Universidad Rey Juan Carlos

MilkTeam

Isaac Lozano Osorio, Jakub Jan Luczyn, Raúl Martín Santamaría

SWERC 2018

December 1, 2018

Contest (1)

content.txt

10 lines

```
Contest (1)
Data structures (1)
Graph (2)
Strings (9)
Mathematics (11)
Geometry (12)
Numerical (17)
Number theory (18)
Combinatorial (19)
Various (20)
```

template.cpp

18 lines

```cpp
#include <bits/stdc++.h>
#define oo 0x3f3f3f3f3f3f3f3fLL
using namespace std;

typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

#define FOR(i, a, b) for(int i = (a); i < int(b); i++)
#define trav(i, v) for(auto &i : v)
#define has(c, e) ((c).find(e) != (c).end())
#define sz(c) int((c).size())
#define all(c) c.begin(), c.end()
#define debug(x) cerr << #x << ": " << x << endl;

int main() {
  return 0;
}
```

troubleshoot.txt

56 lines

```
Pre-submit:
Write a few simple test cases, if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all datastructures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Wrong copy code?
Any uninitialized variables?
Any overflows?
Same variable name?
Correct recursion?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a team mate.
Ask the team mate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a team mate do it.

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).

Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
Use vector? Change to array.
What do your team mates think about your algorithm?

Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all datastructures between test cases?
```

Data structures (2)

SegmentTree.cpp

Description: Data structure used for storing information about intervals, or segments.
Time: buildTree $\mathcal{O}(N)$ - query and updated $\mathcal{O}(log(n))$

80 lines

```cpp
struct node {
   long long int pa;
   long long int pc;
};
node join(node left,node right)
{
    node res;
    int n = min(left.pa, right.pc);
    res.pa = right.pa + left.pa - n;
    res.pc = left.pc + right.pc - n;
    return res;
}
void buildTree(node *tree,string a,int index,int s,int e)
{
    //base case
    if(s>e) return;
    //reached leaf node
    if(s==e)
    {
        if(a[s]==')') { tree[index].pc=1; tree[index].pa=0;} //
             Base value
        else { tree[index].pa=1;tree[index].pc=0; }
        return ;
    }
    int m = (s+e)/2;
    buildTree(tree,a,2*index,s,m);
    buildTree(tree,a,2*index+1,m+1,e);
    node left = tree[2*index]; node right = tree[2*index+1];
    tree[index]=join(left,right);
  // return left*right;
    return;
}
node query(node *tree,int index,int s,int e,int qs,int qe)
{
    //base case: if query range is outside the node range
    if(qs>e || s>qe) return node{INF,INF}; //Base value
    //complete overlap
    if(s>=qs && e<=qe) return tree[index];
    //now partial overlap case is executed
    int m = (s+e)/2;
    node left = query(tree,2*index,s,m,qs,qe);
    node right = query(tree,2*index+1,m+1,e,qs,qe);
  // return left*right;
    return join(left,right);
}

void updateNode(node *tree,int index,int s,int e,int pos)
{
    if(pos<s || pos>e) return ;
    if(s==e)
    {
        if(tree[index].pc==1) { tree[index].pc=0; tree[index].
             pa=1; }
        else { tree[index].pc=1; tree[index].pa=0; }
        return;
    }
    int m = (s+e)/2;
    updateNode(tree,2*index,s,m,pos);
    updateNode(tree,2*index+1,m+1,e,pos);
    tree[index] = join(tree[2*index],tree[2*index+1]);
    return;
}
int main()
{
    int n; scanf("%d",&n);
  string a;
  node*tree = new node[4*n+1]; //array to store the segment
       tree
  int index = 1; //index of 1st node
  int s =0,e=n-1;
  buildTree(tree,a,index,s,e);//now tree has been built
  scanf("%d",&c);
  for(int i=0; i<c;i++){
    int f; scanf("%d",&f);
    if(f==0)
    {
      node aux = query(tree,index,s,e,0,n);
      if(aux.pa==0 && aux.pc==0) printf("YES\n");
      else printf("NO\n");
    }
    else updateNode(tree,index,s,e,f-1);}
    return 0;
}
```

BIT.cpp

Description: Used to store cumulative frequencies and manipulating cumulative frequency table.
Time: $\mathcal{O}(log(n))$

42 lines

```cpp
long long bit[MAX];

long long query(int indx){
    long long sum = 0;
    while (indx) {
        sum += bit[indx];
        indx -= (indx & -indx);
    }
    return sum;
}

void update(int indx, int x){
   while (indx < MAX) {
        bit[indx] += x;
        indx += (indx & -indx);
    }
}

int main() {
    int n;
    while(scanf("%d",&n)==1){
        long long a[n+1];
        for (int i = 1; i <= n; i++) { //Begin with 1
            scanf("%lld",&a[i]);
            update(i, a[i]);
        }
        int q; scanf("%d",&q);
        while (q--) {
            string choice;
            cin >> choice;
            if (choice == "q") {
                int l, r;
                scanf("%d%d",&l,&r);
                printf("%lld\n",query(r) - query(l-1));
            } else {
                int p; long long int x;
                scanf("%d%lld",&p,&x);
                update(p, x);
            }
        }
    }
}
```

BIT2D.java

Description: 2d variant of the Fenwick tree.
Usage: Update value at (row,col), query rectangle (1,1) to (row,col). Also useful with some hashing problems.
Time: $\mathcal{O}(log(n) * log(n))$ 34 lines

```
public static void main(String[] args) throws IOException {
   FT2D ft = new FT2D(rows, cols);
   ft.sum(toRowPos, toColPos);
   ft.update(rowPos, colPos, diff);
}

public static class FT2D {
  public long ROWS,COLS;
  long[][] tree;
  public FT2D(int rows, int cols) {
    this.ROWS = rows+3;
    this.COLS = cols+3;
    this.tree = new long[rows+3][cols+3];
  }
  public long sum(int row, int col){
    long r = 0;
    for (int i=row+1; i>0; i-= (i&-i)) {
      for (int j = col+1; j>0; j-= (j&-j)) {
        r+=tree[i][j];
        //*= for mul, also init at 1
        //^=for XOR
      }
    }
    return r;
  }
  public void update(int row, int col, long diff){
    if(diff==0)return;
    for (int i=row+1; i<ROWS; i+= (i&-i)) {
      for (int j = col+1; j < COLS; j+= (j&-j)) {
        tree[i][j]+=diff;
      }
    }
  }
}
```

Trie.cpp

Description: Keys Tree of any alphabet
Time: Search - Insert $\mathcal{O}(M)$ M = String length 63 lines

```
const int put = 26; //alphabet size
struct TrieNode
{
    struct TrieNode *children[put];
    //bool isEndOfWord; check end of word
    int num;
};
struct TrieNode *getNode(void)
{
    struct TrieNode *pNode =  new TrieNode;
    //pNode->isEndOfWord = false;
    pNode->num=0;
    for (int i = 0; i < put; i++)  pNode->children[i] = NULL;
    return pNode;
}
void insert(struct TrieNode *root, string key)
{
    struct TrieNode *pCrawl = root;

    for (unsigned int i = 0; i < key.length(); i++)
    {
        int index = key[i] - 'a';
        if (!pCrawl->children[index]) pCrawl->children[index] =
             getNode();
        pCrawl->num++; //add to get repetitions
        pCrawl = pCrawl->children[index];
    }
    pCrawl->num++;
    //pCrawl->isEndOfWord = true;
}
int searchWord(char key,int a)
{
   int index = key - '0';
   if (!auxNode->children[index]) return -1;
   auxNode = auxNode->children[index];
   if(auxNode->isEndOfWord) auxNode->num++;
   if (a==1) auxNode2=auxNode;
   return 0;
} //Check if the word exist. Move with pointers
  //DONT FORGET RESTORE DE POINTER TO THE BEGIN
int search(struct TrieNode *root, string key)
{
    struct TrieNode *pCrawl = root;
    for (unsigned int i = 0; i < key.length(); i++)
    {
        int index = key[i] - 'a';
        if (!pCrawl->children[index]) return 0;
        pCrawl = pCrawl->children[index];
    }
    return pCrawl->num;
}

int main()
{
    int n,m;
    while(scanf("%d%d",&n,&m)==2)
    {
    string word;
    struct TrieNode *root = getNode();
    for(int i=0; i<n;i++) {cin>>word; insert(root,word);}
    for(int i=0; i<m;i++) {cin>>word; cout<<search(root,word)<<
         '\n';}
    }
    return 0;
}
```

Graph (3)

ArticulationPointAndBridges.cpp

Description: Finds all articulation points and bridges of a undirected graph.
Time: $\mathcal{O}(E+V)$ 69 lines

```
void ArticulationPointandBridges(vector<vi>& adj, vector<Edge>
     E, vi& articulation, vi& bridges) {
    int n = sz(adj), m = sz(E);
    articulation = vi(n, -1);
    bridges = vi(m, -1);
    vi dfs_low(n, -1), dfs_num(n ,-1), parent(n, -1);
    articulation = vi(n, 0);
    bridges = vi(m, 0);
    int dfsroot, rootc, cnt = 0;
    function<void(int)> ABdfs = [&](int u) {
        dfs_num[u] = dfs_low[u] = cnt++;
        for(int t : adj[u]){
            int v = E[t].u^E[t].v^u;
            if(dfs_num[v] == -1){
                if(u == dfsroot) rootc++;
                parent[v] = t;
                ABdfs(v);
                dfs_low[u] = min(dfs_low[u], dfs_low[v]);
                if(dfs_low[v] >= dfs_num[u])articulation[u] =
                     1;
                if(dfs_low[v] > dfs_num[u]) bridges[t] = 1;
            } else if (t != parent[u]){
                dfs_low[u] = min(dfs_low[u], dfs_num[v]);
            }
        }
    };
    FOR(i, 0, n) {
        if(dfs_num[i] == -1) {
            rootc = 0, dfsroot = i;
            ABdfs(i);
            if (rootc <= 1) articulation[i] = 0;
        }
    }
}

int main() {
    int n,m;
    while(scanf("%d",&n)==1)
    {
        vector<Edge> E;
        vector<vector<int>> adj(n+1);
        vi articulation;
        vi bridge;
        bool connected[n+1][n+1];
        memset(connected,false, sizeof(connected));
        for(int i=0; i<n;i++)
        {
            int x,y; scanf("%d (%d)",&x,&m);
            for(int j=0; j<m;j++)
            {
                scanf("%d",&y);
                if(connected[x][y] || connected[y][x] )
                    continue;
                connected[x][y] = true;
                E.push_back({x,y});
                adj[x].push_back(E.size()-1);
                adj[y].push_back(E.size()-1);
            }
        }
        ArticulationPointandBridges(adj,E,articulation,bridge);

        int cnt = 0;
        vector<Edge> resu;
        for(unsigned i=0;i<bridge.size();i++)
            if(bridge[i]==1) { cnt++; resu.push_back(reverse(E[
                i]));}
        printf("%d critical links\n",cnt);
        sort(resu.begin(),resu.end(),cmp);
        for(int i=0; i<cnt;i++)   printf("%d - %d\n", resu[i].u
            ,resu[i].v);
        cin.ignore();  printf("\n"); }
    return 0;
}
```

EulerianCycle.cpp

Description: returns de eulerian cycle/tour starting at u. If its a tour it must start at a vertex with odd degree. It is common to add edges between odd vertex to find a pseudo euler tour. Start and end in same vertex and visit each edge exactly once. All vertex degree must be even.
Usage: adj should contain index of edge in the vector<edge>, if undirected add index to both rows of adj list. If directed make sure if it needs to be connected, difference between in/out edges. If it is a tour then u must be a vertex with odd degree, else it can be any edge.
Time: $\mathcal{O}(E)$ 64 lines

```
struct edge{
  int u, v;
  bool used;
};

void Eulerdfs(int u, vi &nxt, vi &Euler, vector<edge> &E, const
      vector<vi> &adj) {
  while(nxt[u] < adj[u].size()){
    int go = adj[u][nxt[u]++];
    if(!E[go].used){
      E[go].used = 1;
      int to = (E[go].u ^ E[go].v ^ u);
      Eulerdfs(to, nxt, Euler, E, adj);
    }
  }
  Euler.push_back(u);
}

vi Eulerian(int u, vector<edge> &E, const vector<vi> &adj) {
  vi nxt (adj.size(), 0); vi Euler;
  Eulerdfs(u, nxt, Euler, E, adj);
  reverse(Euler.begin(), Euler.end());
  return Euler;
}

int main()
{
    int cases,ca=1;
    scanf("%d",&cases);
    while(cases--)
    {
        int n; scanf("%d",&n);
        vector<edge> E;
        vector<vector<int>> adj(51);
        vector<int> degree(51);
        for(int i=0; i<n;i++)
        {
            int x,y; scanf("%d%d",&x,&y);
            E.push_back({x,y,false});
            adj[x].push_back(E.size()-1);
            adj[y].push_back(E.size()-1);
            degree[x - 1]++;
            degree[y - 1]++;
        }
        printf("Case #%d\n",ca++);
        bool good = true;
        int start = -1;
        for (int i = 0; i < 50; i++) {
            if (degree[i] % 2) { //All degree must be even
                good = false;
                break;
            }
            else if (degree[i] && start == -1)
                start = i;
        }
        if(!good) printf("some beads may be lost\n");
        else{
        vector<int> res = Eulerian(start+1,E,adj);
        for(int i=0; i<sz(res)-1;i++) printf("%d %d\n",res[i],
            res[(i+1)%sz(res)]);
        }
        printf("\n");
    }
    return 0;
}
```

EulerianPath.cpp

Description: Same as Eulerian Cycle but with any end vertex.
Time: $\mathcal{O}(E)$ 35 lines

```
struct Edge;
typedef list<Edge>::iterator iter;
struct Edge
{
  int next_vertex;
  iter reverse_edge;
    Edge(int next_vertex) :next_vertex(next_vertex) { }
};

const int max_vertices = ;
int num_vertices;
list<Edge> adj[max_vertices]; // adjacency list
vector<int> path;

void find_path(int v)
{
  while(adj[v].size() > 0)
  {
    int vn = adj[v].front().next_vertex;
        adj[vn].erase(adj[v].front().reverse_edge);
        adj[v].pop_front();
        find_path(vn);
    }
    path.push_back(v);
}

void add_edge(int a, int b)
{
  adj[a].push_front(Edge(b));
  iter ita = adj[a].begin();
  adj[b].push_front(Edge(a));
  iter itb = adj[b].begin();
  ita->reverse_edge = itb;
  itb->reverse_edge = ita;
}
```

StronglyConnectedComponents.cpp

Description: Finds strongly connected components (Every node can go to every other node in the component).
Time: $\mathcal{O}(E + V)$ 58 lines

```
#define MAXE 650
#define MAXV 26
struct edge {
        int e, nxt;
    };
int V;
vector<vector<int> > edges, edges_reverse;

int group_cnt, group_num[MAXV];
bool visited[MAXV];
int stk[MAXV];
void fill_forward(int x) {
    visited[x] = true;
    for (int i = 0; i < edges[x].size(); i++)
        if (!visited[edges[x][i]])
            fill_forward(edges[x][i]);
    stk[++stk[0]] = x;
}
void fill_backward(int x) {
    visited[x] = false;
    group_num[x] = group_cnt;
    for (int i = 0; i < edges_reverse[x].size(); i++)
        if (visited[edges_reverse[x][i]])
            fill_backward(edges_reverse[x][i]);
}
void SCC(int number_of_nodes) {
    V = number_of_nodes;
    edges = vector<vector<int> >(V + 1);
    edges_reverse = vector<vector<int> >(V + 1);
    group_cnt = -1;
}

// add edge v1->v2
void add_edge(int v1, int v2) {
    edges[v1].push_back(v2);
    edges_reverse[v2].push_back(v1);
}
void run() {
    int i;
    stk[0] = 0;
    memset(visited, false, sizeof(visited));
    for (i = 1; i <= V; i++)
        if (!visited[i])
            fill_forward(i);
    group_cnt = 0;
    for (i = stk[0]; i >= 1; i--)
        if (visited[stk[i]]) {
            group_cnt++;
            fill_backward(stk[i]);
        }
}

int main()
{
  SCC();
  run();
}
```

Hopcroft-Karp.cpp

Description: Solves the bipartite maching problem
Time: $\mathcal{O}\left(\sqrt{V} \cdot E\right)$ 91 lines

```
const int SINK = 50001;
const int MAX_N = 50005;
const int INF = 0x3f3f3f3f;

int N, M, P;
int pair_g1[MAX_N], pair_g2[MAX_N], dist[MAX_N];
list< int > adj[MAX_N];
```

```
bool bfs() {
    queue< int > q; // queue of G1 nodes

    for (int v = 0; v < N; ++v) {
        if (pair_g1[v] == SINK) {
            dist[v] = 0;
            q.push(v);
        }
        else {
            dist[v] = INF;
        }
    }
    dist[SINK] = INF;

    while (!q.empty()) {
        int v = q.front();
        q.pop();

        if (dist[v] < dist[SINK]) {
            for (list< int >::iterator it = adj[v].begin(); it
                 != adj[v].end(); ++it) {
                int u = *it;

                if (dist[pair_g2[u]] == INF) {
                    dist[pair_g2[u]] = dist[v] + 1;
                    q.push(pair_g2[u]);
                }
            }
        }
    }
    return dist[SINK] != INF;
}

bool dfs(int u) {
    // DFS starting at a node u in G1
    if (u == SINK) {
        // we've reached a free node in G2 (i.e. one pointing
            to SINK)
        return true;
    }
    for (list< int >::iterator it = adj[u].begin(); it != adj[u
         ].end(); ++it) {
        int v = *it;

        if (dist[pair_g2[v]] == dist[u] + 1) {
            if (dfs(pair_g2[v])) {
                pair_g1[u] = v;
                pair_g2[v] = u;
                return true;
            }
        }
    }
    dist[u] = INF;
    return false;
}
int hopcroft_karp() {
    int matching = 0;

    for (int i = 0; i < N; ++i) {
        pair_g1[i] = SINK;
    }
    for (int j = 0; j < M; ++j) {
        pair_g2[j] = SINK;
    }

    while (bfs()) {
        for (int i = 0; i < N; ++i) {
            if (pair_g1[i] == SINK) {
                matching += dfs(i);
            }
        }
    }
    return matching;
}
int main() {
    int a,b;
    while (scanf("%d %d %d", &N, &M, &P) == 3) {
        for (int i = 0; i < N; i++) adj[i].clear();
        for (int i = 0; i < P; i++) {
            scanf("%d%d",&a,&b);
            adj[a-1].push_back(b-1);
        }
        printf("%d\n", hopcroft_karp());
    }
    return 0;
}
```

Dinic.cpp

Description: Find MaxFlow
Time: $\mathcal{O}\left(V^2E\right)$ 62 lines

```
struct Node {
    int x, y, v;// x->y, v
    int next;
} edge[50005];
int e, head[1005], dis[1005], previ[1005], record[1005];
void addEdge(int x, int y, int v) {
    edge[e].x = x, edge[e].y = y, edge[e].v = v;
    edge[e].next = head[x], head[x] = e++;
    edge[e].x = y, edge[e].y = x, edge[e].v = 0;
    edge[e].next = head[y], head[y] = e++;
}
int maxflow(int s, int t) {
    int flow = 0;
    int i, j, x, y;
    while(1) {
        memset(dis, 0, sizeof(dis));
        dis[s] =  0xffff; // oo
        queue<int> Q;
        Q.push(s);
        while(!Q.empty()) {
            x = Q.front();
            Q.pop();
            for(i = head[x]; i != -1; i = edge[i].next) {
                y = edge[i].y;
                if(dis[y] == 0 && edge[i].v > 0) {
                    previ[y] = x;
                    record[y] = i;
                    dis[y] = std::min(dis[x], edge[i].v);
                    Q.push(y);
                }
            }
            if(dis[t])  break;
        }
        if(dis[t] == 0) break;
        flow += dis[t];
        for(x = t; x != s; x = previ[x]) {
            int ri = record[x];
            edge[ri].v -= dis[t];
            edge[ri^1].v += dis[t];
        }
    }
    return flow;
}

int main() {
    int n, i=1;
    while(scanf("%d", &n) && n!=0) {
        e=0;
        memset(head, -1, sizeof(head));
        int s, t, c, x, y, p;
        cin >> s >> t >> c;
        while(c--) {
            cin >> x >> y >> p;
            addEdge(x, y, p);
            addEdge(y, x, p);
        }
        int flow = maxflow(s, t);
        cout << "Network " << i << "\nThe bandwidth is " <<
             flow <<".\n\n";
        i++;
    }
    return 0;
}
```

PushRelabel.cpp

Description: Maximum flow (zero capacity edges are residual edges). Solves problems 10000 vertices and 1000000 edges, worst-case has very few edges.
Time: $\mathcal{O}\left(V^3\right)$ 105 lines

```
typedef long long LL;

struct Edge {
  int from, to, cap, flow, index;
  Edge(int from, int to, int cap, int flow, int index) :
    from(from), to(to), cap(cap), flow(flow), index(index) {}
};

struct PushRelabel {
  int N;
  vector<vector<Edge> > G;
  vector<LL> excess;
  vector<int> dist, active, count;
  queue<int> Q;

  PushRelabel(int N) : N(N), G(N), excess(N), dist(N), active(N
      ), count(2*N) {}

  void AddEdge(int from, int to, int cap) {
    G[from].push_back(Edge(from, to, cap, 0, G[to].size()));
    if (from == to) G[from].back().index++;
    G[to].push_back(Edge(to, from, 0, 0, G[from].size() - 1));
  }

  void Enqueue(int v) {
    if (!active[v] && excess[v] > 0) { active[v] = true; Q.push
        (v); }
  }

  void Push(Edge &e) {
    int amt = int(min(excess[e.from], LL(e.cap - e.flow)));
    if (dist[e.from] <= dist[e.to] || amt == 0) return;
    e.flow += amt;
    G[e.to][e.index].flow -= amt;
    excess[e.to] += amt;
    excess[e.from] -= amt;
    Enqueue(e.to);
  }

  void Gap(int k) {
    for (int v = 0; v < N; v++) {
      if (dist[v] < k) continue;
      count[dist[v]]--;
      dist[v] = max(dist[v], N+1);
      count[dist[v]]++;
      Enqueue(v);
    }
  }

  void Relabel(int v) {
    count[dist[v]]--;
    dist[v] = 2*N;
    for (int i = 0; i < G[v].size(); i++)
      if (G[v][i].cap - G[v][i].flow > 0)
  dist[v] = min(dist[v], dist[G[v][i].to] + 1);
    count[dist[v]]++;
    Enqueue(v);
  }

  void Discharge(int v) {
    for (int i = 0; excess[v] > 0 && i < G[v].size(); i++) Push
         (G[v][i]);
    if (excess[v] > 0) {
      if (count[dist[v]] == 1)
  Gap(dist[v]);
      else
  Relabel(v);
    }
  }

  LL GetMaxFlow(int s, int t) {
    count[0] = N-1;
    count[N] = 1;
    dist[s] = N;
    active[s] = active[t] = true;
    for (int i = 0; i < G[s].size(); i++) {
      excess[s] += G[s][i].cap;
      Push(G[s][i]);
    }

    while (!Q.empty()) {
      int v = Q.front();
      Q.pop();
      active[v] = false;
      Discharge(v);
    }

    LL totflow = 0;
    for (int i = 0; i < G[s].size(); i++) totflow += G[s][i].
         flow;
    return totflow;
  }
};

int main() {
  int n, m;
  scanf("%d%d", &n, &m);
  //FASTFLOW
  PushRelabel pr(n);
  for (int i = 0; i < m; i++) {
   int a, b, c;
    scanf("%d%d%d", &a, &b, &c);
    if (a == b) continue;
    pr.AddEdge(a-1, b-1, c);
    pr.AddEdge(b-1, a-1, c);
  }
  printf("%Ld\n", pr.GetMaxFlow(0, n-1));
  return 0;
}
```

MaxFlow.cpp

Description: Returns maximum flow.
Usage: To obtain a cut in the mincut problem one must bfs from the source. All the vertices reached from it using only edges with CAP > 0 are in the same cut
Time: $\mathcal{O}(V^2 * E)$ for general graphs. For unit capacities $\mathcal{O}\left(min(V^(2/3), E^(1/2)) * E\right)$. For maximum matching $\mathcal{O}(E * sqrt(V))$)(bipartite unit weighted graf). It is generally very fast.

62 lines

```
typedef long long ll;
typedef vector<int> vi;
const ll INF = 1000000000000000000LL;

#define VEI(w,e) ((E[e].u == w) ? E[e].v : E[e].u)
#define CAP(w,e) ((E[e].u == w) ? E[e].cap[0] - E[e].flow : E[e
     ].cap[1] + E[e].flow)
#define ADD(w,e,f) E[e].flow += ((E[e].u == w) ? (f) : (-(f)))

struct Edge { int u, v; ll cap[2], flow; };

vi d, act;

bool bfs(int s, int t, vector<vi>& adj, vector<Edge>& E) {
  queue<int> Q;
  d = vi(adj.size(), -1);
  d[t] = 0; Q.push(t);
  while (not Q.empty()) {
    int u = Q.front(); Q.pop();
    for (int i = 0; i < int(adj[u].size()); ++i) {
      int e = adj[u][i], v = VEI(u, e);
      if (CAP(v, e) > 0 and d[v] == -1) {
        d[v] = d[u] + 1;
        Q.push(v);
      }
    }
  }
  return d[s] >= 0;
}

ll dfs(int u,int t,ll bot,vector<vi>& adj,vector<Edge>& E) {
  if (u == t) return bot;
  for (; act[u] < int(adj[u].size()); ++act[u]) {
    int e = adj[u][act[u]];
    if (CAP(u, e) > 0 and d[u] == d[VEI(u, e)] + 1) {
      ll inc=dfs(VEI(u,e),t,min(bot,CAP(u,e)),adj,E);
      if (inc) {
        ADD(u, e, inc);
        return inc;
      }
    }
  }
  return 0;
}

ll maxflow(int s, int t, vector<vi>& adj, vector<Edge>& E) {
  for (int i=0; i<int(E.size()); ++i) E[i].flow = 0;
  ll flow = 0, bot;
  while (bfs(s, t, adj, E)) {
    act = vi(adj.size(), 0);
    while ((bot = dfs(s,t,INF, adj, E))) flow += bot;
  }
  return flow;
}

void addEdge(int u, int v, Vvi& adj, vector<Edge>& E, ll cap){
  Edge e; e.u = u; e.v = v;
  e.cap[0] = cap; e.cap[1] = 0;
  e.flow = 0;
  adj[u].push_back(E.size());
  adj[v].push_back(E.size());
  E.push_back(e);
}
```

MaxflowMinCap.cpp

Description: Normal maxflow but with minimum capacity in each edge.
Usage: Need to copy maxflow, pass source sink adj matrix, Edge matrix and choose scaling if needed
Time: $\mathcal{O}(V^2 * E)$ for general graphs. For unit capacities $\mathcal{O}\left(min(V^(2/3), E^(1/2)) * E\right)$

"FastMaxFlow.h" 49 lines

```
struct Edge { int u, v; ll cap[2], mincap, flow; };

void addEdge(int x, int y, ll c, ll m, vector<vi>& adj, vector<
     Edge>& E) {
    Edge e; e.u = x; e.v = y;
    e.cap[0] = c - m; e.cap[1] = 0; e.mincap = m;
    adj[x].push_back(E.size()); adj[y].push_back(E.size());
    E.push_back(e);
}

ll mincap(int s, int t, vector<vi>& adj, vector<Edge>& E, int F
     = 0) {
    int n = adj.size();
    int m = E.size();
    vector<ll> C(n, 0);
    for (int i = 0; i < m; ++i) {
        C[E[i].u] -= E[i].mincap;
        C[E[i].v] += E[i].mincap;
    }
    adj.push_back(vi(0));
    adj.push_back(vi(0));
    ll flowsat = 0;
    for (int i = 0; i < n; ++i) {
        if (C[i] > 0) {
            addEdge(n, i, C[i], 0, adj, E);
            flowsat += C[i];
        }
        else if (C[i] < 0) addEdge(i, n + 1, -C[i], 0, adj, E);
    }
    addEdge(t, s, oo, 0, adj, E);
    for (int i = 0; i < (int)E.size(); ++i) E[i].flow = 0;
    if (flowsat != maxflow(n, n + 1, adj, E, F)) return -1;
    maxflow(s, t, adj, E, F);
    while ((int)E.size() > m) E.pop_back();
    adj.pop_back();
    adj.pop_back();
    for (int i = 0; i < n; ++i) {
        int j = (int)adj[i].size() - 1;
        while (j >= 0 and adj[i][j] >= m) {
            --j;
            adj[i].pop_back();
        }
    }
    ll flow = 0;
    for (int i = 0; i < m; ++i) {
        E[i].flow += E[i].mincap;
        if (E[i].u == s) flow += E[i].flow;
        else if (E[i].v == s) flow -= E[i].flow;
    }
    return flow;
}
```

MinCostMaxFlow.cpp

Description: - Pair of (maximum flow value, minimum cost value). Look at positive values only. Use adjacency matrix. For a regular max flow, set all edge costs to 0.
Time: $\mathcal{O}\left(V^4 * MAX_EDGE_COST\right)$

104 lines

```
typedef vector<int> VI;
typedef vector<VI> VVI;
typedef long long L;
```

```cpp
typedef vector<L> VL;
typedef vector<VL> VVL;
typedef pair<int, int> PII;
typedef vector<PII> VPII;

const L INF = numeric_limits<L>::max() / 4;

struct MinCostMaxFlow {
  int N;
  VVL cap, flow, cost;
  VI found;
  VL dist, pi, width;
  VPII dad;

  MinCostMaxFlow(int N) :
    N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N, VL(N)),
    found(N), dist(N), pi(N), width(N), dad(N) {}

  void AddEdge(int from, int to, L cap, L cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
  }

  void Relax(int s, int k, L cap, L cost, int dir) {
    L val = dist[s] + pi[s] - pi[k] + cost;
    if (cap && val < dist[k]) {
      dist[k] = val;
      dad[k] = make_pair(s, dir);
      width[k] = min(cap, width[s]);
    }
  }

  L Dijkstra(int s, int t) {
    fill(found.begin(), found.end(), false);
    fill(dist.begin(), dist.end(), INF);
    fill(width.begin(), width.end(), 0);
    dist[s] = 0;
    width[s] = INF;

    while (s != -1) {
      int best = -1;
      found[s] = true;
      for (int k = 0; k < N; k++) {
        if (found[k]) continue;
        Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);
        Relax(s, k, flow[k][s], -cost[k][s], -1);
        if (best == -1 || dist[k] < dist[best]) best = k;
      }
      s = best;
    }

    for (int k = 0; k < N; k++)
      pi[k] = min(pi[k] + dist[k], INF);
    return width[t];
  }

  pair<L, L> GetMaxFlow(int s, int t) {
    L totflow = 0, totcost = 0;
    while (L amt = Dijkstra(s, t)) {
      totflow += amt;
      for (int x = t; x != s; x = dad[x].first) {
        if (dad[x].second == 1) {
          flow[dad[x].first][x] += amt;
          totcost += amt * cost[dad[x].first][x];
        } else {
          flow[x][dad[x].first] -= amt;
          totcost -= amt * cost[x][dad[x].first];
        }
      }
    }
    return make_pair(totflow, totcost);
  }
};

int main() { //Data Flow
  int N, M;

  while (scanf("%d%d", &N, &M) == 2) {
    VVL v(M, VL(3));
    for (int i = 0; i < M; i++)
      scanf("%Ld%Ld%Ld", &v[i][0], &v[i][1], &v[i][2]);
    L D, K;
    scanf("%Ld%Ld", &D, &K);

    MinCostMaxFlow mcmf(N+1);
    for (int i = 0; i < M; i++) {
      mcmf.AddEdge(int(v[i][0]), int(v[i][1]), K, v[i][2]);
      mcmf.AddEdge(int(v[i][1]), int(v[i][0]), K, v[i][2]);
    }
    mcmf.AddEdge(0, 1, D, 0);

    pair<L, L> res = mcmf.GetMaxFlow(0, N);

    if (res.first == D) {
      printf("%Ld\n", res.second);
    } else {
      printf("Impossible.\n");
    }
  }
  return 0;
}
```

MinCut.cpp

Description: Divide network flow into two components removing set of edges with the lowest combined cost. (min cut value, nodes in half of min cut)
Time: $\mathcal{O}(V^3)$ 54 lines

```cpp
typedef vector<int> VI;
typedef vector<VI> VVI;

const int INF = 1000000000;

pair<int, VI> GetMinCut(VVI &weights) {
  int N = weights.size();
  VI used(N), cut, best_cut;
  int best_weight = -1;

  for (int phase = N-1; phase >= 0; phase--) {
    VI w = weights[0];
    VI added = used;
    int prev, last = 0;
    for (int i = 0; i < phase; i++) {
      prev = last;
      last = -1;
      for (int j = 1; j < N; j++)
  if (!added[j] && (last == -1 || w[j] > w[last])) last = j;
      if (i == phase-1) {
  for (int j = 0; j < N; j++) weights[prev][j] += weights[last
    ][j];
  for (int j = 0; j < N; j++) weights[j][prev] = weights[prev][
    j];
  used[last] = true;
  cut.push_back(last);
  if (best_weight == -1 || w[last] < best_weight) {
    best_cut = cut;
    best_weight = w[last];
  }
      } else {
  for (int j = 0; j < N; j++)
    w[j] += weights[last][j];
  added[last] = true;
      }
    }
  }
  return make_pair(best_weight, best_cut);
}

int main() {
  int N;
  cin >> N;
  for (int i = 0; i < N; i++) {
    int n, m;
    cin >> n >> m;
    VVI weights(n, VI(n));
    for (int j = 0; j < m; j++) {
      int a, b, c;
      cin >> a >> b >> c;
      weights[a-1][b-1] = weights[b-1][a-1] = c;
    }
    pair<int, VI> res = GetMinCut(weights);
    cout << "Case #" << i+1 << ": " << res.first << endl;
  }
}
```

MaxBipartiteMaching.cpp

Description: Max Bipartite Maching in a Network Flow.
Time: $\mathcal{O}(VE^2)$ 52 lines

```cpp
typedef vector<int> VI;
typedef vector<VI> VVI;

bool FindMatch(int i, const VVI &w, VI &mr, VI &mc, VI &seen) {
  for (unsigned int j = 0; j < w[i].size(); j++) {
    if (w[i][j] && !seen[j]) {
      seen[j] = true;
      if (mc[j] < 0 || FindMatch(mc[j], w, mr, mc, seen)) {
        mr[i] = j;
        mc[j] = i;
        return true;
      }
    }
  }
  return false;
}

int BipartiteMatching(const VVI &w, VI &mr, VI &mc) {
  mr = VI(w.size(), -1);
  mc = VI(w[0].size(), -1);

  int ct = 0;
  for (unsigned int i = 0; i < w.size(); i++) {
    VI seen(w[0].size());
    if (FindMatch(i, w, mr, mc, seen)) ct++;
  }
  return ct;
}
int main() {
    int a,b,cases; scanf("%d",&cases);
    while(cases--)
    {
        int s,c,N,M; scanf("%d%d%d%d",&N,&M,&s,&c);
        VVI adj(M, VI(N));
        vector<pair<int,int>> P;
        for (int i = 0; i < N; i++) {
            scanf("%d%d",&a,&b);
            P.push_back({a,b});
        }
        for (int i = 0; i < M; i++) {
            scanf("%d%d",&a,&b);
            for(int j=0; j<N;j++)
            {
                pair<int,int> person = P[j];
                if(((abs(person.first-a)+abs(person.second-b))
                     <=s*c/200)) adj[i][j]=1;
            }
        }
        vector<int> V; vector<int> V_;
        printf("%d\n",BipartiteMatching(adj, V,V_));
    }
    return 0;
}
```

Dijkstra.cpp

Description: Finds shortest path to every points from an origin.
Time: $\mathcal{O}(E + V log(V))$ 61 lines

```
const int MAXN = 100100;
const int INF = 0x3f3f3f3f;

struct edge{
  int from, to, weight;
  edge(){}
  edge(int a, int b, int c){
    from = a;
    to = b;
    weight = c;
  }
};

struct state{
  int node, dist;
  state(){}
  state(int a, int b){
    node = a; dist = b;
  }
  bool operator<(const state &other)const{
    return other.dist < dist;
  }
};

vector<edge> graph[MAXN];
int dist[MAXN];

int a=1, b=3;
int N,E;

int dijkstra(int start, int end){
  dist[start] = 0;
  priority_queue<state> pq;
  pq.push(state(start, 0));
  while(!pq.empty()){
    state cur = pq.top(); pq.pop();
    if(dist[cur.node] < cur.dist) continue;
    if(cur.node == end) return cur.dist;
    for(int i=0;i<graph[cur.node].size();i++){
      int dest = graph[cur.node][i].to;
      int wht = graph[cur.node][i].weight + cur.dist;
      if(dist[dest] <= wht) continue;
      dist[dest] = wht;
      pq.push(state(dest, wht));
    }
  }
  return -1;
}
```

```
int main(){
  scanf("%d %d",&N,&E);
  memset(dist,0x3f,sizeof(dist));
  for(int i=1;i<=N;i++) graph[i].clear(); // clean graph
    for(int i=0;i<E;i++){
    int from, to, weight; scanf("%d %d %d",&from, &to, &weight)
        ;
    graph[from].push_back(edge(from,to,weight));
    graph[to].push_back(edge(to, from, weight)); // delete if
         undirected graph
    }
  printf("Dijkstra %d  %d %d\n",a,b,dijkstra(a,b));
  return 0;
}
```

Kruskal.cpp

Description: Finds Minimum Spanning Tree
Time: $\mathcal{O}(E * log(E))$ 64 lines

```
const int MAXN = 100100;
const int INF = 0x3f3f3f3f;

struct edge{
  int from, to, weight;
  edge(){}
  edge(int a, int b, int c){
    from = a;
    to = b;
    weight = c;
  }
  bool operator<(const edge &other)const{
    return weight < other.weight;
  }
};

struct UF{
    int parents[MAXN];
    int sz[MAXN];
    int components;
    int mst_sum;
    UF(int n){
        for(int i=0;i<n;i++){
            parents[i] = i; sz[i] = 1;
        }
        components = n;
        mst_sum = 0;
    }
    int find(int n){
        return n == parents[n] ? n : find(parents[n]);
    }
    bool isConnected(int a, int b){
        return find(a) == find(b);
    }
    void connect(int a, int b, int weight){
        if(isConnected(a, b)) return;
        int A,B; A = find(a); B = find(b);
        if(sz[A] > sz[B]){
            parents[B] = A;
            sz[A] += sz[B];
        }
        else{
            parents[A] = B;
            sz[B] += sz[A];
        }
        mst_sum += weight;
        components--;
    }
};
```

```
int a=1,N,E;
int main(){
  scanf("%d %d",&N,&E);
  vector<edge> edges;
  UF uf = UF(N);
  for(int i=0;i<E;i++){
    int from, to, weight; scanf("%d %d %d",&from, &to, &weight)
        ;
    edges.push_back(edge(from,to,weight));
  }
  sort(edges.begin(), edges.end());
  for(int i=0;i<E;i++) uf.connect(edges[i].from, edges[i].to,
       edges[i].weight);
  printf("Kruskal %d %d\n",a,uf.mst_sum);
  return 0;
}
```

LCA.cpp

Description: Gives the Lowest Common Ancestor.
Time: $\mathcal{O}(N)$ 70 lines

```
const int max_nodes, log_max_nodes;

int num_nodes, log_num_nodes, root;
vector<int> children[max_nodes];
// children[i] contains the children of node i int A[max_nodes
     ][log_max_nodes+1];
// A[i][j] is the 2^j-th ancestor of node i, or -1 if that
     ancestor does not exist int L[max_nodes];
// L[i] is the distance between node i and the root
// floor of the binary logarithm of n
int lb(unsigned int n)
{
  if(n==0)  return -1;
  int p = 0;
  if (n >= 1<<16) { n >>= 16; p += 16; }
  if (n >= 1<< 8) { n >>=  8; p +=  8; }
  if (n >= 1<< 4) { n >>=  4; p +=  4; }
  if (n >= 1<< 2) { n >>=  2; p +=  2; }
  if (n >= 1<< 1) {           p +=  1; }
  return p;
}

void DFS(int i, int l)
{
  L[i] = l;
  for(int j = 0; j < children[i].size(); j++)
    DFS(children[i][j], l+1);
}

int LCA(int p, int q) {
  // ensure node p is at least as deep as node q
  if(L[p] < L[q])  swap(p, q);

  // "binary search" for the ancestor of node p situated on the
       same level as q
  for(int i = log_num_nodes; i >= 0; i--)
    if(L[p] - (1<<i) >= L[q])
  p = A[p][i];
  if(p == q)   return p;

   // "binary search" for the LCA
  for(int i = log_num_nodes; i >= 0; i--)
  {
    if(A[p][i] != -1 && A[p][i] != A[q][i])
    {
      p = A[p][i];
      q = A[q][i];
    }
    }
    return A[p][0];
}

int main(int argc,char* argv[])
{
  // read num_nodes, the total number of nodes
  log_num_nodes=lb(num_nodes);
  for(int i = 0; i < num_nodes; i++)
  {
    int p;
    // read p, the parent of node i or -1 if node i is the root
        A[i][0] = p;
        if(p != -1)  children[p].push_back(i);
        else  root = i;
    }
    // precompute A using dynamic programming
    for(int j = 1; j <= log_num_nodes; j++)
      for(int i = 0; i < num_nodes; i++)
        if(A[i][j-1] != -1)  A[i][j] = A[A[i][j-1]][j-1];
        else  A[i][j] = -1;
    // precompute L
    DFS(root, 0);
    return 0;
  }
```

Topologicalsort.cpp

Description: Linear ordering of its vertices such that for every directed edge u -> v from vertex u to vertex v, u comes before v in the ordering
Time: $\mathcal{O}(E+V)$ 35 lines

```
const int MAXN = 100100;
const int INF = 0x3f3f3f3f;

vector<int> graph[MAXN];
bool visited[MAXN];
stack<int> topological_order;

int N,E;

void DFS(int node){
  if(visited[node]) return;
  visited[node] = true;
  for(int i=0;i<graph[node].size();i++){
    int dest = graph[node][i];
    DFS(dest);
  }
  topological_order.push(node);
}

int main(){
  scanf("%d %d",&N,&E);
  for(int i=1;i<=N;i++) graph[i].clear(); // clean graph
  for(int i=0;i<E;i++){
    int from, to; scanf("%d %d",&from, &to);
    graph[from].push_back(to);
  }
  //Supposing node 1 is not dependent has no ancestor
  DFS(1);
  while(!topological_order.empty()){
    printf(" %d",topological_order.top());
    topological_order.pop();
  }
  printf("\n");
  return 0;
}
```

BFS-DFS.cpp

Description: BFS-DFS
Time: $\mathcal{O}(E+V)$ 55 lines

```
const int MAXN = 100100;
const int INF = 0x3f3f3f3f;

vector<int> graph[MAXN];
bool visited[MAXN];

int a=1, b=6;
int N,E;

int DFS(int node, int target){
  if(node == target) return 0;
  if(visited[node]) return INF;
  visited[node] = true;
  int best_result = INF;
  for(int i=0;i<graph[node].size();i++){
    int dest = graph[node][i];
    best_result = min(
        best_result,
        DFS(dest, target)+1
    );
  }
  return best_result;
}

int BFS(int start, int target){
  queue<pair<int,int> > q;
  q.push(make_pair(start,0));
  visited[start] = true;
  while(!q.empty()){
    pair<int,int> current = q.front(); q.pop();
    if(current.first == target) return current.second;
    for(int i=0;i<graph[current.first].size();i++){
      int dest = graph[current.first][i];
      if(visited[dest]) continue;
      visited[dest] = true;
      q.push(make_pair(dest,current.second+1));
    }
  }
  return -1;
}

int main(){
  scanf("%d %d",&N,&E);
  for(int i=1;i<=N;i++) graph[i].clear(); // clean graph
  for(int i=0;i<E;i++){
    int from, to; scanf("%d %d",&from, &to);
    graph[from].push_back(to);
    graph[to].push_back(from); // delete if undirected
  }
  memset(visited,0,sizeof(visited));
  printf("BFS  %d  %d  %d\n",a,b,BFS(a,b));
  memset(visited,0,sizeof(visited));
  printf("DFS  %d  %d  %d\n",a,b,DFS(a,b));
  return 0;
}
```

FloydWarshall.cpp

Description: Shortest path in a weighted graph beetwen any pair of nodes.
Time: $\mathcal{O}(N^3)$ 16 lines

```
//It could be not symmetric!
   int dist[4][4] = { {  0, 5  ,INF,10},
                      {INF, 0  ,3  ,INF},
                      {INF, INF,0  ,1},
                      {INF, INF,INF,0}};
void floydWarshall ()
{
    int dist[4][4], i, j, k;
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            dist[i][j] = graph[i][j];
    for (k = 0; k < 4; k++)
        for (i = 0; i < 4; i++)
            for (j = 0; j < 4; j++)
                 dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j
                      ]);
}
```

ConnectedComponents.cpp

Description: DFS that solves Connected Components
Time: $\mathcal{O}(E+V)$ 36 lines

```
const int MAXN = 100100;
const int INF = 0x3f3f3f3f;

vector<int> graph[MAXN];
bool visited[MAXN];

int N,E;

void DFS(int node){
  if(visited[node]) return;
  visited[node] = true;
  for(int i=0;i<graph[node].size();i++){
    int dest = graph[node][i];
    DFS(dest);
  }
}

int main(){
  scanf("%d %d",&N,&E);
  for(int i=1;i<=N;i++) graph[i].clear(); // clean graph
  for(int i=0;i<E;i++){
    int from, to; scanf("%d %d",&from, &to);
    graph[from].push_back(to);
    graph[to].push_back(from); // undirected or not
  }
  memset(visited,0,sizeof(visited));
  int comps = 0;
  for(int i=1;i<=N;i++){
    if(!visited[i]){
      DFS(i);
      comps++;
    }
  }
  printf("%d components\n", comps);
  return 0;
}
```

Hungarian.cpp

Description: Kuhn-Munkres Algorithm. Solves the assignament problem maximizing the value. In this case it shows how can be used to minimize (Put negative cost).
Time: $\mathcal{O}(N^3)$ 110 lines

```
const int MAXV = 2005;
struct KM {
  int _mem[MAXV*MAXV];
  int *w[MAXV];
  int lx[MAXV], ly[MAXV];
  int16_t mx[MAXV], my[MAXV];
  int16_t aug[MAXV], vis[MAXV];
  pair<int, int> slack[MAXV];
  int nx, ny;
  int match() {
    for (int i = 0; i < nx; i++)
      lx[i] = *max_element(w[i], w[i]+ny);
    fill(ly, ly+ny, 0);
    fill(mx, mx+nx, -1);
    fill(my, my+ny, -1);
    fill(slack, slack+ny, make_pair(0, 0));
    for (int root = 0; root < nx; root++) {
      fill(aug, aug+ny, -1);
      fill(vis, vis+nx, 0);
      vis[root] = 1;
      for (int y = 0; y < ny; y++)
        slack[y] = make_pair(lx[root]+ly[y]-w[root][y], root);
      int sy = -1;
      for (;;) {
        int delta = INT_MAX, sx = -1;
        for (int y = 0; y < ny; y++) {
          if (aug[y] == -1 && slack[y].first < delta) {
            delta = slack[y].first;
            sx = slack[y].second, sy = y;
          }
        }
//        assert(vis[sx]);
        if (delta > 0) {
          for (int x = 0; x < nx; x++) {
            if (vis[x])
              lx[x] -= delta;
          }
          for (int y = 0; y < ny; y++) {
            if (aug[y] > -1)
              ly[y] += delta;
            else
              slack[y].first -= delta;
          }
        }
//        assert(lx[sx] + ly[sy] == w[sx][sy]);
        aug[sy] = sx;
        sx = my[sy];
        if (sx == -1)
          break;
        vis[sx] = 1;
        for (int y = 0; y < ny; y++) {
          if (aug[y] == -1) {
            if (lx[sx]+ly[y]-w[sx][y] < slack[y].first)
              slack[y] = make_pair(lx[sx]+ly[y]-w[sx][y], sx);
          }
        }
      }

      while (sy != -1) {
        int sx = aug[sy];
        int ty = mx[sx];
        my[sy] = sx;
        mx[sx] = sy;
        sy = ty;
      }
    }

    int ret = 0;
    for (int i = 0; i < nx; i++)
      ret += lx[i];
    for (int i = 0; i < ny; i++)
      ret += ly[i];
    return ret;
  }
  void init(int nx, int ny) {
    this->nx = nx, this->ny = ny;
    for (int i = 0; i < nx; i++)
      w[i] = _mem + i*ny;
  }
} km;

int main() {
  int n, m;
  const int MAXN = 2005;
  int bx[MAXN], by[MAXN];
  int cx[MAXN], cy[MAXN];
  int sx, sy;
  while (scanf("%d %d", &n, &m) == 2) {
    for (int i = 0; i < n; i++)
      scanf("%d %d", &bx[i], &by[i]);
    for (int i = 0; i < m; i++)
      scanf("%d %d", &cx[i], &cy[i]);
    scanf("%d %d", &sx, &sy);
    km.init(n, m+(n-1));
    int ret = 0;
    for (int i = 0; i < n; i++) {
      int b = abs(bx[i]-sx)+abs(by[i]-sy);
      ret += b;
      for (int j = 0; j < m; j++) {
        int d = abs(bx[i]-cx[j])+abs(by[i]-cy[j]);
        km.w[i][j] = -d;
      }
      for (int j = 0; j < n-1; j++)
        km.w[i][j+m] = -b;
    }
    ret += -km.match();
    printf("%d\n", ret);
  }
  return 0;
}
```

3.1 Planar Graphs

Formula Euler: v-e+f = 2
Theorem 1: If $v >= 3$ then $e <= 3v - 6$.
Theorem 2: If $v > 3$ and there are no cycles of length 3, then $e <= 2v - 4$.

3.2 Bipartite Graphs

let G = (V,E) be a Bipartite graph, we will call each part L and R respectively.
Maximum Matching is equal to **Minimum Vertex Cover**, to get the nodes in the MVC we choose the unmatched vertices of L and run a BFS/DFS in the graph with edges in the matching directed from R to L, and edges not in the matching directed from L to R. Let's all this Z, now MVC = $(L\backslash Z) \cup (R \cap Z)$.
The **Minimum Edge Cover** is obtained by doing a Maximum Matching, Then run a BFS/DFS from unmatched vertices of L, the MEC is the unreachable vertices of A and reachable vertices of B.
The **Maximum Independent Set** is the complementary of the **Minimum Vertex Cover**

3.3 DAG

The **Minimum Path Cover** is given by the edges in the MM of the bipartite graph after doubling the vertices.

Strings (4)

AhoCorasick.cpp
Description: Builds an Ahocorasick Trie, with suffix links.
Usage: This is an offline Algorithm, Pass the vector of patterns to Trie.init(v), find function returns the number of times each string appears
Time: $\mathcal{O}(n + m + z)$ 72 lines

```
const int MaxM = 200005;

struct Trie{
    static const int Alpha = 26;
    static const int first = 'a';
    int lst = 1;
    struct node{
        int nxt[Alpha] = {}, p = -1;
        char c;
        vector<int> end; //if all patterns are different, can
            use flag instead
        int SuffixLink = -1;
        int cnt = 0;
  };
  vector<node> V;
  int num;
  stack<int> reversebfs;
  inline int getval(char c) {
    return c - first;
  }
  void CreateSuffixLink() {
    queue<int> q;
    for(q.push(0); q.size(); q.pop()) {
      int pos = q.front();
      reversebfs.push(pos);
      if(!pos || !V[pos].p) V[pos].SuffixLink = 0;
      else {
        int val = getval(V[pos].c);
        int j = V[V[pos].p].SuffixLink;
        V[pos].SuffixLink = V[j].nxt[val];
      }
      for(int i = 0; i < Alpha; ++i) {
        if(V[pos].nxt[i]) q.push(V[pos].nxt[i]);
        else if(!pos || !V[pos].p)  V[pos].nxt[i] = V[0].nxt[i
            ];
        else V[pos].nxt[i] = V[V[pos].SuffixLink].nxt[i];
      }
    }
  }

  void init(vector<string> &v) {
    V.resize(MaxM);
    num = v.size();
    int id = 0;
    for(auto &s : v) {
      int pos = 0;
      for(char &c : s) {
        int val = getval(c);
        if(!V[pos].nxt[val]) {
          V[lst].p = pos; V[lst].c = c; V[pos].nxt[val] = lst
              ++;
        }
        pos = V[pos].nxt[val];
      }
      V[pos].end.emplace_back(id++);
    }
    CreateSuffixLink();
  }

  vector<int> find(string& word) {
    int pos = 0;
    vector<int> ans(num, 0);
    for(auto &c : word) {
      int val = getval(c);
      pos = V[pos].nxt[val];
      V[pos].cnt++; //We count the times we reach each node,
           and then do a reverse propagation
    }
    for(;reversebfs.size();reversebfs.pop()) {
      int x = reversebfs.top(); //When we process x, we know we
            have been there V[x].cnt times;
      for(int i : V[x].end) ans[i] += V[x].cnt;
      if(V[x].SuffixLink != -1) V[V[x].SuffixLink].cnt += V[x].
           cnt;
    }
    return ans;
  }
};
```

Manacher.cpp

Description: call with String str of length n, returns: r[0..2*n-2],r[i] radius of longest palindrome with center i/2 in str
Time: $\mathcal{O}(n)$

11 lines

```
void manacher(int n, char *str, int *r) {
  r[0] = 0;
  int p = 0;
  FOR(i, 1, 2*n-1) {
    r[i] = (p/2 + r[p] >= (i+1)/2) ? min(r[2*p - i], p/2 + r[p]
          - i/2) : 0;
    while (i/2 + r[i] + 1 < n && (i+1)/2 - r[i] - 1 >= 0
    && str[i/2 + r[i] + 1] == str[(i+1)/2 - r[i] - 1]) r[i]++;
    if (i/2 + r[i] > p/2 + r[p]) p = i;
  }
  // FOR(i,0,2*n-1) r[i] = r[i]*2 + !(i&1); // change radius to
        diameter
}
```

SuffixArray.cpp

Description: string matching
Time: $\mathcal{O}(P+T)$ where P is the length of the pattern, T is length of the text

146 lines

```
typedef pair<int, int> ii;

#define MAX_N 100010 // O(n log n)
char T[MAX_N]; // up to 100K characters
int n; //length of string
int RA[MAX_N], tempRA[MAX_N];
int SA[MAX_N], tempSA[MAX_N];
int c[MAX_N];
char P[MAX_N]; // the pattern string (for string matching)
int m; // the length of pattern string
int Phi[MAX_N];// for computing LCP
int PLCP[MAX_N];
int LCP[MAX_N];
bool cmp(int a, int b) { return strcmp(T + a, T + b) < 0; }

void countingSort(int k) {  // O(n)
  int i, sum, maxi = max(300, n); // up to 255 chars
  memset(c, 0, sizeof c);
  for (i = 0; i < n; i++)
    c[i + k < n ? RA[i + k] : 0]++;
  for (i = sum = 0; i < maxi; i++) {
    int t = c[i]; c[i] = sum; sum += t;
  }
  for (i = 0; i < n; i++) tempSA[c[SA[i]+k < n ? RA[SA[i]+k] :
       0]++] = SA[i];
  for (i = 0; i < n; i++) SA[i] = tempSA[i];
}
void constructSA() { // Up to 100000 characters
  int i, k, r;
  for (i = 0; i < n; i++) RA[i] = T[i];
  for (i = 0; i < n; i++) SA[i] = i;
  for (k = 1; k < n; k <<= 1) {
    countingSort(k);
    countingSort(0);
    tempRA[SA[0]] = r = 0;
    for (i = 1; i < n; i++)
      tempRA[SA[i]] =
      (RA[SA[i]] == RA[SA[i-1]] && RA[SA[i]+k] == RA[SA[i-1]+k
           ]) ? r : ++r;
    for (i = 0; i < n; i++)
      RA[i] = tempRA[i];
    if (RA[SA[n-1]] == n-1) break;
} }

void computeLCP() {
  int i, L;
  Phi[SA[0]] = -1;
  for (i = 1; i < n; i++)
    Phi[SA[i]] = SA[i-1];
  for (i = L = 0; i < n; i++) {// LCP in O(n)
    if (Phi[i] == -1) { PLCP[i] = 0; continue; }
    while (T[i + L] == T[Phi[i] + L]) L++;
    PLCP[i] = L;
    L = max(L-1, 0);
  }
  for (i = 0; i < n; i++)
    LCP[i] = PLCP[SA[i]];
}

ii stringMatching() { // O(m log n)
  int lo = 0, hi = n-1, mid = lo;// valid matching = [0..n-1]
  while (lo < hi) {
    mid = (lo + hi) / 2;
    int res = strncmp(T + SA[mid], P, m);
    if (res >= 0) hi = mid;
    else          lo = mid + 1;
  }
  if (strncmp(T + SA[lo], P, m) != 0) return ii(-1, -1); // if
       not found
  ii ans; ans.first = lo;
  lo = 0; hi = n - 1; mid = lo;
  while (lo < hi) {
    mid = (lo + hi) / 2;
    int res = strncmp(T + SA[mid], P, m);
    if (res > 0) hi = mid;
    else          lo = mid + 1;
  }
  if (strncmp(T + SA[hi], P, m) != 0) hi--;
  ans.second = hi;
  return ans;
}
ii LRS() {
  int i, idx = 0, maxLCP = -1;
  for (i = 1; i < n; i++) // O(n)
    if (LCP[i] > maxLCP)
      maxLCP = LCP[i], idx = i;
  return ii(maxLCP, idx);
}
int owner(int idx) { return (idx < n-m-1) ? 1 : 2; }

ii LCS() {
  int i, idx = 0, maxLCP = -1;
  for (i = 1; i < n; i++) // O(n)
    if (owner(SA[i]) != owner(SA[i-1]) && LCP[i] > maxLCP)
      maxLCP = LCP[i], idx = i;
  return ii(maxLCP, idx);
}
int main() {
  printf("Suffix Array:\n");
  strcpy(T, "GATAGACA");
  n = (int)strlen(T);
  T[n++] = '$';
  //if '\n' uncomment T[n-1] = '$'; T[n] = 0;
  constructSA();// O(n log n)
  printf("\nSR of string T = '%s':\n", T);
  printf(" i\tSA[i]\tSuffix\n");
  for (int i = 0; i < n; i++) printf("%2d\t%2d\t%s\n", i, SA[i
       ], T + SA[i]);
  computeLCP();// O(n)
  ii ans = LRS();
  char lrsans[MAX_N];
  strncpy(lrsans, T + SA[ans.second], ans.first);
  printf("\nLongest Repeated Substring O(n)\n");
  printf("\nLRS is '%s' = %d\n\n", lrsans, ans.first);
  printf("\nString matching O(m log n)\n");
  //printf("\nenter string P below,find P in T:\n");
  strcpy(P, "A");
  m = (int)strlen(P);
  //if '\n' uncomment P[m-1] = 0; m--;
  ii pos = stringMatching();
  if (pos.first != -1 && pos.second != -1) {
    printf("%s is found SA[%d..%d] of %s\n", P, pos.first, pos.
         second, T);
    printf("They are:\n");
    for (int i = pos.first; i <= pos.second; i++)
      printf("  %s\n", T + SA[i]);
  } else printf("%s is not found in %s\n", P, T);
  printf("\nLongest Common Substring O(n)\n");
  printf("\nRemember, T = '%s'\nNow, enter another string P:\n"
       , T);
  // T already has '$' at the back
  strcpy(P, "CATA");
  m = (int)strlen(P);
  // if '\n' is read, uncomment the next line
  //P[m-1] = 0; m--;
  strcat(T, P);
  strcat(T, "#");
  n = (int)strlen(T);
  //Un prefijo de un sufijo es un substring
  constructSA();
  computeLCP();
  printf("\nLongest Common Prefix O(n)\n");
  printf("\nThe LCP information of 'T+P' = '%s':\n", T);
  printf(" i\tSA[i]\tLCP[i]\tOwner\tSuffix\n");
  for (int i = 0; i < n; i++)
    printf("%2d\t%2d\t%2d\t%2d\t%s\n", i, SA[i], LCP[i], owner(
         SA[i]), T + SA[i]);
  ans = LCS();
  char lcsans[MAX_N];
  strncpy(lcsans, T + SA[ans.second], ans.first);
  printf("\nThe LCS is '%s' = %d\n", lcsans, ans.first);
  return 0;
}
```

Z.cpp

Description: Computes the longest prefix which ends at the i-th position

15 lines

```
vector<int> Z(string &s) {
    int n = s.size();
    int L, R;
    L = R = 0;
    vector<int> Z(n, 0);
    for (int i = 1; i < n; ++i){
        if (i < R) Z[i] = min(Z[i-L], R-i);
        else Z[i] = 0;
        while (Z[i] + i < n and s[Z[i]] == s[i+Z[i]]) ++Z[i];
        if (i+Z[i] > R){
            L = i;
            R = i + Z[i];
        }
    }
}
```

KMP.cpp

Description: string matching

Time: $\mathcal{O}(P+T)$ where P is the length of the pattern, T is length of the text

14 lines

```
string s, p; cin >> s >> p;
vector<int> pi(p.size() + 1, 0);
int k = 0;
for (int i = 2; i <= p.size(); ++i) {
  while (k > 0 and p[i - 1] != p[k]) k = pi[k];
  if (p[i - 1] == p[k]) ++k;
  pi[i] = k;
}
k = 0;
for (int i = 0; i < s.size(); ++i) {
  while (k > 0 and s[i] != p[k]) k = pi[k];
  if (p[k] == s[i]) ++k;
  if (k == p.size()) k = pi[k]; //Matching
}
```

MinRotation.cpp

Description: Finds the lexicographically smallest rotation of a string.

Usage: `rotate(v.begin(), v.begin()+min_rotation(v), v.end());`

Time: $\mathcal{O}(N)$

8 lines

```
int min_rotation(string s) {
  int a=0, N=sz(s); s += s;
  FOR(b,0,N) FOR(i,0,N) {
    if (a+i == b || s[a+i] < s[b+i]) {b += max(0, i-1); break;}
    if (s[a+i] > s[b+i]) { a = b; break; }
  }
  return a;
}
```

Mathematics (5)

5.1 Equations

$$ax^2+bx+c=0 \Rightarrow x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

The extremum is given by $x=-b/2a$.

$$\begin{matrix} ax+by=e \\ cx+dy=f \end{matrix} \Rightarrow \begin{matrix} x=\dfrac{ed-bf}{ad-bc} \\ y=\dfrac{af-ec}{ad-bc} \end{matrix}$$

In general, given an equation $Ax=b$, the solution to a variable x_i is given by

$$x_i=\frac{\det A_i'}{\det A}$$

where A_i' is A with the i'th column replaced by b.

5.2 Geometry

5.2.1 Quadrilaterals

With side lengths a,b,c,d, diagonals e,f, diagonals angle θ, area A and magic flux $F=b^2+d^2-a^2-c^2$:

$$4A=2ef\cdot\sin\theta=F\tan\theta=\sqrt{4e^2f^2-F^2}$$

For cyclic quadrilaterals the sum of opposite angles is $180°$, $ef=ac+bd$, and $A=\sqrt{(p-a)(p-b)(p-c)(p-d)}$.

5.2.2 Pick's theorem

$A=i+b/2-1$ Boundary point(b): a lattice point on the polygon (including vertices) Interior Point(i): a lattice point in the polygon's interior region

5.2.3 Volumes

	Sphere	Cube	Tetrahedron
Area	$4\pi r^2$	$6a^2$	$a^2\sqrt{3}$
Volume	$\frac{4}{3}\pi r^3$	a^3	$\frac{1}{12}a^3\sqrt{2}$

	Octahedron	Dodecahedron	Icosahedron
Area	$2\sqrt{3}a^2$	$3a^2\sqrt{25+10\sqrt{5}}$	$5\sqrt{3}a^2$
Volume	$\frac{1}{3}\sqrt{2}a^3$	$\frac{1}{4}(15+7\sqrt{5})a^3$	$\frac{5}{12}(3+\sqrt{5})a^3$

5.3 Sums

$$c^a+c^{a+1}+\cdots+c^b=\frac{c^{b+1}-c^a}{c-1}, c\neq 1$$

$$1+2+3+\cdots+n=\frac{n(n+1)}{2}$$

$$1^2+2^2+3^2+\cdots+n^2=\frac{n(2n+1)(n+1)}{6}$$

$$1^3+2^3+3^3+\cdots+n^3=\frac{n^2(n+1)^2}{4}$$

$$1^4+2^4+3^4+\cdots+n^4=\frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

5.3.1 Triangles

Side lengths: a,b,c

Semiperimeter: $p=\dfrac{a+b+c}{2}$

Area: $A=\sqrt{p(p-a)(p-b)(p-c)}$

Circumradius: $R=\dfrac{abc}{4A}$

Inradius: $r=\dfrac{A}{p}$

Length of median (divides triangle into two equal-area triangles): $m_a=\frac{1}{2}\sqrt{2b^2+2c^2-a^2}$

Length of bisector (divides angles in two):

$$s_a=\sqrt{bc\left[1-\left(\frac{a}{b+c}\right)^2\right]}$$

Law of sines: $\dfrac{\sin\alpha}{a}=\dfrac{\sin\beta}{b}=\dfrac{\sin\gamma}{c}=\dfrac{1}{2R}$

Law of cosines: $a^2=b^2+c^2-2bc\cos\alpha$

Law of tangents: $\dfrac{a+b}{a-b}=\dfrac{\tan\dfrac{\alpha+\beta}{2}}{\tan\dfrac{\alpha-\beta}{2}}$

5.4 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x. It will then have an expected value (mean) $\mu=\mathbb{E}(X)=\sum_x xp_X(x)$ and variance $\sigma^2=V(X)=\mathbb{E}(X^2)-(\mathbb{E}(X))^2=\sum_x(x-\mathbb{E}(X))^2p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$\mathbb{E}(aX+bY)=a\mathbb{E}(X)+b\mathbb{E}(Y)$$

For independent X and Y,

$$V(aX+bY)=a^2V(X)+b^2V(Y).$$

5.4.1 Discrete distributions

Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\mathrm{Bin}(n,p)$, $n=1,2,\ldots$, $0\leq p\leq 1$.

$$p(k)=\binom{n}{k}p^k(1-p)^{n-k}$$

$$\mu = np, \sigma^2 = np(1-p)$$

Bin(n,p) is approximately Po(np) for small p.

BinomialModPrime.cpp

Description: Lucas' thm: Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod p$. fact and invfact must hold pre-computed factorials / inverse factorials, e.g. from ModInverse.h.
Time: $\mathcal{O}(\log_p n)$

10 lines

```
ll chooseModP(ll n, ll m, int p, vi& fact, vi& invfact) {
  ll c = 1;
  while (n || m) {
    ll a = n % p, b = m % p;
    if (a < b) return 0;
    c = c * fact[a] % p * invfact[b] % p * invfact[a - b] % p;
    n /= p; m /= p;
  }
  return c;
}
```

GCD/LCM

GCDLCM.cpp

Description: Lucas' thm: Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod p$. fact and invfact must hold pre-computed factorials / inverse factorials, e.g. from ModInverse.h.
Time: $\mathcal{O}(\log_p n)$

5 lines

```
int gcd(int a, int b) {
while (b > 0) {
int temp = b; b = a % b; a = temp; }
return a; }
int lcm(int a, int b){ return a*(b/gcd(a,b));}
```

Geometry (6)

6.1 Geometric primitives

Point.h

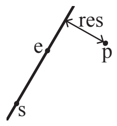
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.)

25 lines

```
template <class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T a=0, T b=0) : x(a), y(b) {}
  bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
  bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return (a-*this).cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes dist()=1
  P perp() const { return P(-y, x); } // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
};
```

lineDistance.h

Description:
Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance.



"Point.h" 4 lines

```
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
  return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

pointsToLine.h

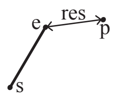
Description: Convert two points to Line

9 lines

```
// the answer is stored in the third parameter (pass by
     reference)
void pointsToLine(point p1, point p2, line &l) {
  if (fabs(p1.x - p2.x) < EPS) {  // vertical line is fine
    l.a = 1.0;    l.b = 0.0;    l.c = -p1.x;// default values
  } else {
    l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
    l.b = 1.0; // IMPORTANT: we fix the value of b to 1.0
    l.c = -(double)(l.a * p1.x) - p1.y;
} }
```

SegmentDistance.h

Description:
Returns the shortest distance between point p and the line segment from point s to e.



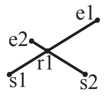
Usage: `Point<double> a, b(2,2), p(1,1);`
`bool onSegment = segDist(a,b,p) < 1e-10;`

"Point.h" 6 lines

```
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
  if (s==e) return (p-s).dist();
  auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
  return ((p-s)*d-(e-s)*t).dist()/d;
}
```

SegmentIntersection.h

Description:
If a unique intersetion point between the line segments going from s1 to e1 and from s2 to e2 exists r1 is set to this point and 1 is returned. If no intersection point exists 0 is returned and if infinitely many exists 2 is returned and r1 and r2 are set to the two ends of the common line. The wrong position will be returned if P is Point<int> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long. Use segmentIntersectionQ to get just a true/false answer.



Usage:
```
Point<double> intersection, dummy;
if (segmentIntersection(s1,e1,s2,e2,intersection,dummy)==1)
cout << "segments intersect at " << intersection << endl;
```

"Point.h" 27 lines

```
template<class P>
int segmentIntersection(const P& s1, const P& e1,
    const P& s2, const P& e2, P& r1, P& r2) {
  if (e1==s1) {
    if (e2==s2) {
      if (e1==e2) { r1 = e1; return 1; } //all equal
      else return 0; //different point segments
    } else return segmentIntersection(s2,e2,s1,e1,r1,r2);//swap
  }
  //segment directions and separation
  P v1 = e1-s1, v2 = e2-s2, d = s2-s1;
  auto a = v1.cross(v2), a1 = v1.cross(d), a2 = v2.cross(d);
  if (a == 0) { //if parallel
    auto b1=s1.dot(v1), c1=e1.dot(v1),
         b2=s2.dot(v1), c2=e2.dot(v1);
    if (a1 || a2 || max(b1,min(b2,c2))>min(c1,max(b2,c2)))
      return 0;
    r1 = min(b2,c2)<b1 ? s1 : (b2<c2 ? s2 : e2);
    r2 = max(b2,c2)>c1 ? e1 : (b2>c2 ? s2 : e2);
    return 2-(r1==r2);
  }
  if (a < 0) { a = -a; a1 = -a1; a2 = -a2; }
  if (0<a1 || a<-a1 || 0<a2 || a<-a2)
    return 0;
  r1 = s1-v1*a2/a;
  return 1;
}
```

SegmentIntersectionQ.h

Description: Like segmentIntersection, but only returns true/false. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.

"Point.h" 16 lines

```
template <class P>
bool segmentIntersectionQ(P s1, P e1, P s2, P e2) {
  if (e1 == s1) {
    if (e2 == s2) return e1 == e2;
    swap(s1,s2); swap(e1,e2);
  }
  P v1 = e1-s1, v2 = e2-s2, d = s2-s1;
  auto a = v1.cross(v2), a1 = d.cross(v1), a2 = d.cross(v2);
  if (a == 0) { // parallel
    auto b1 = s1.dot(v1), c1 = e1.dot(v1),
         b2 = s2.dot(v1), c2 = e2.dot(v1);
    return !a1 && max(b1,min(b2,c2)) <= min(c1,max(b2,c2));
  }
  if (a < 0) { a = -a; a1 = -a1; a2 = -a2; }
  return (0 <= a1 && a1 <= a && 0 <= a2 && a2 <= a);
}
```

segmentIntersectionPoint.h

Description: Segment Intersection given the points

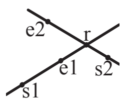
34 lines

```
double dist(point p1, point p2) {// Euclidean distance
  return hypot(p1.x - p2.x, p1.y - p2.y); }

struct vec { double x, y;
  vec(double _x, double _y) : x(_x), y(_y) {} };

vec toVec(point a, point b) {
  return vec(b.x - a.x, b.y - a.y); }

double dot(vec a, vec b) { return (a.x * b.x + a.y * b.y); }

double norm_sq(vec v) { return v.x * v.x + v.y * v.y; }

vec scale(vec v, double s) {  // nonnegative s = [<1 .. 1 ..
    >1]
  return vec(v.x * s, v.y * s); }// shorter.same.longer

point translate(point p, vec v) { // translate p according to v
  return point(p.x + v.x , p.y + v.y); }
```

```
double distToLine(point p, point a, point b, point &c) {
  // formula: c = a + u * ab
  vec ap = toVec(a, p), ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(ab);
  c = translate(a, scale(ab, u)); // translate a to c
  return dist(p, c); }

double distToLineSegment(point p, point a, point b, point &c) {
  vec ap = toVec(a, p), ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(ab);
  if (u < 0.0) { c = point(a.x, a.y);// closer to a
    return dist(p, a); }
  if (u > 1.0) { c = point(b.x, b.y); // closer to b
    return dist(p, b); }
  return distToLine(p, a, b, c); } // run distToLine as above
```

lineIntersection.h

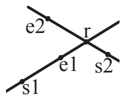
Description:
If a unique intersetion point of the lines going through s1,e1 and s2,e2 exists r is set to this point and 1 is returned. If no intersection point exists 0 is returned and if infinitely many exists -1 is returned. If s1==e1 or s2==e2 -1 is returned. The wrong position will be returned if P is Point<int> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.



Usage: point<double> intersection;
```
if (1 == LineIntersection(s1,e1,s2,e2,intersection))
cout << "intersection point at " << intersection << endl;
```
"Point.h" 9 lines
```
template <class P>
int lineIntersection(const P& s1, const P& e1, const P& s2,
    const P& e2, P& r) {
  if ((e1-s1).cross(e2-s2)) { //if not parallell
    r = s2-(e2-s2)*(e1-s1).cross(s2-s1)/(e1-s1).cross(e2-s2);
    return 1;
  } else
    return -((e1-s1).cross(s2-s1)==0 || s2==e2);
}
```

lineIntersectionv2.h

Description:
If a unique intersetion point of the lines going through s1,e1 and s2,e2 exists r is set to this point and 1 is returned. If no intersection point exists 0 is returned and if infinitely many exists -1 is returned. If s1==e1 or s2==e2 -1 is returned. The wrong position will be returned if P is Point<int> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.



Usage: point<double> intersection;
```
if (1 == LineIntersection(s1,e1,s2,e2,intersection))
cout << "intersection point at " << intersection << endl;
```
11 lines
```
struct line { double a, b, c; };

bool areIntersect(line l1, line l2, point &p) {
    int den = l1.a*l2.b - l1.b*l2.a;
    if(!den) return false; //same line or parallel
    p.x = l1.c*l2.b - l1.b*l2.c;
    if(p.x)  p.x /= den;
    p.y = l1.a*l2.c - l1.c*l2.a;
    if(p.y)  p.y /= den;
    return true;
}
```

sideOf.h

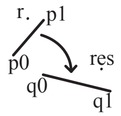
Description: Returns where p is as seen from s towards e. 1/0/-1 ⇔ left/on line/right. If the optional argument *eps* is given 0 is returned if p is within distance *eps* from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.
Usage: bool left = sideOf(p1,p2,q)==1;
"Point.h" 11 lines
```
template <class P>
int sideOf(const P& s, const P& e, const P& p) {
  auto a = (e-s).cross(p-s);
  return (a > 0) - (a < 0);
}
template <class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

onSegment.h

Description: Returns true iff p lies on the line segment from s to e. Intended for use with e.g. Point<long long> where overflow is an issue. Use (segDist(s,e,p)<=epsilon) instead when using Point<double>.
"Point.h" 5 lines
```
template <class P>
bool onSegment(const P& s, const P& e, const P& p) {
  P ds = p-s, de = p-e;
  return ds.cross(de) == 0 && ds.dot(de) <= 0;
}
```

linearTransformation.h

Description:



Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.
"Point.h" 6 lines
```
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
  P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
  return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

Angle.h

Description: A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.
Usage: vector<Angle> v = {w[0], w[0].t360() ...}; // sorted
```
int j = 0; FOR(i,0,n) { while (v[j] < v[i].t180()) ++j; }
// sweeps j such that (j-i) represents the number of positively
oriented triangles with vertices at 0 and i
```
37 lines
```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int quad() const {
    assert(x || y);
    if (y < 0) return (x >= 0) + 2;
    if (y > 0) return (x <= 0);
    return (x <= 0) * 2;
  }
  Angle t90() const { return {-y, x, t + (quad() == 3)}; }
  Angle t180() const { return {-x, -y, t + (quad() >= 2)}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.quad(), a.y * (ll)b.x) <
         make_tuple(b.t, b.quad(), a.x * (ll)b.y);
}

// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

CanFormTriangle.cpp

Description: Return true if you can form a triangle
Time: $\mathcal{O}(1)$
2 lines
```
bool canFormTriangle(double a, double b, double c) {
  return (a + b > c) && (a + c > b) && (b + c > a); }
```

CanFormQuadrangle.cpp

Description: Return true if you can form a quadrangle
Time: $\mathcal{O}(1)$
2 lines
```
bool canFormQuadrangle(double a, double b, double c,double d) {
  return (a + b + c> d) && (a + c + d > b) && (b + c + d > a)
       && (a+b+d>c); }
```

hasIntersectQuadrangles.cpp

Description: Return true if two quadrangles intersect
Time: $\mathcal{O}(1)$
7 lines
```
int hasIntersectQuadrangles(int lx, int ly, int rx, int ry, int
     la, int lb, int ra, int rb) {
    lx = lxsol = max(lx, la);
    ly = lysol = max(ly, lb);
    rx = rxsol = min(rx, ra);
    ry = rysol = min(ry, rb);
    return lx < rx && ly < ry;
}
```

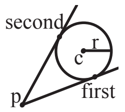
6.2 Circles

CircleIntersection.h

Description: Computes a pair of points at which two circles intersect. Returns false in case of no intersection.
"Point.h" 14 lines
```
typedef Point<double> P;
bool circleIntersection(P a, P b, double r1, double r2,
    pair<P, P>* out) {
  P delta = b - a;
  assert(delta.x || delta.y || r1 != r2);
  if (!delta.x && !delta.y) return false;
  double r = r1 + r2, d2 = delta.dist2();
  double p = (d2 + r1*r1 - r2*r2) / (2.0 * d2);
  double h2 = r1*r1 - p*p*d2;
  if (d2 > r*r || h2 < 0) return false;
  P mid = a + delta*p, per = delta.perp() * sqrt(h2 / d2);
  *out = {mid + per, mid - per};
  return true;
}
```

circleTangents.h

Description:



Returns a pair of the two points on the circle with radius r centered around c whos tangent lines intersect p. If p lies within the circle NaN-points are returned. P is intended to be Point<double>. The first point is the one to the right as seen from the p towards c.

Usage: typedef Point<double> P;
pair<P,P> p = circleTangents(P(100,2),P(0,0),2);

"Point.h" 6 lines

```
template <class P>
pair<P,P> circleTangents(const P &p, const P &c, double r) {
  P a = p-c;
  double x = r*r/a.dist2(), y = sqrt(x-x*x);
  return make_pair(c+a*x+a.perp()*y, c+a*x-a.perp()*y);
}
```

inCircle.cpp

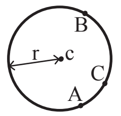
Description: Return incircle radio, area and perimeter of triangle.
Time: $\mathcal{O}(1)$

12 lines

```
double perimeter(double ab, double bc, double ca) {
  return ab + bc + ca; }

double area(double ab, double bc, double ca) {
  // Heron's formula
  double s = 0.5 * perimeter(ab, bc, ca);
  return sqrt(s) * sqrt(s - ab) * sqrt(s - bc) * sqrt(s - ca);
      }

double rInCircle(double ab, double bc, double ca) {
  double per = perimeter(ab, bc, ca);
  if(per==0) return 0;
  return area(ab, bc, ca) / (0.5 * per); }
```

circumcircle.h

Description:



The circumcirle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.

"Point.h" 10 lines

```
#define PI acos(-1.0)
typedef Point<double> P;
double ccRadius(const P& A, const P& B, const P& C) {
  return (B-A).dist()*(C-B).dist()*(A-C).dist()/
      abs((B-A).cross(C-A))/2;
}
P ccCenter(const P& A, const P& B, const P& C) {
  P b = C-A, c = B-A;
  return A + (b*c.dist2()-c*b.dist2()).perp()/b.cross(c)/2;
}
```

MinimumEnclosingCircle.h

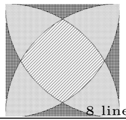
Description: Computes the minimum circle that encloses a set of points.
Time: expected $\mathcal{O}(n)$

"circumcircle.h" 28 lines

```
pair<double, P> mec2(vector<P>& S, P a, P b, int n) {
  double hi = INFINITY, lo = -hi;
  FOR(i,0,n) {
    auto si = (b-a).cross(S[i]-a);
    if (si == 0) continue;
    P m = ccCenter(a, b, S[i]);
    auto cr = (b-a).cross(m-a);
    if (si < 0) hi = min(hi, cr);
    else lo = max(lo, cr);
  }
  double v = (0 < lo ? lo : hi < 0 ? hi : 0);
  P c = (a + b) / 2 + (b - a).perp() * v / (b - a).dist2();
  return {(a - c).dist2(), c};
}
pair<double, P> mec(vector<P>& S, P a, int n) {
  random_shuffle(S.begin(), S.begin() + n);
  P b = S[0], c = (a + b) / 2;
  double r = (a - c).dist2();
  FOR(i,1,n) if ((S[i] - c).dist2() > r * (1 + 1e-8)) {
    tie(r,c) = (n == sz(S) ?
      mec(S, S[i], i) : mec2(S, a, S[i], i));
  }
  return {r, c};
}
pair<double, P> enclosingCircle(vector<P> S) {
  assert(!S.empty()); auto r = mec(S, S[0], sz(S));
  return {sqrt(r.first), r.second};
}
```

QuarterCircles.cpp

Description:



Returns de Area of the QuarterCircles

8 lines

```
#define PI acos(-1)
double a;
double x, y, z;
z = a*a - a*a*PI/4;
z -= a*a*PI/4 - a*a*PI/6 - ( a*a*PI/6 - a*a*sqrt(3.0)/4 ); //
   outside
y = a*a - a*a*PI/4 - 2*z; // "triangles"
x = a*a - 4*y - 4*z; //middle
printf("%.3lf %.3lf %.3lf\n", x, 4*y ,4*z);
```

insideCircle.cpp

Description: Return points inside circle
Time: $\mathcal{O}(1)$

4 lines

```
int insideCircle(point p, point c, double r) { // all integer
   version
  double dx = p.x - c.x, dy = p.y - c.y;
  double Euc = dx * dx + dy * dy, rSq = r * r; // all integer
  return Euc < rSq ? 0 : Euc == rSq ? 1 : 2; } //inside/border/
     outside
```

circle2PtsRad.h

Description: Given 2 points on the circle (p1 and p2) and radius r of the corresponding circle, we can determine the location of the centers (c1 and c2) of the two possible circles

9 lines

```
bool circle2PtsRad(point p1, point p2, double r, point &c) {
  double d2 = (p1.x - p2.x) * (p1.x - p2.x) +
              (p1.y - p2.y) * (p1.y - p2.y);
  double det = r * r / d2 - 0.25;
  if (det < 0.0) return false;
  double h = sqrt(det);
  c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
  c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
  return true; }
```

inCircumCircle.h

Description: Read below

25 lines

```
// returns 1 if there is a circumCenter center, returns 0
    otherwise
// if this function returns 1, ctr will be the circumCircle
    center
// and r is the same as rCircumCircle
int circumCircle(point p1, point p2, point p3, point &ctr,
    double &r){
  double a = p2.x - p1.x, b = p2.y - p1.y;
  double c = p3.x - p1.x, d = p3.y - p1.y;
  double e = a * (p1.x + p2.x) + b * (p1.y + p2.y);
  double f = c * (p1.x + p3.x) + d * (p1.y + p3.y);
  double g = 2.0 * (a * (p3.y - p2.y) - b * (p3.x - p2.x));
  if (fabs(g) < EPS) return 0;

  ctr.x = (d*e - b*f) / g;
  ctr.y = (a*f - c*e) / g;
  r = dist(p1, ctr);  // r = distance from center to 1 of the 3
      points
  return 1; }

// returns true if point d is inside the circumCircle defined
    by a,b,c
int inCircumCircle(point a, point b, point c, point d) {
  return (a.x - d.x) * (b.y - d.y) * ((c.x - d.x) * (c.x - d.x)
        + (c.y - d.y) * (c.y - d.y)) +
         (a.y - d.y) * ((b.x - d.x) * (b.x - d.x) + (b.y - d.y)
            * (b.y - d.y)) * (c.x - d.x) +
         ((a.x - d.x) * (a.x - d.x) + (a.y - d.y) * (a.y - d.y)
           ) * (b.x - d.x) * (c.y - d.y) -
         ((a.x - d.x) * (a.x - d.x) + (a.y - d.y) * (a.y - d.y)
           ) * (b.y - d.y) * (c.x - d.x) -
         (a.y - d.y) * (b.x - d.x) * ((c.x - d.x) * (c.x - d.x)
            + (c.y - d.y) * (c.y - d.y)) -
         (a.x - d.x) * ((b.x - d.x) * (b.x - d.x) + (b.y - d.y)
            * (b.y - d.y)) * (c.y - d.y) > 0 ? 1 : 0;
}
```

6.3 Polygons

insidePolygon.h

Description: Returns true if p lies within the polygon described by the points between iterators begin and end. If strict false is returned when p is on the edge of the polygon. Answer is calculated by counting the number of intersections between the polygon and a line going from p to infinity in the positive x-direction. The algorithm uses products in intermediate steps so watch out for overflow. If points within epsilon from an edge should be considered as on the edge replace the line "if (onSegment..." with the comment bellow it (this will cause overflow for int and long long).

Usage: typedef Point<int> pi;
vector<pi> v; v.push_back(pi(4,4));
v.push_back(pi(1,2)); v.push_back(pi(2,1));
bool in = insidePolygon(v.begin(),v.end(), pi(3,4), false);
Time: $\mathcal{O}(n)$

"Point.h", "onSegment.h", "SegmentDistance.h" 14 lines

```
template <class It, class P>
bool insidePolygon(It begin, It end, const P& p,
    bool strict = true) {
  int n = 0; //number of isects with line from p to (inf,p.y)
  for (It i = begin, j = end-1; i != end; j = i++) {
    //if p is on edge of polygon
    if (onSegment(*i, *j, p)) return !strict;
    //or: if (segDist(*i, *j, p) <= epsilon) return !strict;
    //increment n if segment intersects line from p
    n += (max(i->y,j->y) > p.y && min(i->y,j->y) <= p.y &&
        ((*j-*i).cross(p-*i) > 0) == (i->y <= p.y));
  }
  return n&1; //inside if odd number of intersections
}
```

PolygonArea.h

Description: Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

"Point.h" 6 lines

```
template <class T>
T polygonArea2(vector<Point<T>>& v) {
  T a = v.back().cross(v[0]);
  FOR(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
  return a;
}
```

PolygonAreav2.h

Description: Return polygon area. 6 lines

```
double calc_area(vector<P<double>> Pa) {
    double ans = 0;
    for(int i = 0; i < (int)Pa.size()-1; i++)
        ans += Pa[i].x*Pa[i+1].y - Pa[i].y*Pa[i+1].x;
    return fabs(ans)/2.0;
}
```

PolygonCenter.h

Description: Returns the center of mass for a polygon.

"Point.h" 10 lines

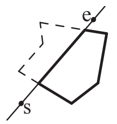
```
typedef Point<double> P;
Point<double> polygonCenter(vector<P>& v) {
  auto i = v.begin(), end = v.end(), j = end-1;
  Point<double> res{0,0}; double A = 0;
  for (; i != end; j=i++) {
    res = res + (*i + *j) * j->cross(*i);
    A += j->cross(*i);
  }
  return res / A / 3;
}
```

PolygonCenterOfMass.h

Description: Return center of mass 18 lines

```
template <class P>
P centroid(vector<P> g)   //center of mass
{
    double cx = 0.0, cy = 0.0;
    for(unsigned int i = 0; i < g.size() - 1; i++)
    {
        double x1 = g[i].x, y1 = g[i].y;
        double x2 = g[i+1].x, y2 = g[i+1].y;

        double f = x1 * y2 - x2 * y1;
        cx += (x1 + x2) * f;
        cy += (y1 + y2) * f;
    }
    double res = calc_area(g);    //remove abs
    cx /= 6.0 * res;
    cy /= 6.0 * res;
    return P(cx, cy);
}
```

PolygonCut.h



Description:
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.
Usage: vector<P> p = ...;
p = polygonCut(p, P(0,0), P(1,0));
"Point.h", "lineIntersection.h" 15 lines

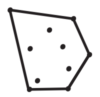
```
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  FOR(i,0,sz(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0)) {
      res.emplace_back();
      lineIntersection(s, e, cur, prev, res.back());
    }
    if (side)
      res.push_back(cur);
  }
  return res;
}
```

PolygonConvex.cpp

Description: Returns if the polygon it is convex
Time: $\mathcal{O}(N)$ 16 lines

```
double cross(vec a, vec b) { return a.x * b.y - a.y * b.x; }
// note: to accept collinear points, we have to change the > 0
// returns true if point r is on the left side of line pq
bool ccw(point p, point q, point r) {
  return cross(toVec(p, q), toVec(p, r)) > 0; }
// returns true if point r is on the same line as the line pq
bool collinear(point p, point q, point r) {
  return fabs(cross(toVec(p, q), toVec(p, r))) < EPS; }
bool isConvex(const vector<point> &P) {
  int sz = (int)P.size();
  if (sz <= 3) return false;
  bool isLeft = ccw(P[0], P[1], P[2]);
  for (int i = 1; i < sz-1; i++)
    if (ccw(P[i], P[i+1], P[(i+2) == sz ? 1 : i+2]) != isLeft)
      return false;
  return true; }
```

ConvexHull.h



Description:
Returns a vector of indices of the convex hull in counter-clockwise order. Points on the edge of the hull between two other points are not considered part of the hull.
Usage: vector<P> ps, hull;
trav(i, convexHull(ps)) hull.push_back(ps[i]);
Time: $\mathcal{O}(n \log n)$
"Point.h" 20 lines

```
typedef Point<ll> P;
pair<vi, vi> ulHull(const vector<P>& S) {
  vi Q(sz(S)), U, L;
  iota(all(Q), 0);
  sort(all(Q), [&S](int a, int b){ return S[a] < S[b]; });
  trav(it, Q) {
#define ADDP(C, cmp) while (sz(C) > 1 && S[C[sz(C)-2]].cross(\
  S[it], S[C.back()]) cmp 0) C.pop_back(); C.push_back(it);
    ADDP(U, <=); ADDP(L, >=);
  }
  return {U, L};
}

vi convexHull(const vector<P>& S) {
  vi u, l; tie(u, l) = ulHull(S);
  if (sz(S) <= 1) return u;
  if (S[u[0]] == S[u[1]]) return {0};
  l.insert(l.end(), u.rbegin()+1, u.rend()-1);
  return l;
}
```

newConvexHull.cpp

Description: Return a vector with the convexhull / convert to array if TLE
Time: $\mathcal{O}(Nlog(N))$ 20 lines

```
template <class P>
double cross(P o, P a, P b) {
    return (a.x-o.x)*(b.y-o.y) - (a.y-o.y)*(b.x-o.x);

template <class P>
vector<P> CH(vector<P> Pa){
    vector<P> res;
    sort(Pa.begin(),Pa.end());
    int n = Pa.size();
    int m=0;
    for (int i=0;i<n;i++){
        while (m>1&&cross(res[m-2], res[m-1], Pa[i]) <= 0)res.
            pop_back(),m--;
        res.push_back(Pa[i]),m++;
    }
    for (int i = n-1, t = m+1; i >= 0; i--) {
        while (m>=t&&cross(res[m-2], res[m-1], Pa[i]) <= 0)res.
            pop_back(),m--;
        res.push_back(Pa[i]),m++;
    }
    return res;
}
```

PolygonPerimeter.cpp

Description: Returns the perimeter of a polygon
Time: $\mathcal{O}(N)$ 10 lines

```
template <class P>
double dist(P p1, P p2) {
  return hypot(p1.x - p2.x, p1.y - p2.y); }

template <class P>
double perimeter(const vector<P> &Pa) {
  double result = 0.0;
  for (int i = 0; i < (int)Pa.size()-1; i++)
    result += dist(Pa[i], Pa[i+1]);
  return result; }
```

PolygonDiameter.h

Description: Calculates the max squared distance of a set of points.
"ConvexHull.h" 19 lines

```
vector<pii> antipodal(const vector<P>& S, vi& U, vi& L) {
  vector<pii> ret;
  int i = 0, j = sz(L) - 1;
  while (i < sz(U) - 1 || j > 0) {
    ret.emplace_back(U[i], L[j]);
    if (j == 0 || (i != sz(U)-1 && (S[L[j]] - S[L[j-1]])
          .cross(S[U[i+1]] - S[U[i]]) > 0)) ++i;
    else --j;
  }
  return ret;
}

pii polygonDiameter(const vector<P>& S) {
  vi U, L; tie(U, L) = ulHull(S);
  pair<ll, pii> ans;
  trav(x, antipodal(S, U, L))
    ans = max(ans, {(S[x.first] - S[x.second]).dist2(), x});
  return ans.second;
}
```

PointInsideHull.h

Description: Determine whether a point t lies inside a given polygon (counter-clockwise order). The polygon must be such that every point on the circumference is visible from the first point in the vector. It returns 0 for points outside, 1 for points on the circumference, and 2 for points inside.
Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "onSegment.h" 22 lines

```
typedef Point<ll> P;
int insideHull2(const vector<P>& H, int L, int R, const P& p) {
  int len = R - L;
  if (len == 2) {
    int sa = sideOf(H[0], H[L], p);
    int sb = sideOf(H[L], H[L+1], p);
    int sc = sideOf(H[L+1], H[0], p);
    if (sa < 0 || sb < 0 || sc < 0) return 0;
    if (sb==0 || (sa==0 && L == 1) || (sc == 0 && R == sz(H)))
      return 1;
    return 2;
  }
  int mid = L + len / 2;
  if (sideOf(H[0], H[mid], p) >= 0)
    return insideHull2(H, mid, R, p);
  return insideHull2(H, L, mid+1, p);
}

int insideHull(const vector<P>& hull, const P& p) {
  if (sz(hull) < 3) return onSegment(hull[0], hull.back(), p);
  else return insideHull2(hull, 1, sz(hull), p);
}
```

LineHullIntersection.h

Description: Line-convex polygon intersection. The polygon must be ccw and have no colinear points. isct(a, b) returns a pair describing the intersection of a line with the polygon: • $(-1,-1)$ if no collision, • $(i,-1)$ if touching the corner i, • (i,i) if along side $(i,i+1)$, • (i,j) if crossing sides $(i,i+1)$ and $(j,j+1)$. In the last case, if a corner i is crossed, this is treated as happening on side $(i,i+1)$. The points are returned in the same order as the line hits the polygon.
Time: $\mathcal{O}(N+Q\log n)$

"Point.h" 63 lines

```
ll sgn(ll a) { return (a > 0) - (a < 0); }
typedef Point<ll> P;
struct HullIntersection {
  int N;
  vector<P> p;
  vector<pair<P, int>> a;

  HullIntersection(const vector<P>& ps) : N(sz(ps)), p(ps) {
    p.insert(p.end(), all(ps));
    int b = 0;
    FOR(i,1,N) if (P{p[i].y,p[i].x} < P{p[b].y, p[b].x}) b = i;
    FOR(i,0,N) {
      int f = (i + b) % N;
      a.emplace_back(p[f+1] - p[f], f);
    }
  }

  int qd(P p) {
    return (p.y < 0) ? (p.x >= 0) + 2
         : (p.x <= 0) * (1 + (p.y <= 0));
  }

  int bs(P dir) {
    int lo = -1, hi = N;
    while (hi - lo > 1) {
      int mid = (lo + hi) / 2;
      if (make_pair(qd(dir), dir.y * a[mid].first.x) <
        make_pair(qd(a[mid].first), dir.x * a[mid].first.y))
        hi = mid;
      else lo = mid;
    }
    return a[hi%N].second;
  }

  bool isign(P a, P b, int x, int y, int s) {
    return sgn(a.cross(p[x], b)) * sgn(a.cross(p[y], b)) == s;
  }

  int bs2(int lo, int hi, P a, P b) {
    int L = lo;
    if (hi < lo) hi += N;
    while (hi - lo > 1) {
      int mid = (lo + hi) / 2;
      if (isign(a, b, mid, L, -1)) hi = mid;
      else lo = mid;
    }
    return lo;
  }

  pii isct(P a, P b) {
    int f = bs(a - b), j = bs(b - a);
    if (isign(a, b, f, j, 1)) return {-1, -1};
    int x = bs2(f, j, a, b)%N,
        y = bs2(j, f, a, b)%N;
    if (a.cross(p[x], b) == 0 &&
        a.cross(p[x+1], b) == 0) return {x, x};
    if (a.cross(p[y], b) == 0 &&
        a.cross(p[y+1], b) == 0) return {y, y};
    if (a.cross(p[f], b) == 0) return {f, -1};
    if (a.cross(p[j], b) == 0) return {j, -1};
    return {x, y};
  }
};
```

6.4 Misc. Point Set Problems

closestPair.h

Description: $i1$, $i2$ are the indices to the closest pair of points in the point vector p after the call. The distance is returned.
Time: $\mathcal{O}(n\log n)$

"Point.h" 58 lines

```
template <class It>
bool it_less(const It& i, const It& j) { return *i < *j; }
template <class It>
bool y_it_less(const It& i,const It& j) {return i->y < j->y;}

template<class It, class IIt> /* IIt = vector<It>::iterator */
double cp_sub(IIt ya, IIt yaend, IIt xa, It &i1, It &i2) {
  typedef typename iterator_traits<It>::value_type P;
  int n = yaend-ya, split = n/2;
  if(n <= 3) { // base case
    double a = (*xa[1]-*xa[0]).dist(), b = 1e50, c = 1e50;
    if(n==3) b=(*xa[2]-*xa[0]).dist(), c=(*xa[2]-*xa[1]).dist()
      ;
    if(a <= b) { i1 = xa[1];
      if(a <= c) return i2 = xa[0], a;
      else return i2 = xa[2], c;
    } else { i1 = xa[2];
      if(b <= c) return i2 = xa[0], b;
      else return i2 = xa[1], c;
  } }
  vector<It> ly, ry, stripy;
  P splitp = *xa[split];
  double splitx = splitp.x;
  for(IIt i = ya; i != yaend; ++i) { // Divide
    if(*i != xa[split] && (**i-splitp).dist2() < 1e-12)
      return i1 = *i, i2 = xa[split], 0;// nasty special case!
    if (**i < splitp) ly.push_back(*i);
    else ry.push_back(*i);
  } // assert((signed)lefty.size() == split)
  It j1, j2; // Conquer
  double a = cp_sub(ly.begin(), ly.end(), xa, i1, i2);
  double b = cp_sub(ry.begin(), ry.end(), xa+split, j1, j2);
  if(b < a) a = b, i1 = j1, i2 = j2;
  double a2 = a*a;
  for(IIt i = ya; i != yaend; ++i) { // Create strip (y-sorted)
    double x = (*i)->x;
    if(x >= splitx-a && x <= splitx+a) stripy.push_back(*i);
  }
  for(IIt i = stripy.begin(); i != stripy.end(); ++i) {
    const P &p1 = **i;
    for(IIt j = i+1; j != stripy.end(); ++j) {
      const P &p2 = **j;
      if(p2.y-p1.y > a) break;
      double d2 = (p2-p1).dist2();
      if(d2 < a2) i1 = *i, i2 = *j, a2 = d2;
  } }
  return sqrt(a2);
}

template<class It> // It is random access iterators of point<T>
double closestpair(It begin, It end, It &i1, It &i2 ) {
  vector<It> xa, ya;
  assert(end-begin >= 2);
  for (It i = begin; i != end; ++i)
    xa.push_back(i), ya.push_back(i);
  sort(xa.begin(), xa.end(), it_less<It>);
  sort(ya.begin(), ya.end(), y_it_less<It>);
  return cp_sub(ya.begin(), ya.end(), xa.begin(), i1, i2);
}
```

HeronWithMedians.cpp

Description: Heron Theorem with medians 4/3 12 lines

```
int main()
{
    double d1,d2,d3;
    while(scanf("%lf%lf%lf",&d1,&d2,&d3)==3)
    {
        double res = (d1+d2+d3)/2;
        res=(res-d1)*(res-d2)*(res-d3)*res;
        if(res<EPS) printf("-1.000\n");
        else printf("%.3lf\n",sqrt(res)*4/3);
    }
    return 0;
}
```

6.5 3D

PolyhedronVolume.h

Description: Magic formula for the volume of a polyhedron. Faces should point outwards. 6 lines

```
template <class V, class L>
double signed_poly_volume(const V& p, const L& trilist) {
  double v = 0;
  trav(i, trilist) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
  return v / 6;
}
```

Point3D.h

Description: Class to handle points in 3D space. T can be e.g. double or long long. 32 lines

```
template <class T> struct Point3D {
  typedef Point3D P;
  typedef const P& R;
  T x, y, z;
  explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
  bool operator<(R p) const {
    return tie(x, y, z) < tie(p.x, p.y, p.z); }
  bool operator==(R p) const {
    return tie(x, y, z) == tie(p.x, p.y, p.z); }
  P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
  P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
  P operator*(T d) const { return P(x*d, y*d, z*d); }
  P operator/(T d) const { return P(x/d, y/d, z/d); }
  T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
  P cross(R p) const {
    return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
  }
  T dist2() const { return x*x + y*y + z*z; }
  double dist() const { return sqrt((double)dist2()); }
  //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
  double phi() const { return atan2(y, x); }
  //Zenith angle (latitude) to the z-axis in interval [0, pi]
  double theta() const { return atan2(sqrt(x*x+y*y),z); }
  P unit() const { return *this/(T)dist(); } //makes dist()=1
  //returns unit vector normal to *this and p
  P normal(P p) const { return cross(p).unit(); }
  //returns point rotated 'angle' radians ccw around axis
  P rotate(double angle, P axis) const {
    double s = sin(angle), c = cos(angle); P u = axis.unit();
    return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
  }
};
```

sphericalDistance.h

Description: Returns the shortest distance on the sphere with radius radius between the points with azimuthal angles (longitude) f1 (ϕ_1) and f2 (ϕ_2) from x axis and zenith angles (latitude) t1 (θ_1) and t2 (θ_2) from z axis. All angles measured in radians. The algorithm starts by converting the spherical coordinates to cartesian coordinates so if that is what you have you can use only the two last rows. dx*radius is then the difference between the two points in the x direction and d*radius is the total distance between the points.

8 lines

```
double sphericalDistance(double f1, double t1,
    double f2, double t2, double radius) {
  double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
  double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
  double dz = cos(t2) - cos(t1);
  double d = sqrt(dx*dx + dy*dy + dz*dz);
  return radius*2*asin(d/2);
}
```

Numerical (7)

Polynomial.h

17 lines

```
struct Poly {
  vector<double> a;
  double operator()(double x) const {
    double val = 0;
    for(int i = sz(a) - 1; i--;) (val *= x) += a[i];
    return val;
  }
  void diff() {
    FOR(i,1,sz(a)) a[i-1] = i*a[i];
    a.pop_back();
  }
  void divroot(double x0) {
    double b = a.back(), c; a.back() = 0;
    for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
    a.pop_back();
  }
};
```

PolyRoots.h

Description: Finds the real roots to a polynomial.
Usage: `poly_roots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0`
Time: $\mathcal{O}\left(n^2\log(1/\epsilon)\right)$

"Polynomial.h" 23 lines

```
vector<double> poly_roots(Poly p, double xmin, double xmax) {
  if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
  vector<double> ret;
  Poly der = p;
  der.diff();
  auto dr = poly_roots(der, xmin, xmax);
  dr.push_back(xmin-1);
  dr.push_back(xmax+1);
  sort(all(dr));
  FOR(i,0,sz(dr)-1) {
    double l = dr[i], h = dr[i+1];
    bool sign = p(l) > 0;
    if (sign ^ (p(h) > 0)) {
      FOR(it,0,60) { // while (h - l > 1e-8)
        double m = (l + h) / 2, f = p(m);
        if ((f <= 0) ^ sign) l = m;
        else h = m;
      }
      ret.push_back((l + h) / 2);
    }
  }
  return ret;
}
```

PolyInterpolate.h

Description: Given n points (x[i], y[i]), computes an n-1-degree polynomial p that passes through them: $p(x) = a[0] * x^0 + ... + a[n-1] * x^{n-1}$. For numerical precision, pick $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \dots n-1$.
Time: $\mathcal{O}\left(n^2\right)$

13 lines

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd res(n), temp(n);
  FOR(k,0,n-1) FOR(i,k+1,n)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  double last = 0; temp[0] = 1;
  FOR(k,0,n) FOR(i,0,n) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
}
```

MatrixInverse.h

Description: Invert matrix A. Returns rank; result is stored in A unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.
Time: $\mathcal{O}\left(n^3\right)$

35 lines

```
int matInv(vector<vector<double>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<double>> tmp(n, vector<double>(n));
  FOR(i,0,n) tmp[i][i] = 1, col[i] = i;

  FOR(i,0,n) {
    int r = i, c = i;
    FOR(j,i,n) FOR(k,i,n)
      if (fabs(A[j][k]) > fabs(A[r][c]))
        r = j, c = k;
    if (fabs(A[r][c]) < 1e-12) return i;
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    FOR(j,0,n)
      swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
    swap(col[i], col[c]);
    double v = A[i][i];
    FOR(j,i+1,n) {
      double f = A[j][i] / v;
      A[j][i] = 0;
      FOR(k,i+1,n) A[j][k] -= f*A[i][k];
      FOR(k,0,n) tmp[j][k] -= f*tmp[i][k];
    }
    FOR(j,i+1,n) A[i][j] /= v;
    FOR(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
  }

  for (int i = n-1; i > 0; --i) FOR(j,0,i) {
    double v = A[j][i];
    FOR(k,0,n) tmp[j][k] -= v*tmp[i][k];
  }

  FOR(i,0,n) FOR(j,0,n) A[col[i]][col[j]] = tmp[i][j];
  return n;
}
```

SolveLinear.h

Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.
Time: $\mathcal{O}\left(n^2 m\right)$

39 lines

```
typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
  int n = A.size(), m = x.size(), rank = 0, br, bc;
  if (n) assert(A[0].size() == m);
  // FOR(i, 0, n) FOR(j, 0, m) A[i][j] %= MOD; also b[i]...
  vi col(m); iota(col.begin(), col.end(), 0);

  FOR(i,0,n) {
    double v, bv = 0;
    FOR(r,i,n) FOR(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
      FOR(j,i,n) if (fabs(b[j]) > eps) return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    FOR(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    FOR(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      FOR(k,i,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }

  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    FOR(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

SolveLinear2.h

Description: To get all uniquely determined values of x back from SolveLinear, make the following changes:

"SolveLinear.h" 7 lines

```
FOR(j,0,n) if (j != i) // instead of FOR(j,i+1,n)
// ... then at the end:
x.assign(m, undefined);
FOR(i,0,rank) {
  FOR(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
  x[col[i]] = b[i] / A[i][i];
fail:; }
```

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.
Time: $\mathcal{O}(N^3)$ 33 lines

```
double det(vector<vector<double>>& a) {
  int n = sz(a); double res = 1;
  FOR(i,0,n) {
    int b = i;
    FOR(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
    if (i != b) swap(a[i], a[b]), res *= -1;
    res *= a[i][i];
    if (res == 0) return 0;
    FOR(j,i+1,n) {
      double v = a[j][i] / a[i][i];
      if (v != 0) FOR(k,i+1,n) a[j][k] -= v * a[i][k];
    }
  }
  return res;
}

ll det(vector<vector<ll>>& a, ll mod) {
  int n = sz(a); ll ans = 1;
  FOR(i,0,n) {
    FOR(j,i+1,n) {
      while (a[j][i] != 0) { // gcd step
        ll t = a[i][i] / a[j][i];
        if (t) FOR(k,i,n)
          a[i][k] = (a[i][k] - a[j][k] * t) % mod;
        swap(a[i], a[j]);
        ans *= -1;
      }
    }
    ans = ans * a[i][i] % mod;
    if (!ans) return 0;
  }
  return (ans + mod) % mod;
}
```

Number theory (8)

8.1 Modular arithmetic

ModularArithmetic.h

Description: Operators for modular arithmetic. You need to set mod to some number first and then you can use the structure.

"euclid.h" 18 lines

```
const ll mod = 17; // change to something else
struct Mod {
  ll x;
  Mod(ll xx) : x(xx) {}
  Mod operator+(Mod b) { return Mod((x + b.x) % mod); }
  Mod operator-(Mod b) { return Mod((x - b.x + mod) % mod); }
  Mod operator*(Mod b) { return Mod((x * b.x) % mod); }
  Mod operator/(Mod b) { return *this * invert(b); }
  Mod invert(Mod a) {
    ll x, y, g = euclid(a.x, mod, x, y);
    assert(g == 1); return Mod((x + mod) % mod);
  }
  Mod operator^(ll e) {
    if (!e) return Mod(1);
    Mod r = *this ^ (e / 2); r = r * r;
    return e&1 ? *this * r : r;
  }
};
```

ModInverse.h

Description: Pre-computation of modular inverses. Assumes LIM $\leq$ mod and that mod is a prime. 3 lines

```
const ll mod = 1000000007, LIM = 200000;
ll* inv = new ll[LIM] - 1; inv[1] = 1;
FOR(i,2,LIM) inv[i] = mod - (mod / i) * inv[mod % i] % mod;
```

ModPow.h

6 lines

```
ll modpow(ll a, ll e, const ll mod) {
  ll cur = 1;
  for(;e;e >>= 1, a = (a*a)%mod){
    if (e&1) {cur *= e; cur %= mod;}
  } return cur;
}
```

ModSum.h

Description: Sums of mod'ed arithmetic progressions.
modsum(to, c, k, m) $= \sum_{i=0}^{\text{to}-1}(ki+c)\%m$. divsum is similar but for floored division.
Time: $\log(m)$, with a large constant. 16 lines

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }

ull divsum(ull to, ull c, ull k, ull m) {
  ull res = k / m * sumsq(to) + c / m * to;
  k %= m; c %= m;
  if (!k) return res;
  ull to2 = (to * k + c) / m;
  return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}

ll modsum(ull to, ll c, ll k, ll m) {
  c = ((c % m) + m) % m;
  k = ((k % m) + m) % m;
  return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

ModMulLL.h

Description: Calculate $a \cdot b \bmod c$ (or $a^b \bmod c$) for large c.
Time: $\mathcal{O}(64/bits \cdot \log b)$, where $bits = 64 - k$, if we want to deal with k-bit numbers. 19 lines

```
typedef unsigned long long ull;
const int bits = 10;
// if all numbers are less than 2^k, set bits = 64-k
const ull po = 1 << bits;
ull mod_mul(ull a, ull b, ull &c) {
  ull x = a * (b & (po - 1)) % c;
  while ((b >>= bits) > 0) {
    a = (a << bits) % c;
    x += (a * (b & (po - 1))) % c;
  }
  return x % c;
}
ull mod_pow(ull a, ull b, ull mod) {
  if (b == 0) return 1;
  ull res = mod_pow(a, b / 2, mod);
  res = mod_mul(res, res, mod);
  if (b & 1) return mod_mul(res, a, mod);
  return res;
}
```

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots.
Time: $\mathcal{O}(\log^2 p)$ worst case, often $\mathcal{O}(\log p)$

"ModPow.h" 24 lines

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(modpow(a, (p-1)/2, p) == 1);
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0)
    ++r, s /= 2;
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p), g = modpow(n, s, p);
  for (;; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m)
      t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

8.2 Primality

eratosthenes.h

Description: Prime sieve for generating all primes up to a certain limit. isprime$[i]$ is true iff i is a prime.
Time: lim=100'000'000 $\approx$ 0.8 s. Runs 30% faster if only odd indices are stored. 11 lines

```
const int MAX_PR = 5000000;
bitset<MAX_PR> isprime;
vi eratosthenes_sieve(int lim) {
  isprime.set(); isprime[0] = isprime[1] = 0;
  for (int i = 4; i < lim; i += 2) isprime[i] = 0;
  for (int i = 3; i*i < lim; i += 2) if (isprime[i])
    for (int j = i*i; j < lim; j += i*2) isprime[j] = 0;
  vi pr;
  FOR(i,2,lim) if (isprime[i]) pr.push_back(i);
  return pr;
}
```

MillerRabin.h

Description: Miller-Rabin primality probabilistic test. Probability of failing one iteration is at most 1/4. 15 iterations should be enough for 50-bit numbers.
Time: 15 times the complexity of $a^b \bmod c$.

"ModMulLL.h" 16 lines

```
bool prime(ull p) {
  if (p == 2) return true;
  if (p == 1 || p % 2 == 0) return false;
  ull s = p - 1;
  while (s % 2 == 0) s /= 2;
  FOR(i,0,15) {
    ull a = rand() % (p - 1) + 1, tmp = s;
    ull mod = mod_pow(a, tmp, p);
    while (tmp != p - 1 && mod != 1 && mod != p - 1) {
      mod = mod_mul(mod, mod, p);
      tmp *= 2;
    }
    if (mod != p - 1 && tmp % 2 == 0) return false;
  }
  return true;
}
```

factor.h

Description: Pollard's rho algorithm. It is a probabilistic factorisation algorithm, whose expected time complexity is good. Before you start using it, run `init(bits)`, where bits is the length of the numbers you use. to get factor multiple times, uncomment comments with (*)

Time: Expected running time should be good enough for 50-bit numbers.

"MillerRabin.h", "eratosthenes.h", "euclid.h" 37 lines

```
vector<ull> pr;
ull f(ull a, ull n, ull &has) {
  return (mod_mul(a, a, n) + has) % n;
}
vector<ull> factor(ull d) {
  vector<ull> res;
  for (size_t i = 0; i < pr.size() && pr[i]*pr[i] <= d; i++)
    if (d % pr[i] == 0) {
      while (d % pr[i] == 0) /*{ */ d /= pr[i];
      res.push_back(pr[i]); /*} (*)*/
    }
  //d is now a product of at most 2 primes.
  if (d > 1) {
    if (prime(d))
      res.push_back(d);
    else while (true) {
      ull has = rand() % 2321 + 47;
      ull x = 2, y = 2, c = 1;
      for (; c==1; c = gcd((y > x ? y - x : x - y), d)) {
        x = f(x, d, has);
        y = f(f(y, d, has), d, has);
      }
      if (c != d) {
        res.push_back(c); d /= c;
        if (d != c /* || true (*)*/) res.push_back(d);
        break;
      }
    }
  }
  return res;
}
void init(int bits) {//how many bits do we use?
  vi p = eratosthenes_sieve(1 << ((bits + 2) / 3));
  pr.resize(p.size());
  for (size_t i=0; i<pr.size(); i++)
    pr[i] = p[i];
}
```

8.3 Divisibility

euclid.h

Description: Finds the Greatest Common Divisor to the integers a and b. Euclid also finds two integers x and y, such that $ax + by = \gcd(a, b)$. If a and b are coprime, then x is the inverse of $a \pmod b$.

7 lines

```
ll gcd(ll a, ll b) { return __gcd(a, b); }
ll euclid(ll a, ll b, ll &x, ll &y) {
  if (b) { ll d = euclid(b, a % b, y, x);
    return y -= a/b * x, d; }
  return x = 1, y = 0, a;
}
```

8.4 Primes

$p = 962592769$ is such that $2^{21} \mid p-1$, which may be useful. For hashing use 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). There are 78498 primes less than 1 000 000.

Primitive roots exist modulo any prime power p^a, except for $p = 2, a > 2$, and there are $\phi(\phi(p^a))$ many. For $p = 2, a > 2$, the group $\mathbb{Z}_{2^a}^\times$ is instead isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

8.5 Estimates

$\sum_{d|n} d = O(n \log\log n)$.

The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 200 000 for $n < 1e19$.

Combinatorial (9)

9.1 Partitions and subsets

9.1.1 Partition function

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1,\ p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	~2e5	~2e8

9.1.2 Binomials

binomialModPrime.h

Description: Lucas' thm: Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k} \binom{n_i}{m_i} \pmod p$. fact and invfact must hold pre-computed factorials / inverse factorials, e.g. from ModInverse.h.

Time: $\mathcal{O}\left(\log_p n\right)$

10 lines

```
ll chooseModP(ll n, ll m, int p, vi& fact, vi& invfact) {
  ll c = 1;
  while (n || m) {
    ll a = n % p, b = m % p;
    if (a < b) return 0;
    c = c * fact[a] % p * invfact[b] % p * invfact[a - b] % p;
    n /= p; m /= p;
  }
  return c;
}
```

9.2 General purpose numbers

9.2.1 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k),\ c(0,0) = 1$$
$$\textstyle\sum_{k=0}^{n} c(n,k)x^k = x(x+1)\ldots(x+n-1)$$

$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$

9.2.2 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$

$$E(n,0) = E(n,n-1) = 1$$

$$E(n,k) = \sum_{j=0}^{k} (-1)^j \binom{n+1}{j} (k+1-j)^n$$

9.2.3 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

$$S(n,k) = S(n-1,k-1) + kS(n-1,k)$$

$$S(n,1) = S(n,n) = 1$$

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

9.2.4 Bell numbers

Total number of partitions of n distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$. For p prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod p$$

9.2.5 Catalan numbers

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

$$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2} C_n,\ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$

- sub-diagonal monotone paths in an $n \times n$ grid.

- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

Various (10)

10.1 Misc. algorithms

TernarySearch.cpp

Description: Find the smallest i in $[a, b]$ that maximizes $f(i)$, assuming that $f(a) < \ldots < f(i) \geq \cdots \geq f(b)$. To reverse which of the sides allows non-strict inequalities, change the $<$ marked with (A) to $<=$, and reverse the loop at (B). To minimize f, change it to $>$, also at (B).
Usage: `int ind = ternSearch(0,n-1,[&](int i){return a[i];});`
Time: $\mathcal{O}(\log(b-a))$

13 lines

```
template<class F>
int ternSearch(int a, int b, F f) {
  assert(a <= b);
  while (b - a >= 5) {
    int mid = (a + b) / 2;
    if (f(mid) < f(mid+1)) // (A)
      a = mid;
    else
      b = mid+1;
  }
  FOR(i,a+1,b+1) if (f(a) < f(i)) a = i; // (B)
  return a;
}
```

matFastPow.cpp

Description: FastPow of any size (square) matrix given as vector<vector<ll>>, resulting values are modulus mod. Also contains matMul which works with any matrix with dimensions (m,n)(k,j) if n equals k;

30 lines

```
typedef vector<ll> vl;
vector<vl> createMat(int s, bool ide){
    //create 0 matrix (false) or identity matrix (true)
    vector<vl> mat(s);
    for(int i=0; i<s;i++){
        for(int j=0;j<s;j++) mat[i].push_back((i==j&&ide)?1:0)
            ;}
    return mat;
}
vector<vl> matMul(vector<vl> m1, vector<vl> m2, int mod){
    vector<vl> res = createMat(m2.size(),false);
    for (int i = 0; i < m1.size(); ++i)
        for (int j = 0; j < m2[0].size(); ++j)
            for (int k = 0; k < m2.size(); ++k) {
                res[i][j]+=m1[i][k]*m2[k][j];
                res[i][j]%=mod;
            }
    return res;
}
vector<vl> matPow(vector<vl> m, int pow, int mod) {
    if (pow == 0) return createMat(m.size(), true);
    if (pow == 1) return m;
    if (pow == 2) return matMul(m,m,mod);
    if (pow%2==0){
        vector<vl> aux = matPow(m,pow/2,mod);
        return matMul(aux,aux,mod);
    } else {
        vector<vl> aux = matPow(m,pow-1,mod);
        return matMul(aux,m,mod);
    }
}
```

BITRange.cpp

Description: BIT for range updates.
Usage: `Starts at 1`

24 lines

```
typedef long long ll;
ll B1[100005], B2[100005];
int N=100000; //arraysize

ll query(ll* ft, int b) {
  ll sum = 0;
  for (; b > 0; b -= (b&-b)) sum += ft[b];
  return sum;
}
void update(ll* ft, int k, ll v) {
  for (; k <= N; k += (k&-k)) ft[k] += v;
}
ll query(int n) { // sum from 1 to n
  return query(B1, n) * n - query(B2, n);
}
ll range_query(int i, int j){ //sum from i to j (both incl)
  return query(j) - query(i - 1);
}
void range_update(int i, int j, ll v){//to update point use i==
     j
  update(B1, i, v);
  update(B1, j + 1, -v);
  update(B2, i, v * (i - 1));
  update(B2, j + 1, -v * j);
}
```

NextPermutation.cpp

Description: Next Permutation

20 lines

```
int main() {
    int casos;
    scanf("%d", &casos);
    while (casos--) {
        int C, V, A[16] = {};
        char s[16], mm[3] = "CV";
        scanf("%d %d", &C, &V);
        for (int i = C; i < C+V; i++)
            A[i] = 1;
        int f = 0;
        do {
            for (int i = 0; i < C+V; i++)
                s[i] = mm[A[i]];
            s[C+V] = '\0';
            if (f)  putchar(' ');
            printf("%s", s), f = 1;
        } while (next_permutation(A, A+C+V));
        puts("");
    }
    return 0; }
```

VeniceTechnique.cpp

Description: Venice Technique

74 lines

```
/*
We want a data structure capable of doing three main update-
     operations and some
sort of query. The three modify operations are: add: Add an
     element to the set.
remove: Remove an element from the set. updateAll: This one
     normally changes in
this case subtract X from ALL the elements. For this technique
     it is completely
required that the update is done to ALL the values in the set
     equally.
And also for this problem in particular we may need one query:
getMin: Give me the smallest number in the set.
*/
// Interface of the Data Structure
struct VeniceSet {
  void add(int);
  void remove(int);
  void updateAll(int);
  int getMin(); // custom for this problem
  int size();
};

/*
Imagine you have an empty land and the government can make
     queries of the following
type: * Make a building with A floors. * Remove a building with
      B floors. * Remove
C floors from all the buildings. (A lot of buildings can be
     vanished) * Which is the
smallest standing building. (Obviously buildings which are
     already banished don't count)
The operations 1,2 and 4 seems very easy with a set, but the 3
     is very cost effective
probably O(N) so you might need a lot of workers. But what if
     instead of removing C
floors we just fill the streets with enough water (as in venice
     ) to cover up the
first C floors of all the buildings :O. Well that seems like
     cheating but at least
those floor are now vanished :). So in order to do that we
     apart from the SET
we can maintain a global variable which is the water level. so
     in fact if we
have an element and want to know the number of floors it has we
      can just do
height - water_level and in fact after water level is for
     example 80, if we
want to make a building of 3 floors we must make it of 83
     floors so that it
can touch the land.
*/
struct VeniceSet {
  multiset<int> S;
  int water_level = 0;
  void add(int v) {
    S.insert(v + water_level);
  }
  void remove(int v) {
    S.erase(S.find(v + water_level));
  }
  void updateAll(int v) {
    water_level += v;
  }
  int getMin() {
    return *S.begin() - water_level;
  }
  int size() {
    return S.size();
  }
};
VeniceSet mySet;
for (int i = 0; i < N; ++i) {
  mySet.add(V[i]);
  mySet.updateAll(T[i]); // decrease all by T[i]
  int total = T[i] * mySet.size(); // we subtracted T[i] from
       all elements

  // in fact some elements were already less than T[i]. So we
       probbaly are counting
  // more than what we really subtracted. So we look for all
       those elements
  while (mySet.getMin() < 0) {
    // get the negative number which we really did not
         subtracted T[i]
    int toLow = mySet.getMin();

    // remove from total the amount we over counted
    total -= abs(toLow);

    // remove it from the set since I will never be able to
         substract from it again
    mySet.remove(toLow);
  }
  cout << total << endl;
}
cout << endl;
```

StableMarriageProblem.cpp

Description: Stable Marriage Problem
Time: $\mathcal{O}\left(n^2\right)$

49 lines

```
// Gale-Shapley algorithm for the stable marriage problem.
// madj[i][j] is the jth highest ranked woman for man i.
// fpref[i][j] is the rank woman i assigns to man j.
// Returns a pair of vectors (mpart, fpart), where mpart[i]
     gives
// the partner of man i, and fpart is analogous
/*
The Stable Marriage Problem states that given N men and N women
     , where each person has ranked all members of the opposite
      sex in order of preference,
marry the men and women together such that there are no two
     people of opposite sex who would both rather have each
     other than their current partners.
If there are no such people, all the marriages are "stable".

Consider the following example.

Let there be two men m1 and m2 and two women w1 and w2.
Let m1's list of preferences be {w1, w2}
Let m2's list of preferences be {w1, w2}
Let w1's list of preferences be {m1, m2}
Let w2's list of preferences be {m1, m2}

The matching { {m1, w2}, {w1, m2} } is not stable because m1
     and w1 would prefer each other over their assigned
     partners.
The matching {m1, w1} and {m2, w2} is stable because there are
     no two people of opposite sex that would prefer each other
      over their assigned partners.

It is always possible to form stable marriages from lists of
     preferences
The idea is to iterate through all free men while there is any
     free man available. Every free man goes to all women in
     his preference list according
to the order. For every woman he goes to, he checks if the
     woman is free, if yes, they both become engaged. If the
     woman is not free, then the woman
chooses either says no to him or dumps her current engagement
     according to her preference list. So an engagement done
     once can be broken if a woman
gets better option.
*/
pair<vector<int>, vector<int> > stable_marriage(vector<vector<
     int> >& madj, vector<vector<int> >& fpref) {
  int n = madj.size();
  vector<int> mpart(n, -1), fpart(n, -1);
  vector<int> midx(n);
  queue<int> mfree;
  for (int i = 0; i < n; i++) {
    mfree.push(i);
  }
  while (!mfree.empty()) {
    int m = mfree.front(); mfree.pop();
    int f = madj[m][midx[m]++];
    if (fpart[f] == -1) {
      mpart[m] = f; fpart[f] = m;
    } else if (fpref[f][m] < fpref[f][fpart[f]]) {
      mpart[fpart[f]] = -1; mfree.push(fpart[f]);
      mpart[m] = f; fpart[f] = m;
    } else {
      mfree.push(m);
    }
  }
  return make_pair(mpart, fpart);
}
```

LinearDionphanatic.cpp

Description: Linear Dionphanatic

64 lines

```
int extended_euclid(int a, int b, int &x, int &y) {
    int xx = y = 0;
    int yy = x = 1;
    while (b) {
        int q = a / b;
        int t = b; b = a%b; a = t;
        t = xx; xx = x - q*xx; x = t;
        t = yy; yy = y - q*yy; y = t;
    }
    return a;
}
// Linear Diophantine Equation Solution: Given, a*x+b*y=c. Find
     valid x and y if possible.
bool linear_diophantine (int a, int b, int c, int & x0, int &
     y0, int & g) {
    g = extended_euclid (abs(a), abs(b), x0, y0);
    if (c % g != 0)
        return false;
    x0 *= c / g;
    y0 *= c / g;
    if (a < 0)   x0 *= -1;
    if (b < 0)   y0 *= -1;
    return true;
}
// for each integer k, // x1 = x + k * b/g // y1 = y - k * a/g
// is a solution to the equation where g = gcd(a,b).
void shift_solution (int & x, int & y, int a, int b, int cnt) {
    x += cnt * b;
    y -= cnt * a;
}
// Now How many solution where x in range[x1,x2] and y in range
     [y1,y2] ?
int find_all_solutions(int a,int b,int c,int &minx,int &maxx,
     int &miny,int &maxy)
{
    int x,y,g;
    if(linear_diophantine(a,b,c,x,y,g) == 0) return 0;
    a/=g, b/=g;
    int sign_a = a>0 ? +1 : -1;
    int sign_b = b>0 ? +1 : -1;

    shift_solution (x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution (x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    int lx1 = x;
    shift_solution (x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution (x, y, a, b, -sign_b);
    int rx1 = x;
    shift_solution (x, y, a, b, - (miny - y) / a);
    if (y < miny)
        shift_solution (x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    int lx2 = x;
    shift_solution (x, y, a, b, - (maxy - y) / a);
    if (y > maxy)
        shift_solution (x, y, a, b, sign_a);
    int rx2 = x;

    if (lx2 > rx2)
        swap (lx2, rx2);
    int lx = max (lx1, lx2);
    int rx = min (rx1, rx2);

    return (rx - lx) / abs(b) + 1;
}
```

Simplex.h

Description: Solves a general linear maximization problem: maximize $c^T x$ subject to $Ax \le b$, $x \ge 0$. Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of $c^T x$ otherwise. The input vector is set to an optimal x (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that $x = 0$ is viable.
Usage: `vvd A = {{1,-1}, {-1,1}, {-1,-2}};`
`vd b = {1,1,-4}, c = {-1,-1}, x;`
`T val = LPSolver(A, b, c).solve(x);`
Time: $\mathcal{O}(NM * \#pivots)$, where a pivot may be e.g. an edge relaxation. $\mathcal{O}(2^n)$ in the general case.

68 lines

```
typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;

const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j

struct LPSolver {
  int m, n;
  vi N, B;
  vvd D;

  LPSolver(const vvd& A, const vd& b, const vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
      FOR(i,0,m) FOR(j,0,n) D[i][j] = A[i][j];
      FOR(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i];}
      FOR(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
      N[n] = -1; D[m+1][n] = 1;
    }

  void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    FOR(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
      T *b = D[i].data(), inv2 = b[s] * inv;
      FOR(j,0,n+2) b[j] -= a[j] * inv2;
      b[s] = a[s] * inv2;
    }
    FOR(j,0,n+2) if (j != s) D[r][j] *= inv;
    FOR(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }

  bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
      int s = -1;
      FOR(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
      if (D[x][s] >= -eps) return true;
      int r = -1;
      FOR(i,0,m) {
        if (D[i][s] <= eps) continue;
        if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                     < MP(D[r][n+1] / D[r][s], B[r])) r = i;
      }
      if (r == -1) return false;
      pivot(r, s);
    }
  }

  T solve(vd &x) {
    int r = 0;
    FOR(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
      pivot(r, n);
      if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
      FOR(i,0,m) if (B[i] == -1) {
        int s = 0;
        FOR(j,1,n+1) ltj(D[i]);
        pivot(i, s);
      }
    }
    bool ok = simplex(1); x = vd(n);
    FOR(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
  }
};
```

Tridiagonal.h

Description: $x = \text{tridiagonal}(d, p, q, b)$ solves the equation system

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} d_0 & p_0 & 0 & 0 & \cdots & 0 \\ q_0 & d_1 & p_1 & 0 & \cdots & 0 \\ 0 & q_1 & d_2 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & q_{n-3} & d_{n-2} & p_{n-2} \\ 0 & 0 & \cdots & 0 & q_{n-2} & d_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix}.$$

This is useful for solving problems on the type

$$a_i = b_i a_{i-1} + c_i a_{i+1} + d_i,\ 1 \le i \le n,$$

where a_0, a_{n+1}, b_i, c_i and d_i are known. a can then be obtained from

$$\{a_i\} = \text{tridiagonal}(\{1, -1, -1, ..., -1, 1\}, \{0, c_1, c_2, \dots, c_n\}, \{b_1, b_2, \dots, b_n, 0\}, \{a_0, d_1, d_2, \dots, d_n, a_{n+1}\}).$$

Fails if the solution is not unique.
If $|d_i| > |p_i| + |q_{i-1}|$ for all i, or $|d_i| > |p_{i-1}| + |q_i|$, or the matrix is positive definite, the algorithm is numerically stable and neither `tr` nor the check for `diag[i] == 0` is needed.
Time: $\mathcal{O}(N)$ 26 lines

```
typedef double T;
vector<T> tridiagonal(vector<T> diag, const vector<T>& super,
    const vector<T>& sub, vector<T> b) {
  int n = sz(b); vi tr(n);
  FOR(i,0,n-1) {
    if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i] == 0
      b[i+1] -= b[i] * diag[i+1] / super[i];
      if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];
      diag[i+1] = sub[i]; tr[++i] = 1;
    } else {
      diag[i+1] -= super[i]*sub[i]/diag[i];
      b[i+1] -= b[i]*sub[i]/diag[i];
    }
  }
  for (int i = n; i--;) {
    if (tr[i]) {
      swap(b[i], b[i-1]);
      diag[i-1] = diag[i];
      b[i] /= super[i-1];
    } else {
      b[i] /= diag[i];
      if (i) b[i-1] -= b[i]*super[i-1];
    }
  }
  return b;
}
```

10.2 Dynamic programming

DPKnapsack01.java

Description: 0/1 Knapsack Problem - Given items of certain weights/values and maximum allowed weight how to pick items to pick items from this set to maximize sum of value of items such that sum of weights is less than or equal to maximum allowed weight.
Time: $\mathcal{O}(W * totalitems)$ 19 lines

```
public class Knapsack01 {
    public int bottomUpDP(int val[], int wt[], int W){
        int K[][] = new int[val.length+1][W+1];
        for(int i=0; i <= val.length; i++){
            for(int j=0; j <= W; j++){
                if(i == 0 || j == 0){
                    K[i][j] = 0;
                    continue;
                }
                if(j - wt[i-1] >= 0){
                    K[i][j] = Math.max(K[i-1][j], K[i-1][j-wt[i
                        -1]] + val[i-1]);
                }else{
                    K[i][j] = K[i-1][j];
                }
            }
        }
        return K[val.length][W];
    }
}
```

DigitCountDP.cpp

Description: Digit Cound DP 38 lines

```
// Calculate how many numbers in the range from A to B that
     have digit d in only the even positions and
// no digit occurs in the even position and the number is
     divisible by m.

string A, B; int m, d;
ll dp[2002][2002][2][2];

ll calc(int idx, int Mod, bool s, bool b)
{
  if(idx==B.size()) return Mod==0;

  if(dp[idx][Mod][s][b]!=-1)
    return dp[idx][Mod][s][b];

  ll ret=0;

  int low=s ? 0 : A[idx]-'0';
  int high=b ? 9 : B[idx]-'0';

  for(int i=low; i<=high; i++)
  {
    if(idx%2 && i!=d) continue;
    if(idx%2==0 && i==d) continue;

    ret=(ret+calc(idx+1, (Mod*10+i)%m, s || i>low, b || i<high)
         )%mod;

    // if(ret>=mod) ret-=mod;
  }

  return dp[idx][Mod][s][b]=ret;
}

int main()
{
  cin>>m>>d>>A>>B;
  ms(dp,-1);
  prnt(calc(0,0,0,0));
  return 0;
}
```

EditDistance.cpp

Description: Edit Distance Top Down 22 lines

```
int dp[34][34];
string a, b;

int editDistance(int i, int j)
{
  if (dp[i][j]!=-1) return dp[i][j];
  if (i==0) return dp[i][j]=j;
  if (j==0) return dp[i][j]=i;

  int cost;
  if (a[i-1]==b[j-1]) cost=0;
  else cost=1;
  return dp[i][j]=min(editDistance(i-1,j)+1,min(editDistance(i,
       j-1)+1,
          editDistance(i-1,j-1)+cost));
}
int main()
{
  ms(dp,-1);
  cin>>a>>b;
  prnt(editDistance(a.size(),b.size()));
    return 0;
}
```

LCS.cpp

Description: Longest Common Substring 38 lines

```
int lcs(string str1,string str2,int len1, int len2){
    if(len1 == (int)str1.length() || len2 == (int)str2.length()
         ) return 0;
    if(str1[len1] == str2[len2]) return 1 + lcs(str1,str2,len1
         +1,len2+1);
    else  return max(lcs(str1,str2,len1+1,len2),lcs(str1,str2,
         len1,len2+1));
}
int main()
{
    string input1 = "acbcf";
    string input2 = "abcdaf";
    cout<<"TOP DOWN: " <<lcs(input1,input2,0,0)<<'\n';

    int T[input1.length()+1][input2.length()+1];
    for(unsigned int i=0; i<=input1.length();i++) T[i][0]=0;
    for(unsigned int i=0; i<=input2.length();i++) T[0][i]=0;

    for(unsigned int i=1; i <= input1.length(); i++)
    {
       for(unsigned int j=1; j <= input2.length(); j++)
       {
           if(input1[i-1]==input2[j-1]) T[i][j]=T[i-1][j-1]+1;
           else T[i][j]=max(T[i-1][j],T[i][j-1]);
           cout<<T[i][j]<<' ';
       }
       cout<<'\n';
    }
    cout<<T[input1.length()][input2.length()]<<'\n';

    cout<<"GET STRING\n";
    string sol;
    int i = input1.length();
    int j = input2.length();
    while(i!=0 && j!=0)
    {
           if(max(T[i-1][j],T[i][j-1])!=T[i][j]) { sol=input2[j
                -1]+sol; i--; j--;}
           else if(T[i][j-1]==0) i--; else j--;
    }
    cout<<sol<<'\n';
}
```

MatrixChainMultiplication.cpp

Description: Matrix Chain Multiplication Top Down 42 lines

```
#define n 4

int arr[] = {1, 2, 3, 4};
int dp[n][n];

int topDown(int i, int j)
{
    if(i==j) return 0;
    if(dp[i][j]!=-1) return dp[i][j];
    dp[i][j]=INF;
    for(int k=i; k<=j;k++)
    {
        int temp = topDown(i,k)+topDown(k+1,j)+arr[i-1]*arr[k]*
             arr[j];
        dp[i][j]=min(dp[i][j],temp);
    }
    return dp[i][j];
}

int main()
{
    memset(dp,-1,sizeof(dp));
    int m[n][n];

    for (int i=1; i<n; i++) m[i][i] = 0; // cost is zero when
         multiplying one matrix.

    for (int L=2; L<n; L++) // L is chain length.
        for (int i=1; i<n-L+1; i++)
        {
            int j = i+L-1;
            m[i][j] = INT_MAX;
            for (int k=i; k<=j-1; k++)
            {
                // q = cost/scalar multiplications
                int q = m[i][k] + m[k+1][j] + arr[i-1]*arr[k]*
                     arr[j];
                if (q < m[i][j])  m[i][j] = q;
            }
        }

    cout<<m[1][n-1]<<endl;
    cout<<topDown(1,n-1)<<endl;
    return 0;
}
```

PalindromeString.cpp

Description: Tower of Hanoi 19 lines

```
bool isPalindrome[100][100];

int main()
{
  string s; cin>>s;
  int len=s.size();
  for(int i=0; i<len; i++) isPalindrome[i][i]=true;
  for(int k=1; k<len; k++)
  {
    for(int i=0; i+k<len; i++)
    {
      int j=i+k;

      isPalindrome[i][j]=(s[i]==s[j]) &&
      (isPalindrome[i+1][j-1] || i+1>=j-1);
    }
  }
  return 0;
}
```

LongestIncreasingSubSeq.java

Description: Given an array, find longest increasing subsequence.
Time: n*log(n) 53 lines

```
public class LongestIncreasingSubSequenceOlogNMethod {
    private int ceilIndex(int input[], int T[], int end, int s)
         {
        int start = 0; //Returns index in T for ceiling of s
        int middle;
        int len = end;
        while(start <= end){
            middle = (start + end)/2;
            if(middle < len && input[T[middle]] < s && s <=
                 input[T[middle+1]]){
                return middle+1;
            }else if(input[T[middle]] < s){
                start = middle+1;
            }else{
                end = middle-1;
            }
        }
        return -1;
    }
    public int longestIncreasingSubSequence(int input[]){
        int T[] = new int[input.length];
        int R[] = new int[input.length];
        for(int i=0; i < R.length ; i++) {
            R[i] = -1;
        }
        T[0] = 0;
        int len = 0;
        for(int i=1; i < input.length; i++){
            if(input[T[0]] > input[i]){
        //if input[i] is less than 0th value of T then replace
             it there.
                T[0] = i;
            }else if(input[T[len]] < input[i]){
        //if input[i] is greater than last value of T then
             append it in T
                len++;
                T[len] = i;
                R[T[len]] = T[len-1];
            }else{
        //do a binary search to find ceiling of input[i] and
             put it there.
                int index = ceilIndex(input, T, len,input[i]);
                T[index] = i;
                R[T[index]] = T[index-1];
            }
        }
        //this prints increasing subsequence in reverse order.
        System.out.print("Longest increasing subsequence ");
        int index = T[len];
        while(index != -1) {
            System.out.print(input[index] + " ");
            index = R[index];
        }

        System.out.println();
        return len+1;
    }
}
```

PartitionsInteger.cpp

Description: Partitions Integer 25 lines

```
//4 = 3 + 1 , 2 + 2 , 2 + 1 + 1 ,1 + 1 +1 + 1
void printAllUniqueParts(int n) {
    int p[n]; int k = 0;  p[k] = n;
    while (true) {
        for(int i=0;i<k+1;i++)
        cout<<p[i]<<" ";
        cout<<endl;
        int rem_val = 0;
        while (k >= 0 && p[k] == 1) {
            rem_val += p[k];
            k--; }
        if (k < 0)  return;
        p[k]--;
        rem_val++;
        while (rem_val > p[k]) {
            p[k+1] = p[k];
            rem_val = rem_val - p[k];
            k++;  }
        p[k+1] = rem_val;
   k++; } }
int main()
{
   printAllUniqueParts(4);
   return 0;
}
```

LongestIncrCommonSubSeq.java

Description: Given an array, find longest increasing subsequence.
Usage: { 2 3 1 6 5 4 6 } AND { 1 3 5 6 } the LCIS is { 3 5 6} .
Time: n*m 44 lines

```
static int LCIS(int arr1[], int n, int arr2[],int m) {
  int table[] = new int[m];
  int parent[] = new int[m];
  // Traverse all elements of arr1[]
  for (int i = 0; i < n; i++) {
    // Initialize current length of LCIS
    int current = 0, last = -1;
    // For each element of arr1[] trvarse all elements of arr2
        [].
    for (int j = 0; j < m; j++) {
      // If both the array have same elements.
      // Note that we don't break the loop here.
      if (arr1[i] == arr2[j]){
        if (current + 1 > table[j]){
          table[j] = current + 1;
          parent[j] = last;
        }
      }
      //previous smaller common for current
      if (arr1[i] > arr2[j]){
        if (table[j] > current){
          current = table[j];
          last = j;
        }
      }
    }
  }
  // max. value in table[] is our res
  int result = 0, index = -1;
  for (int i = 0; i < m; i++){
    if (table[i] > result){
    result = table[i];
    index = i;
    }
  }
  // LCIS is going to store elements of LCIS
  int lcis[] = new int[result];
  for (int i = 0; index != -1; i++){
    lcis[i] = arr2[index];
    index = parent[index];
  }
  for (int i = result - 1; i >= 0; i--)
    System.out.print(lcis[i] + " "); //Printing LCIS
  return result;
}
```

10.3 Game Theory

Conditions that has not apper to aply Grundy No random moves such as the rolling of dice or the dealing of cards are allowed. This rules out games like backgammon and poker. A combinatorial game is a game of perfect information: simultaneous moves and hidden moves are not allowed. This rules out battleship and scissors-paper-rock. No draws in a finite number of moves are possible. This rules out tic-tac-toe. In these notes, we restrict attention to impartial games, generally under the normal play rule.

Nim sum: The cumulative XOR value of the number of coins/stones in each pile/heaps at any point of the game is called Nim-Sum at that point. "If both Alice and Bob play optimally (i.e.- they don't make any mistakes), then the player starting first is guaranteed to win if the Nim-Sum at the beginning of the game is non-zero. Otherwise, if the Nim-Sum evaluates to zero, then player Alice will lose definitely." Sprague-Grundy Theorem says that if both Alice and Bob play optimally (i.e., they don't make any mistakes), then the player starting first is guaranteed to win if the XOR of the Grundy numbers of position in each sub-games at the beginning of the game is non-zero. Otherwise, if the XOR evaluates to zero, then player A will lose definitely, no matter what.

Steps (Must be impartial game) Break the composite game into sub-games. Then for each sub-game, calculate the Grundy Number at that position. Then calculate the XOR of all the calculated Grundy Numbers. If the XOR value is non-zero, then the player who is going to make the turn (First Player) will win else he is destined to lose, no matter what.

Game Theory Tree Graph 1. Colon Principle: Grundy number of a tree is the xor of Grundy number of child subtrees. (Proof: easy).

2. Fusion Principle: Consider a pair of adjacent vertices u, v that has another path (i.e., they are in a cycle). Then, we can contract u and v without changing Grundy number. (Proof: difficult)

GameTheoryXOR.java

Description: 0/1 Knapsack Problem - Given items of certain weights/values and maximum allowed weight how to pick items to pick items from this set to maximize sum of value of items such that sum of weights is less than or equal to maximum allowed weight.
Time: $\mathcal{O}(W * totalitems)$ 8 lines

```
private long getXorFrom1ToX(long x) {
    if (x == 0 || x == 1) return x;
    if (x == 2) return 3;
    if (x == 3) return 0;
    long l = Long.highestOneBit(x);
    if (x % 2 == 1) return getGr(x - 1);
    return l + getGr(x - 1);
  }
```

hanoi.cpp

Description: Tower of Hanoi 27 lines

```
void hanoi(int n)
{
    moveTower(n, 'A', 'B', 'C');
}
void moveTower(int ht, char f, char t, char i)
{
    if (ht > 0)
    {
        moveTower(ht - 1, f, i, t);
        moveRing(ht, f, t);
        moveTower(ht - 1, i, t, f);
    }
}

void moveRing(int d, char f, char t)
{
    cout << "Move ring " << d << " from ";
    cout << f << " to " << t << endl;
}
int main()
{
    cout << "How many disks? ";
    int x; cin >> x;
    hanoi(x);
    return 0;
}
```

JosephusProblem.cpp

Description: Josephus problem 29 lines

```
int josephus(int n, int k, int m){
    int i;
    for (m = n - m, i = m + 1; i <= n; i++){
        m += k;
        if (m >= i) m %= i;
    }
    return m + 1;
}

long long josephus2(long long n, long long k, long long m){
    m = n - m;
    if (k <= 1) return n - m;

    long long i = m;
    while (i < n){
        long long r = (i - m + k - 2) / (k - 1);
        if ((i + r) > n) r = n - i;
        else if (!r) r = 1;
        i += r;
        m = (m + (r * k)) % i;
    }
    return m + 1;
}

int main(){
    int n, k, m;
    printf("%d\n", josephus(10, 1, 2));
    printf("%d\n", josephus(10, 1, 10));
}
```

BellManFord.cpp

Description: BellManFord gives the shortest path in a weighted directed graph (in which the weight of some of the edges can be negative). 29 lines

```
// Is there a negative cycle in the graph?
bool bellman(int src)
{
    // Nodes are indexed from 1
  for (int i = 1; i <= n; i++)
    dist[i] = INF;
  dist[src] = 0;
    for(int i = 2; i <= n; i++)
    {
        for (int j = 0; j < edges.size(); j++)
        {
            int u = edges[j].first;
            int v = edges[j].second;
            ll weight = adj[u][v];
            if (dist[u]!=INF && dist[u] + weight < dist[v])
                dist[v] = dist[u] + weight;
        }
    }
  for (int i = 0; i < edges.size(); i++)
    {
    int u = edges[i].first;
    int v = edges[i].second;
    ll weight = adj[u][v];
        // True if neg-cylce exists
    if (dist[u]!=INF && dist[u] + weight < dist[v])
      return true;
    }
  return false;
}
```

BinarySearch.cpp

Description: Binary Search 9 lines

```
int binary_search(int *array, int searched, int
arraySize) { int first = 0, middle, last = arraySize -
1;
while (first<=last) { middle = (first + last) / 2;
if (searched == array[middle]) return array[middle];
else {
if (array[middle] > searched) last = middle - 1;
else first = middle + 1;
} } return -1; }
```

CountingInversionswithBIT.cpp

Description: Counting Inversions with BIT 57 lines

```
ll tree[200005];
int n, a[200005], b[200005];

void update(int idx, ll x)
{
  while(idx<=n)
  {
    tree[idx]+=x;
    idx+=(idx&-idx);
  }
}

int query(int idx)
{
  ll sum=0;
  while(idx>0)
  {
    sum+=tree[idx];
    idx-=(idx&-idx);
  }
  return sum;
}

int main()
{
  int test;
  scanf("%d", &test);
  while(test--)
  {
    ms(tree,0);
    scanf("%d", &n);
    FOR(i,1,n+1)
    {
      scanf("%d", &a[i]);
      b[i]=a[i];
    }

    sort(b+1,b+n+1);

    // Compressing the array
    FOR(i,1,n+1)
    {
      int rank=int(lower_bound(b+1,b+1+n,a[i])-b-1);
      a[i]=rank+1;
    }
    // FOR(i,1,n+1) cout<<a[i]<<" "; cout<<endl;
    ll ans=0;
    FORr(i,n,1)
    {
      ans+=query(a[i]-1);
      update(a[i],1);
    }
    // prnt(ans);
    printf("%lld\n",ans);
  }
  return 0;
}
```

10.4 Area of polygon with circumradio

Area: $(r * r * n * sin(360/n))/2$

10.5 Unit vector

Given a vector $u = (u_1, u_2)$, the unit vector is $(u_1/|u|, u_2/|u|)$ where —u— it is the length of u

10.6 Optimization tricks

10.6.1 Bit hacks

- `x & -x` is the least bit in `x`.
- `x && !(x & (x - 1))` true, if `x` is power of 2.
- `gray_code[x] = x ^ (x >> 1)`
- `checkerboard[y][x] = (x & 1) ^ (y & 1)`
- `ffs(int x), ffs(ll x)` number of the least significant bit, `ffs(1 << i) = i+1`
- `for (int x = m; x; ) { --x &= m; ... }` loops over all subset masks of `m` (except `m` itself).
- `c = x&-x, r = x+c; (((r^x) >> 2)/c) | r` is the next number after `x` with the same number of bits set.
- `FOR(b,0,K) FOR(i,0,(1 << K)) if (i & 1 << b) D[i] += D[i^(1 << b)];` computes all sums of subsets.

Complexity

Complejidad	Rangos
100000M	$O(1)$
1000M	$O(log(n))$
1M	$O(n)$
100K	$O(log(n))$
1000	$O(n^2)$
100	$O(n^3)$
50	$O(n^4)$
20	$O(2^n)$
13	$O(n!)$

10.7 Techniques

techniques.txt 44 lines

```
Recursion
Greedy algorithm
  Scheduling
  Max contiguous subvector sum
Dynamic programming
  Knapsack
  Coin change
  Longest common subsequence
  Longest increasing subsequence
  Number of paths in a dag
  Shortest path in a dag
Combinatorics
  Computation of binomial coefficients
  Catalan number
  Pick's theorem
Number theory
  Integer parts
  Divisibility
  Euclidean algorithm
  Modular arithmetic
  * Modular multiplication
  * Modular inverses
  * Modular exponentiation by squaring
  Miller-Rabin
Game theory
  Game trees
  Mini-max
  Nim
  Grundy numbers
Optimization
  Binary search
  Ternary search
Numerical methods
  Root-finding with binary/ternary search
Geometry
  Coordinates and vectors
  * Cross product
  * Scalar product
  Convex hull
  Polygon cut
  Closest pair
  Coordinate-compression
  Quadtrees
  All segment-segment intersection
```