x4, y4);

  Vector V1 = (B1 - A1).ort() * r / (B1 - A1).mod();
  Vector V2 = (B2 - A2).ort() * r / (B2 - A2).mod();

  Point X[4];
  X[0] = lineIntersection(A1 + V1, B1 + V1, A2 + V2, B2 + V2);
  X[1] = lineIntersection(A1 + V1, B1 + V1, A2 - V2, B2 - V2);
  X[2] = lineIntersection(A1 - V1, B1 - V1, A2 + V2, B2 + V2);
  X[3] = lineIntersection(A1 - V1, B1 - V1, A2 - V2, B2 - V2);

  sort(X, X + 4);
  printf("[(%.6lf,%.6lf),(%.6lf,%.6lf),(%.6lf,%.6lf),(%.6lf,%.6lf)]\n", X[0].x,
```

```
      X[0].y, X[1].x, X[1].y, X[2].x, X[2].y, X[3].x, X[3].y);
}

void CircleTangentToTwoDisjointCirclesWithRadius() {
  int x1, y1, r1, x2, y2, r2, r;
  scanf("%d %d %d %d %d %d %d", &x1, &y1, &r1, &x2, &y2, &r2, &r);

  Point A(x1, y1), B(x2, y2);

  r1 += r;
  r2 += r;

  double d = dist(A, B);

  if (d > r1 + r2 || d < max(r1, r2) - min(r1, r2))
    printf("[]\n");
  else {
    double a = (r1 * r1 - r2 * r2 + d * d) / (2.0 * d);
    double b = d - a;
    double c = sqrt(abs(r1 * r1 - a * a));

    Vector V = (B - A).unit();
    Point H = A + V * a;

    Point P1 = H + V.ort() * c;
    Point P2 = H - V.ort() * c;

    if (P2 < P1) swap(P1, P2);

    if (P1 == P2)
      printf("[(%.6lf,%.6lf)]\n", P1.x, P1.y);
    else
      printf("[(%.6lf,%.6lf),(%.6lf,%.6lf)]\n", P1.x, P1.y, P2.x, P2.y);
  }
}
```

# 4 Strings

## 4.1 Edit Distance

```
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;

int main() {
  char A[20] = "ACAATCC", B[20] = "AGCATGC";
  int n = (int)strlen(A), m = (int)strlen(B);
  int i, j, table[20][20]; // Needleman Wunsnch's algorithm
  memset(table, 0, sizeof table);
  // insert/delete = -1 point
  for (i = 1; i <= n; i++)
    table[i][0] = i * -1;
  for (j = 1; j <= m; j++)
    table[0][j] = j * -1;
  for (i = 1; i <= n; i++)
    for (j = 1; j <= m; j++) {
      // match = 2 points, mismatch = -1 point
      table[i][j] = table[i - 1][j - 1] + (A[i - 1] == B[j - 1] ? 2 : -1); // cost
      // insert/delete = -1 point
      table[i][j] = max(table[i][j], table[i - 1][j] - 1); // delete
      table[i][j] = max(table[i][j], table[i][j - 1] - 1); // insert
    }
  printf("DP table:\n");
  for (i = 0; i <= n; i++) {
    for (j = 0; j <= m; j++)
      printf("%3d", table[i][j]);
    printf("\n");
  }
  printf("Maximum Alignment Score: %d\n", table[n][m]);
  return 0;
}
```

## 4.2 Trie

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

template <int alphabet_size>
struct TrieNode {
  int n_words, n_prefixes;
  int child[alphabet_size] = {0};

  TrieNode() : n_words(0), n_prefixes(0) { }
};

template <int alphabet_size>
struct Trie {
  static constexpr int npos = -1;
  using TNode = TrieNode<alphabet_size>;
  vector<TNode> nodes;

  Trie() { nodes.emplace_back(); }

  /*
  ** Maps the given char to an unsigned integer
```

```
** inside the range [0..alphabet_size)
*/
int char_to_child(char c) {
 int result = c - '0';
 //assert(0 <= result && result < alphabet_size);
 return result;
}

/*
** Adds the given word to the trie
*/
void add_word(const char *s) {
 int current = 0;

 for (int i = 0; s[i]; i++) {
  nodes[current].n_prefixes += 1;

  int next_child = char_to_child(s[i]);

  int next_node = nodes[current].child[next_child];
  if (next_node == 0) {
   next_node = nodes.size();
   nodes[current].child[next_child] = next_node;
   nodes.emplace_back();
  }
  current = next_node;
 }

 nodes[current].n_prefixes += 1;
 nodes[current].n_words += 1;
}

/*
** Traverses the trie, following the content of string 's'.
** Returns the node ID where the traversal stopped, or
** Trie::npos if it couldn't follow the whole string.
*/
int traverse(const char *s) {
 int current = 0;

 for (int i = 0; s[i]; i++) {
  int next_child = char_to_child(s[i]);

  int next_node = nodes[current].child[next_child];
  if (next_node == 0) {
   return Trie::npos;
  }

  current = next_node;
 }

 return current;
}

 int count_prefixes(const char *s) {
  int node = traverse(s);
  int result = (node == Trie::npos ? 0 : nodes[node].n_prefixes);
  return result;
 }

 int count_words(const char *s) {
  int node = traverse(s);
  int result = (node == Trie::npos ? 0 : nodes[node].n_words);
  return result;
 }
};
```

# 5 Data Structures

## 5.1 Ranged Fenwick Tree

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <cstring>
using namespace std;

// Implementation based on the code provided at Petr's blog. Nevertheless,
// an easy and helpful explanation can be found in TopCoder's forums
// http://petr-mitrichev.blogspot.com/2013/05/fenwick-tree-range-updates.html
// http://apps.topcoder.com/forums/?module=RevisionHistory&messageID=1407869
template <typename T>
class RangedFenwickTree {
 public:
  RangedFenwickTree() {}

  RangedFenwickTree(unsigned int n) { Init(n); }

  T Query(int at) const {
   T mul = 0, add = 0;
   int start = at;
   while (at >= 0) {
    mul += dataMul[at];
    add += dataAdd[at];
    at = (at & (at + 1)) - 1;
   }
   return mul * start + add;
  }

  T QueryInterval(int x, int y) const { return Query(y) - Query(x - 1); }
```

```
  void Update(int x, int y, T delta) {
    InternalUpdate(x, delta, -delta * (x - 1));
    if (y + 1 < (int)this->size()) InternalUpdate(y + 1, -delta, delta * y);
  }

  unsigned int size() const { return dataMul.size(); }

  void Init(unsigned int n) {
    dataMul.assign(n, 0);
    dataAdd.assign(n, 0);
  }

  vector<T> dataMul, dataAdd;

 private:
  void InternalUpdate(int x, T mul, T add) {
    for (int i = x; i < (int)this->size(); i = (i | (i + 1))) {
      dataMul[i] += mul;
      dataAdd[i] += add;
    }
  }
};

// Extension of the Ranged Fenwick Tree to 2D
template <typename T>
class RangedFenwickTree2D {
 public:
  RangedFenwickTree2D() {}

  RangedFenwickTree2D(unsigned int m, unsigned int n) { Init(m, n); }

  T Query(int x, int y) const {
    T mul = 0, add = 0;
    for (int i = x; i >= 0; i = (i & (i + 1)) - 1) {
      mul += dataMul[i].Query(y);
      add += dataAdd[i].Query(y);
    }
    return mul * x + add;
  }

  T QuerySubmatrix(int x1, int y1, int x2, int y2) const {
    T result = Query(x2, y2);
    if (x1 > 0) result -= Query(x1 - 1, y2);
    if (y1 > 0) result -= Query(x2, y1 - 1);
    if (x1 > 0 && y1 > 0) result += Query(x1 - 1, y1 - 1);
    return result;
  }

  void Update(int x1, int y1, int x2, int y2, T delta) {
    for (int i = x1; i < (int)dataMul.size(); i |= i + 1) {
      dataMul[i].Update(y1, y2, delta);
      dataAdd[i].Update(y1, y2, -delta * (x1 - 1));
    }
    for (int i = x2 + 1; i < (int)dataMul.size(); i |= i + 1) {
      dataMul[i].Update(y1, y2, -delta);
      dataAdd[i].Update(y1, y2, delta * x2);
    }
  }

  void Init(unsigned int m, unsigned int n) {
    // dataMul efficient initialization
    if (dataMul.size() == m) {
      for (int i = 0; i < (int)m; i++) dataMul[i].Init(n);
    } else {
      dataMul.assign(m, RangedFenwickTree<T>(n));
    }
    // dataAdd efficient initialization
    if (dataAdd.size() == m) {
      for (int i = 0; i < (int)m; i++) dataAdd[i].Init(n);
    } else {
      dataAdd.assign(m, RangedFenwickTree<T>(n));
    }
  }

  vector<RangedFenwickTree<T> > dataMul, dataAdd;
};

int main() {
  // EXAMPLE USAGE
  // Solution for http://www.spoj.com/problems/USUBQSUB/

  ios_base::sync_with_stdio(0);
  cin.tie(0);

  int n, m;
  cin >> n >> m;
  RangedFenwickTree2D<long long> f(n + 1, n + 1);
  while (m--) {
    int kind, x1, y1, x2, y2;
    cin >> kind >> x1 >> y1 >> x2 >> y2;
    if (kind == 1) {
      cout << f.QuerySubmatrix(x1, y1, x2, y2) << '\n';
    } else {
      int value;
      cin >> value;
      f.Update(x1, y1, x2, y2, value);
    }
  }

  return 0;
}
```

## 5.2 Fenwick Tree

```
// Most of the implementation comes from e-maxx.ru, although several
// things can also be found on the TopCoder tutorial about BITs
// community.topcoder.com/tc?module=Static&d1=tutorials&d2=binaryIndexedTrees
// e-maxx.ru/algo/fenwick_tree
template <typename T>
class FenwickTree {
 public:
  FenwickTree() {}

  FenwickTree(unsigned int n) { Init(n); }

  T Query(int x) const {
    T result = 0;
    for (int i = x; i >= 0; i = (i & (i + 1)) - 1) result += data[i];
    return result;
  }

  T QueryInterval(int x, int y) const { return Query(y) - Query(x - 1); }

  T QuerySingle(int x) const {
    T result = data[x];
    if (x > 0) {
      int y = (x & (x + 1)) - 1;
      x -= 1;
      while (x != y) {
        result -= data[x];
        x = (x & (x + 1)) - 1;
      }
    }
    return result;
  }

  void Update(int x, T delta) {
    for (int i = x; i < (int)data.size(); i = (i | (i + 1))) data[i] += delta;
  }

  unsigned int size() const { return data.size(); }

  void Init(unsigned int n) { data.assign(n, 0); }

  vector<T> data;
};

// Extension of the Fenwick Tree to 2D
template <typename T>
class FenwickTree2D {
 public:
  FenwickTree2D() {}

  FenwickTree2D(unsigned int m, unsigned int n) { Init(m, n); }

  T Query(int x, int y) const {
    T result = 0;
    for (int i = x; i >= 0; i = (i & (i + 1)) - 1) result += data[i].Query(y);
    return result;
  }

  void Update(int x, int y, T delta) {
    for (int i = x; i < (int)data.size(); i = (i | (i + 1)))
      data[i].Update(y, delta);
  }

  void Init(unsigned int m, unsigned int n) {
    if (data.size() == m) {
      for (int i = 0; i < (int)m; i++) data[i].Init(n);
    } else {
      data.assign(m, FenwickTree<T>(n));
    }
  }

  vector<FenwickTree<T> > data;
};

/*
** BIT Linear Construction Snippet
**

class Fenwick{
  int *m, N;
public:
  Fenwick(int a[], int n);
};

Fenwick::Fenwick(int a[], int n){
  N = n;
  m = new int[N];
  memset(m, 0, sizeof(int)*N);
  for(int i=0;i<N;++i){
    m[i] += a[i];
    if((i|(i+1))<N) m[i|(i+1)] += m[i];
  }
}
*/
```

## 5.3 Longest Common Ancestor

```
#define MAX_N 100000
#define LOG2_MAXN 16
```

```
// NOTA : memset(parent,-1,sizeof(parent));
int N, parent[MAX_N], L[MAX_N];
int P[MAX_N][LOG2_MAXN + 1];

int get_level(int u) {
 if (L[u] != -1)
  return L[u];
 else if (parent[u] == -1)
  return 0;
 return 1 + get_level(parent[u]);
}

void init() {
 memset(L, -1, sizeof(L));
 for (int i = 0; i < N; ++i) L[i] = get_level(i);

 memset(P, -1, sizeof(P));

 for (int i = 0; i < N; ++i) P[i][0] = parent[i];

 for (int j = 1; (1 << j) < N; ++j)
  for (int i = 0; i < N; ++i)
   if (P[i][j - 1] != -1) P[i][j] = P[P[i][j - 1]][j - 1];
}

int LCA(int p, int q) {
 if (L[p] < L[q]) swap(p, q);

 int log = 1;
 while ((1 << log) <= L[p]) ++log;
 --log;

 for (int i = log; i >= 0; --i)
  if (L[p] - (1 << i) >= L[q]) p = P[p][i];

 if (p == q) return p;

 for (int i = log; i >= 0; --i) {
  if (P[p][i] != -1 && P[p][i] != P[q][i]) {
   p = P[p][i];
   q = P[q][i];
  }
 }

 return parent[p];
}
```

## 5.4 Segment Tree

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

/*
** Generic segment tree with lazy propagation (requires C++11)
** Sample node implementation that supports
** Query: sum of the elements in range [a, b)
** Update: add a given value X to every element in range [a, b)
*/

struct StNode {
 using NodeType = StNode;
 using i64 = long long;
 i64 val; // Sum of the interval
 i64 lazy; // Sumation pending to apply to children

 // Used, while creating the tree, to update the Node content according to
 // the value given by the ValueProvider
 void set(const NodeType& from) {
  val = from.val;
  lazy = identity().lazy;
 }

 // Updates the Node content to store the result of the 'merge' operation
 // applied on the children.
 // The tree will always call push_lazy() on the Node *before* calling merge()
 void merge(const NodeType& le, const NodeType& ri) {
  val = le.val + ri.val;
  lazy = identity().lazy;
 }

 // Used to update the Node content in a tree update command
 void update(const NodeType& from) {
  auto new_value = from.val;
  val += (e - s) * new_value;
  lazy += new_value;
 }

 // Pushes any pending lazy updates to children
 void push_lazy(NodeType& le, NodeType& ri) {
  if (lazy == identity().lazy) {
   return;
  }

  le.lazy += lazy;
  le.val += (le.e - le.s) * lazy;

  ri.lazy += lazy;
  ri.val += (ri.e - ri.s) * lazy;

  lazy = identity().lazy;
 }
```

```
  // This function should return a NodeType instance such that calling
  // Y.merge(X, identity()) or Y.merge(identity(), X) for any Node X with no
  // pending updates should make Y match X exactly.
  static NodeType identity() {
    static auto tmp = (NodeType){0, 0};
    return tmp;
  }

  // Internal tree data
  int son[2]; // Children of this node
  int s, e; // Interval [s, e), covered by this node
};

template <class Node>
struct SegmentTree {
  using ValueProvider = function<Node(int)>;
  vector<Node> T;

  SegmentTree(int n, const ValueProvider& vp = [](int pos) {
    return Node::identity();
  }) {
    Node nd;
    nd.son[0] = nd.son[1] = -1;
    nd.s = 0, nd.e = n;

    T.reserve(4 * n);
    T.emplace_back(std::move(nd));

    init(vp, 0);
  }

  void init(const ValueProvider& vp, int u) {
    Node& n = T[u];

    if (n.e - n.s == 1) {
      n.set(vp(n.s));
      return;
    }

    Node le(n), ri(n);

    le.e = (n.s + n.e) / 2;
    n.son[0] = T.size();
    T.emplace_back(std::move(le));
    init(vp, n.son[0]);

    ri.s = le.e;
    n.son[1] = T.size();
    T.emplace_back(std::move(ri));
    init(vp, n.son[1]);

    n.merge(T[n.son[0]], T[n.son[1]]);
  }

  void update(int le, int ri, const Node& val, int u = 0) {
    Node& n = T[u];
    if (le >= n.e || n.s >= ri) return;

    if (n.s == le && n.e == ri) {
      n.update(val);
      return;
    }

    n.push_lazy(T[n.son[0]], T[n.son[1]]);

    update(le, min(T[n.son[0]].e, ri), val, n.son[0]);
    update(max(T[n.son[1]].s, le), ri, val, n.son[1]);
    n.merge(T[n.son[0]], T[n.son[1]]);
  }

  Node query(int le, int ri, int u = 0) {
    Node& n = T[u];
    if (n.e <= le || n.s >= ri) return Node::identity();
    if (n.s == le && n.e == ri) return n;

    n.push_lazy(T[n.son[0]], T[n.son[1]]);

    Node r1, r2, r3;
    r1 = query(le, min(T[n.son[0]].e, ri), n.son[0]);
    r2 = query(max(T[n.son[1]].s, le), ri, n.son[1]);
    r3.merge(r1, r2);
    return r3;
  }
};

/*
** USAGE SAMPLE
** http://www.spoj.com/problems/HORRIBLE/
*/
int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);

  int tc;
  cin >> tc;

  for (int cas = 1; cas <= tc; ++cas) {
    int n, c;
    cin >> n >> c;

    SegmentTree<StNode> st(n);

    for (int i = 0; i < c; ++i) {
      int k, p, q;
      cin >> k >> p >> q;
      p -= 1;
```

```
    if (k == 0) {
      int v;
      cin >> v;
      st.update(p, q, (StNode){v});
    } else {
      auto sum = st.query(p, q).val;
      cout << sum << '\n';
    }
  }
 }

 return 0;
}
```

# 6 Math

## 6.1 Combinations

```
long long comb(int n, int m) {
  if (m > n - m) m = n - m;

  long long C = 1;
  // C^{n}_{i} -> C^{n}_{i+1}
  for (int i = 0; i < m; ++i) C = C * (n - i) / (1 + i);
  return C;
}

// Cuando n y m son grandes y se pide comb(n, m) % MOD,
// donde MOD es un numero primo, se puede usar el Teorema de Lucas.

#define MOD 3571
int C[MOD][MOD];

void FillLucasTable() {
  memset(C, 0, sizeof(C));

  for (int i = 0; i < MOD; ++i) C[i][0] = 1;
  for (int i = 1; i < MOD; ++i) C[i][i] = 1;
  for (int i = 2; i < MOD; ++i)
    for (int j = 1; j < i; ++j) C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % MOD;
}

int comb(int n, int k) {
  long long ans = 1;

  while (n != 0) {
    int ni = n % MOD, ki = k % MOD;
    n /= MOD;
    k /= MOD;
    ans = (ans * C[ni][ki]) % MOD;
  }

  return (int)ans;
}
```

## 6.2 Chinese Remainer Theorem

```
// rem y mod tienen el mismo numero de elementos
long long chinese_remainder(vector<int> rem, vector<int> mod) {
  long long ans = rem[0], m = mod[0];
  int n = rem.size();

  for (int i = 1; i < n; ++i) {
    int a = modular_inverse(m, mod[i]);
    int b = modular_inverse(mod[i], m);
    ans = (ans * b * mod[i] + rem[i] * a * m) % (m * mod[i]);
    m *= mod[i];
  }

  return ans;
}
```

## 6.3 Deterministic Miller Rabin

```
/*
** Deterministic Miller-Rabin
** if n < 3,825,123,056,546,413,051, it is enough to test
** a = 2, 3, 5, 7, 11, 13, 17, 19, and 23.
*/
#include <bits/stdc++.h>
using namespace std;
typedef unsigned long long ll;

vector<ll> mr_values({2, 3, 5, 7, 11, 13, 17, 19, 23});

ll mulmod(ll a, ll b, ll n) {
  ll erg = 0;
  ll r = 0;
  while (b > 0) {
    // unsigned long long gives enough room for base 10 operations
    ll x = ((a % n) * (b % 10)) % n;
```

```
    for (ll i = 0; i < r; i++) x = (x * 10) % n;
    erg = (erg + x) % n;
    r++;
    b /= 10;
  }
  return erg;
}

ll fastexp(ll a, ll b, ll n) {
  if (b == 0) return 1;
  if (b == 1) return a % n;
  ll res = 1;
  while (b > 0) {
    if (b % 2 == 1) res = mulmod(a, res, n);
    a = mulmod(a, a, n);
    b /= 2;
  }
  return res;
}

bool mrtest(ll n) {
  if (n == 1) return false;
  ll d = n - 1;
  ll s = 0;
  while (d % 2 == 0) {
    s++;
    d /= 2;
  }
  for (ll j = 0; j < (ll)mr_values.size(); j++) {
    if (mr_values[j] > n - 1) continue;
    ll ad = fastexp(mr_values[j], d, n);
    if (ad % n == 1) continue;
    bool notcomp = false;
    for (ll r = 0; r <= max(0ull, s - 1); r++) {
      ll rr = fastexp(2, r, n);
      ll ard = fastexp(ad, rr, n);
      if (ard % n == n - 1) {
        notcomp = true;
        break;
      }
    }
    if (!notcomp) {
      return false;
    }
  }
  return true;
}

bool isprime(ll n) {
  if (n <= 1) return false;
  if (n == 2) return true;
  if (n % 2 == 0) return false;
  return mrtest(n);
}
```

## 6.4 Big Integer

```
string trim_zeros(const string& a) {
  size_t idx = 0;
  while (a[idx] == '0' && idx < a.size()) idx++;
  if (idx == a.size()) idx--;

  return a.substr(idx);
}

string big_sub(const string& n1, const string& n2) {
  string a = trim_zeros(n1);
  string b = trim_zeros(n2);

  bool minus = false;
  if (esMayor(b, a)) {
    swap(a, b);
    minus = true;
  }

  int i, j, d = (a.length() - b.length());
  for (i = b.length() - 1; i >= 0; i--) {
    if (a[i + d] >= b[i])
      a[i + d] -= b[i] - '0';
    else {
      j = -1;
      while (a[i + d + j] == '0') {
        a[i + d + j] = '9';
        j--;
      }
      a[i + d + j]--;

      a[i + d] += 10 - b[i] + '0';
    }
  }

  return (minus ? "-" : "") + trim_zeros(a);
}

string big_add(const string& a, const string& b) {
  int LA = a.size(), LB = b.size(), L = max(LA, LB), carry = 0;

  string x = string(L, '0');

  while (L--) {
    LA--;
    LB--;
```

```
    if (LA >= 0) carry += a[LA] - '0';
    if (LB >= 0) carry += b[LB] - '0';

    if (carry < 10)
      x[L] = '0' + carry, carry = 0;
    else
      x[L] = '0' + carry - 10, carry = 1;
  }

  if (carry) x = '1' + x;
  return x;
}

string big_mult(string a, string b) {
  if (a == "0" || b == "0")
    return "0";
  else if (a.size() == 1) {
    int m = a[0] - '0';

    string ans = string(b.size(), '0');

    int lleva = 0;

    for (int i = b.size() - 1; i >= 0; i--) {
      int d = (b[i] - '0') * m + lleva;
      lleva = d / 10;
      ans[i] += d % 10;
    }
    if (lleva) ans = (char)(lleva + '0') + ans;
    return ans;
  } else if (b.size() == 1)
    return big_mult(b, a);
  else {
    string ans = "0";
    string ceros = "";
    for (int i = a.size() - 1; i >= 0; i--) {
      string s = big_mult(string(1, a[i]), b) + ceros;
      ceros += "0";
      ans = big_add(ans, s);
    }
    return ans;
  }
}
```

## 6.5 Factorial Prime Factors

```
vector<int> primes; // Filled with prime numbers <= n (at least)

void factorial_prime_factor(const int n, vector<int>& v) {
  v.clear();
  for (size_t i = 0; primes[i] <= n && i < primes.size(); i++) {
    const int& p = primes[i];
    double q = (n / (double)p);
    int d = int(q);

    while (q >= p) {
      q /= p;
      d += int(q);
    }

    v.push_back(d);
  }
}
```

## 6.6 Extended Gcd

```
// a*x + b*y = gcd(a,b)
int extGcd(int a, int b, int &x, int &y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }

  int g = extGcd(b, a % b, y, x);
  y -= a / b * x;
  return g;
}

// ASSUME: gcd(a, m) == 1
int modInv(int a, int m) {
  int x, y;
  extGcd(a, m, x, y);
  return (x % m + m) % m;
}
```

## 6.7 Least Significant Bit Position

```
// http://supertech.csail.mit.edu/papers/debruijn.pdf
int lsbpos(unsigned int v) {
  static const int t[32] = {0, 1, 28, 2, 29, 14, 24, 3, 30, 22, 20,
                            15, 25, 17, 4, 8, 31, 27, 13, 23, 21, 19,
                            16, 7, 26, 12, 18, 6, 11, 5, 10, 9};
```

```
  return t[((unsigned int)((v & -v) * 0x077CB531U)) >> 27];
}
```

# 7 General

## 7.1 Longest Increasing Subsequence

```
#include <vector>
/* Finds longest strictly increasing subsequence. O(n log k) algorithm. */
void find_lis(vector<int> &a, vector<int> &b) {
  vector<int> p(a.size());
  int u, v;

  if (a.empty()) return;

  b.push_back(0);
  for (size_t i = 1; i < a.size(); i++) {
    // If next element a[i] is greater than last element of current longest
    // subsequence a[b.back()], just push it at back of "b" and continue
    if (a[b.back()] < a[i]) {
      p[i] = b.back();
      b.push_back(i);
      continue;
    }
    // Binary search to find the smallest element referenced by b which is just
    // bigger than a[i]
    // Note : Binary search is performed on b (and not a). Size of b is always
    // <= k and hence contributes O(log k) to complexity.
    for (u = 0, v = b.size() - 1; u < v;) {
      int c = (u + v) / 2;
      if (a[b[c]] < a[i])
        u = c + 1;
      else
        v = c;
    }
    // Update b if new value is smaller then previously referenced value
    if (a[i] < a[b[u]]) {
      if (u > 0) p[i] = b[u - 1];
      b[u] = i;
    }
  }
  for (u = b.size(), v = b.back(); u--; v = p[v]) b[u] = v;
}
```

## 7.2 Inversion Counting

```
int v[MAX], sortedV[MAX];

// Merge sort with inversion counting
long long mergeSort(int *V, int lo, int hi) {
  if (lo >= hi) {
    return 0;
  } else {
    int m1 = (lo + hi) / 2, m2 = m1 + 1;
    long long r = 0, rA, rB;
    int i = lo, j = m2, k = 0;

    rA = mergeSort(V, lo, m1);
    rB = mergeSort(V, m2, hi);

    while (i <= m1 && j <= hi) {
      if (V[j] < V[i]) {
        r += (m1 - i + 1);
        sortedV[k++] = V[j++];
      } else {
        sortedV[k++] = V[i++];
      }
    }

    if (i > m1) {
      i = j;
      j = hi;
    } else {
      j = m1;
    }

    while (i <= j) {
      sortedV[k++] = V[i++];
    }

    memcpy(V + lo, sortedV, (hi - lo + 1) * sizeof(int));

    return r + rA + rB;
  }
}
```