CONTENTS

Graph related

BFS

```
//Normal BFS
int getChild(int state, int index, bool& is_child_ok);
//implement me

//Goal is destination, N_state is maximum number of states, N_CH
is number of children
const int GOAL = -1, NS = 1, MX_DEPTH = INT_MAX, N_STATE =
362881, N_CH = 8;
int depths[N_STATE]; //not needed if we only have a single
destination
int st[NS] = { 0 }; //fill with start states
bool vis[N_STATE];
int depth;
inline int bfs() {
	int size;
	memset(vis, 0, sizeof vis);
	depth = 1; //1 based (source has a depth of one)
	queue<int> q;
	for (int i = 0; i < NS; ++i) {
		q.push(st[i]);
		vis[st[i]] = 1;
		depths[st[i]] = depth;
		if (st[i] == GOAL)
			return depth;
	}
	++depth;
	while (q.size() && depth <= MX_DEPTH) {
		size = q.size();
		while (size--) {
			int s = q.front();
			q.pop();
			for (int i = 0; i < N_CH; i++) {
				bool ok = true;
				int ns = getChild(s, i, ok);
				if (!ok || vis[ns])
					continue;
				depths[ns] = depth;
				vis[ns] = 1;
				q.push(ns);
				if (ns == GOAL)
					return depth;
			}
		}
		++depth;
	}
	return -1;
}
```

BFS with path printing

```
//Goal = destination, NS = num of start states, N_STATE = num of
states, N_CH = num of children
const int GOAL = 0, NS = 2, MX_DEPTH = 200, N_STATE = 10001, N_CH
= 2;
struct state {
	int s, p;//p is parent, s is state_id
};
vector<state> q;
state st[NS] = {/*start states, make sure to set parent to -1*/};
bool vis[N_STATE];
int depth;
int bfs() {//returns index of goal state in the global q
	int size;
	memset(vis, 0, sizeof vis);
	depth = 1; //1 based
	q.clear();
	for (int i = 0; i < NS; ++i) {
		q.push_back(st[i]);
		vis[st[i].s] = 1;
		if (st[i].s == GOAL)
			return i;
	}
	++depth;
	for (int cur = 0; cur < (int) q.size() && depth <=
MX_DEPTH;) {
		size = q.size();
		for (; cur < size; ++cur) {
			state s = q[cur];
			for (int i = 0; i < N_CH; i++) {
				state ns = /*generate child i*/;
				if (vis[ns.s])
					continue;
				vis[ns.s] = 1;
				q.push_back(ns);
				if (ns.s == GOAL)
					return q.size() - 1;
			}
		}
		++depth;
	}
	return -1;
}

//take care of stack overflow
void print(int ind) {
	if (q[ind].p != -1)
		print(q[ind].p);
	printf("%d", q[ind].s);//or pushback
}
```

BFS 01

```
//write isGoal, getChild and call bfs01()
inline bool isGoal(int state)
//returns the child with given index and sets is_child_ok to
false if that child is invalid
inline int getChild(int state, int child_index, bool& is_one,
bool& is_child_ok)

const int NS = 4, MX_DEPTH = INT_MAX, N_STATE = 100, N_CH = 3;
int st[NS];
bool vis[N_STATE];
int depth;
int depths[N_STATE];
inline int bfs01() {
	int size;
	memset(vis, 0, sizeof vis);
	depth = 1;
	queue<int> q[2];
	int curQ = 0;
	for (int i = 0; i < NS; ++i) {
		q[curQ].push(st[i]);
	}
	bool one, ok;
	while (q[curQ].size() && depth <= MX_DEPTH) {
		size = q[curQ].size();
		while (size--) {
			int s = q[curQ].front();
			q[curQ].pop();
			vis[s] = 1;
			depths[s] = depth;
			if (isGoal(s))
				return depth;
			for (int i = 0; i < N_CH; i++) {
				int ns = getChild(s, i, one, ok);
				if (!ok || vis[ns])
					continue;
				q[curQ ^ one].push(ns);
				size += !one;
			}
		}
		++depth;
		curQ = !curQ;
	}
	return -1;
}
```

DFS

```
//(i,j) encodes the state
bool vis[2600][2600];
const int N_CH = 4; //propagate changes to dir arrays
```

```
int R, C;
int di[] = { 0, 0, 1, -1 };
int dj[] = { 1, -1, 0, 0 };

bool valid(int ni, int nj) {
      return ni >= 0 && ni < R && nj >= 0 && nj < C &&
!vis[ni][nj];
}
void DFS(int i, int j) {
      vis[i][j] = 1;
      //process the state (i,j)
      for (int k = 0; k < N_CH; ++k) {
            int ni = i + di[k];
            int nj = j + dj[k];
            if (valid(ni, nj))
                  DFS(ni, nj);
      }
}
```

Dijkstra (n^2)

```
const int nSize = 100; // number of nodes;
int cost[nSize][nSize];
int d[nSize];
bool V[nSize];
int inf = 1<<25;

void dijkstra(int start,int goal,int n)
{

    for(int i = 0 ; i<n ; i++)
                d[i] = inf; // take care of over flow when sum
    memset(V,0,sizeof V);
    d[start] = 0;
    int current = start;
    int minD = inf;
    int nextInd = -1; // minmum to choose
    while(current!=goal&&current!= -1)
    {
        V[current] = 1;
        minD = inf;
        nextInd = -1;
        for(int i = 0 ; i < n ; ++ i )
        {
                if(!V[i] && d[current]+cost[current][i]<d[i])
                {
                        d[i] = d[current] + cost[current][i];
                        // law 3awez ageb el path we ana b relax
bkhzn el parent
                        // parent[i] = current;
                }
```

```
                if(!V[i]&&d[i]<minD){nextInd = i; minD = d[i];}
/* b3ml mimization bdal el for loop el zyada (sya3a y3ne)*/


        }
        current = nextInd;
    }
}
```

Dijkstra with Heap

```
//Fill the adj-list and call dijkstra
#define MAX 200001
#define infinity (long long)1e18
vector<vector<pair<int, long long> > > adj;
long long dist[MAX];
long long dijkstra(int s, int e) {
      fill(dist, dist + adj.size(), infinity);
      dist[s] = 0;
      priority_queue<pair<long long, int> > q;
      q.push(make_pair(0, s));
      while (q.size()) {
            int cur = q.top().second;
            long long d = -q.top().first;
            q.pop();
            if (d != dist[cur])
                  continue;
            if (cur == e)
                  return d;
            for (int i = 0; i < adj[cur].size(); ++i) {
                  int j = adj[cur][i].first;
                  long long dd = d + adj[cur][i].second;
                  if (dist[j] > dd) {
                        dist[j] = dd;
                        q.push(make_pair(-dd, j));
                  }
            }
      }
      return infinity;
}
```

Floyed Warshal

```
void floyd() // set global var n to node count, dist[n][n] to
infinity and edge costs
{
    int i,j,k;
    for(k=0;k<n;k++)
        for(i=0;i<n;i++)
            for(j=0;j<n;j++)
                if(dist[i][k]+dist[k][j]<dist[i][j])
                    dist[i][j]=dist[i][k]+dist[k][j]; //set
dist[j][i] if undirected graph
```

```
}
```

Bellman Ford

```
/*
 /// Difference constraints ///
 //5 4 -> means x5 - x4
 //5+4 = end , start --;
 // law kanet gt swap start&end--we hdrab el right hand side fe -
1
 * gatle shwyet eniqualities 3la shkl
 * x-y<=c1
 * x-z<=c2
 xj-xi<=wij create edge from xj to xi by cost wij for each
equation
 if(>=){ swap(xj,xi); wij*=-1 }
 if(< || >) wij--
 create new node and make an edge from it to each node with cost
0
 run bellman from this new node
 if(returned false) there is no solution
 */
#define MAX_EDGES 6001
#define MAX_NODES 1001
#define infinity 1e14
int n, e;//node_count, edge_count
double dist[MAX_NODES];
int f[MAX_EDGES]; //from
int t[MAX_EDGES]; //to
double d[MAX_EDGES]; //edge cost

inline int dcmp(const double& a, const double& b) {
      if (fabs(a - b) < 1e-9)
            return 0;
      return a < b ? -1 : 1;
}
inline bool bellman(int source) {
      fill(dist,dist+n, infinity);
      dist[source] = 0;
      for (int i = 0; i < n; ++i) {
            bool rex = false;
            for (int j = 0; j < e; ++j) {
                  double cst = dist[f[j]] + d[j]; //change to int
                  if (dcmp(cst, dist[t[j]]) < 0) {//change to cst <
dist[t[j]] if int distances
                        dist[t[j]] = cst;
                        rex = true;
                  }
            }
            if (rex && i == n - 1)
                  return false;
      }
```

```
	return true;
}
```

Prim

```
const int NSize = 100;
int cost[NSize][NSize];
int d[NSize];
bool V[NSize];
const int inf = 9999999;
// take care of multiple edges
void  mstPrim(int start,int n)
{
	// lazem undirected graph
	for(int i = 0 ; i<n ; i++)
			d[i] = inf; // take care of over flow when
sum
	 memset(V,0,sizeof V);
	    d[start] = 0;/
	int current = start;
	int minD = inf;
	int nextInd = -1; // minmum to choose
	while(current!= -1)
	{
	    V[current] = 1;
	    minD = inf;
	    nextInd = -1;
	    for(int i = 0 ; i < n ; ++ i )
	    {
			if(!V[i] && cost[current][i]<d[i])
			{
				d[i] =  cost[current][i];
				// law 3awez ageb el path we ana b
relax bkhzn el parent
				// parent[i] = current;
			}
			if(!V[i]&&d[i]<minD){nextInd = i; minD =
d[i];} /* b3ml mimization bdal el for loop el zyada (sya3a
y3ne)*/


	    }
	    current = nextInd;
	}
}
```

Kruskal

```
struct edge
{
	int from,to;
	double cost;
	bool operator < (const edge &t) const
```

```
        {
            return cost < t.cost;
        }

};
const int SIZE = 100;
edge edges[SIZE*SIZE]; // here the size of Number of Nodes (SIZE
* SIZE) is edges from all nodes to all other nodes
pair< double , vector<edge> > mstKruskal(int n)
{
    double cost = 0;
    vector<edge> ed;
    sort(edges,edges+n);
    DisjointSet  dis(n);
    for(int i = 0 ; i < n ; i++)
    {
            if(dis.find(edges[i].from) == dis.find(edges[i].to))
continue;
            dis.join(edges[i].from,edges[i].to);
            cost += edges[i].cost;
            ed.push_back(edges[i]);
    }
    return make_pair(cost,ed);
}
```

Maximum Flow

```
//General notes on max-flow
   //    - Resize adj list to number of nodes
   //    - call addedge to construct graph
   //    - residue flow in adjacency list (Edge::cst)
   //    -Get edges in maxFlow  ( get min cut )
   //      -ff all egdes  if flow zero faks , else emshy
   //       -then in org adj lw edge el from vis wl to msh vis and
   flow +ve
   //        then cut edge
   //    -Get edges in maxFlow  ( get min vertex cover )
   //      -ff all egdes  if flow zero faks , else emshy
   //      -B set of vertex 3l ymen , A set of vertex 3l shmal
   //      -result is (vistited nodes in B) U (not visited nodes in
   A)
   //     - Max independent set = n - max_match
   //        - This nodes is all node that not in Min vertex cover.
   //     - Min path coverage "DAG"  = n- max_match
   //        -path is all edge with flow 1 and Go from B to A in
   result adj

   //Set the global vars using add edge and run max_flow
   struct edge{
        int to, flow, rev;
   };
```

```
vector<vector<edge> > adj;
#define MAX_NODES 200
#define infinity 1e9
bool vis[MAX_NODES];
int source, sink, node_count , edge_count;

void add_edge(int from,int to, int flow)
{
      edge e1={to,flow,adj[to].size()};
      edge e2={from,0,adj[from].size()};
      adj[from].push_back(e1);
      adj[to].push_back(e2);
}
int find_path(int cur, int flow)
{
      if(cur == sink)
            return flow;
      if(vis[cur] || !flow)
            return 0;
      vis[cur] = true;
      for(int i=0 ; i < adj[cur].size(); i++)
      {
            edge &e = adj[cur][i];
            int fl = find_path(e.to,min(flow,e.flow));
            if(fl){
                  edge &r = adj[e.to][e.rev];
                  e.flow -= fl;
                  r.flow += fl;
                  return fl;
            }
      }
      return 0;
}
int max_flow()
{
      memset(vis,0,sizeof(vis));
      int res = 0, fl;
      while((fl=find_path(source,infinity)))
      {
            res += fl;
            memset(vis,0,sizeof(vis));
      }
      return res;
}
```

Max Matching Recursively: O(left * right)

```
#define sz(v) ((int)v.size())
#define rep(i,m) for(int i=0;i<(int)(m);i++)

const int MX = 405;
vii adj;
```

```
int r[MX];
int l[MX];
bool v[MX];
int numR;
bool match(int i) {
	rep(j,sz(adj[i])) {
		if (v[adj[i][j]])
			continue;
		v[adj[i][j]] = 1;
		if (l[adj[i][j]] == -1 || match(l[adj[i][j]])) {
			l[adj[i][j]] = i;
			r[i] = adj[i][j];
			return true;
		}
	}
	return false;
}
int runMatching() {
	int cc = 0;
	mem(r, -1);
	mem(l, -1);
	rep(i,numR) {
		mem(v,0);
		if (match(i))
			cc++;
	}
	return cc;
}
```

Min Cost Max Flow

```
//Add edges with -ve cost if max cost is required
#define rep(i,X,n) for( int (i) = (X) ; (i)<(n) ; (i) ++)
inline int  gInd(int x){
	return (x)%2?(x)-1:(x)+1;
}
int n,m;
struct Edge
{
	int fr,to,cost,flow;
	Edge(){fr=to=cost=flow=-1;}
	Edge(int a,int b,int c,int d)
	:fr(a),to(b),cost(c),flow(d)
	{
	}
};
struct MinCostMaxFlow
{
	vector<Edge> es;
	int flow[105];
	int cost[105];
	int p[105];
	void AddEdge(int fr,int to,int co,int fl)
```

```
	{
		es.PB(Edge(fr,to,co,fl));
		es.PB(Edge(to,fr,-co,0));
	}
	MinCostMaxFlow()
	{
		clear();
	}
	int getPath(int src,int sink)
	{
		memset(flow,0,sizeof(flow));
		fill(cost,cost+n+m+2,100000000);
		cost[src]=0;
		flow[src]=100000000;
		memset(p,-1,sizeof(p));
		rep(i,0,n+m+2)
		{
			bool fl =0;
			rep(j,0,es.size())
			{
				if( cost[es[j].to] >
cost[es[j].fr]+es[j].cost && es[j].flow!=0)
				{
					cost[es[j].to] =
cost[es[j].fr]+es[j].cost;
					flow[es[j].to] =
min(flow[es[j].fr],es[j].flow);
					p[es[j].to] = j;
					fl=1;
				}
			}
			if(!fl)
				break;
		}
		return flow[sink];
	}
	pair<int,int> MCMF(int src,int sink)
	{
		if(src == sink)
			return make_pair(100000000,0);
		int fl,totc=0,totf=0,cur;
		while(1)
		{
			fl = getPath(src,sink);
			if(p[sink]==-1)
				break;
			cur = sink;
			while(p[cur]!=-1)
			{
				es[p[cur]].flow-=fl;
				es[gInd(p[cur])].flow+=fl;
				totc +=fl*es[p[cur]].cost;
```

```
                    cur = es[p[cur]].fr;
                }
                totf += fl;
            }
            return make_pair(totf,totc);
        }
        void clear()
        {
            es.clear();
            memset(flow,0,sizeof(flow));
            memset(cost,0,sizeof(cost));
            memset(p,-1,sizeof(p));
        }
};
```

Dinic max flow: (Better constant factor)

```
const int maxnode = 20000 + 5;
const int maxedge = 1000000 + 5;
const int oo = 1000000000;
int node, src, dest, nedge;
int head[maxnode], point[maxedge], next[maxedge], flow[maxedge],
capa[maxedge];
int dist[maxnode], Q[maxnode], work[maxnode];
void init(int _node, int _src, int _dest) {
      node = _node;
      src = _src;
      dest = _dest;
      for (int i = 0; i < node; i++)
            head[i] = -1;
      nedge = 0;
}
void addedge(int u, int v, int c1, int c2) {
      point[nedge] = v, capa[nedge] = c1, flow[nedge] = 0,
next[nedge] = head[u], head[u] =
                  (nedge++);
      point[nedge] = u, capa[nedge] = c2, flow[nedge] = 0,
next[nedge] = head[v], head[v] =
                  (nedge++);
}
bool dinic_bfs() {
      memset(dist, 255, sizeof(dist));
      dist[src] = 0;
      int sizeQ = 0;
      Q[sizeQ++] = src;
      for (int cl = 0; cl < sizeQ; cl++)
            for (int k = Q[cl], i = head[k]; i >= 0; i = next[i])
                  if (flow[i] < capa[i] && dist[point[i]] < 0) {
                        dist[point[i]] = dist[k] + 1;
                        Q[sizeQ++] = point[i];
                  }
      return dist[dest] >= 0;
```

```
    }

    int dinic_dfs(int x, int exp) {
          if (x == dest)
                return exp;
          for (int &i = work[x]; i >= 0; i = next[i]) {
                int v = point[i], tmp;
                if (flow[i] < capa[i] && dist[v] == dist[x] + 1
                            && (tmp = dinic_dfs(v, min(exp, capa[i] -
    flow[i]))) > 0) {
                      flow[i] += tmp;
                      flow[i ^ 1] -= tmp;
                      return tmp;
                }
          }
          return 0;
    }
    int dinic_flow() {
          int result = 0;
          while (dinic_bfs()) {
                for (int i = 0; i < node; i++)
                      work[i] = head[i];
                while (1) {
                      int delta = dinic_dfs(src, oo);
                      if (delta == 0)
                            break;
                      result += delta;
                }
          }
          return result;
}
```

Strongly Connected Components (recursive tarjan)

```
    vector<vector<int> > SCCs /* The components itself*/;
    #define comps SCCs
    vector<int> compIndex /* for each node, what is the index of the
    component this node inside*/
                ,ind, lowLink;
    stack<int> st;
    vector<bool> inst;
    vector<vector<int> > adj /*The intial graph*/;
    int idx = 0; //must be intialized by zero;

    void tarjanSCC(int i) {
          lowLink[i] = ind[i] = idx++;
          st.push(i);
          inst[i] = true;
          for (int j = 0; j < adj[i].size(); j++) {
                int k = adj[i][j];
                if (ind[k] == -1) {
                      tarjanSCC(k);
```

```
                lowLink[i] = min(lowLink[i], lowLink[k]);
            } else if (inst[k]) {
                lowLink[i] = min(lowLink[i], lowLink[k]);
            }
        }
        if (lowLink[i] == ind[i]) {
            vector<int> comp;
            int n = -1;
            while (n != i) {
                n = st.top();
                st.pop();
                comp.push_back(n);
                inst[n] = 0;
                compIndex[n] = comps.size();
            }
            comps.push_back(comp);
        }
}

void SCC() {
        comps.clear();
        compIndex.resize(adj.size());
        ind.clear();
        ind.resize(adj.size(), -1);
        lowLink.resize(adj.size());
        inst.resize(adj.size());
        idx = 0; //must be intialized by zero;
        for (int i = 0; i < adj.size(); i++)
            if (ind[i] == -1)
                tarjanSCC(i);
}

int cntSrc /*the number of source components*/,
cntSnk /*the number of sink copmonents*/;
vector<vector<int> > cmpAdj /*The new graph between components*/;
vector<int> inDeg, outDeg /*the in degree and out degree for each
component*/;

void computeNewGraph() {
        outDeg.clear();
        outDeg.resize(comps.size());
        inDeg.clear();
        inDeg.resize(comps.size());
        cntSrc = cntSnk = comps.size();
        cmpAdj.clear();
        cmpAdj.resize(comps.size());
        for (int i = 0; i < adj.size(); i++) {
            for (int j = 0; j < adj[i].size(); j++) {
                int k = adj[i][j];
                if (compIndex[k] != compIndex[i]) {

        cmpAdj[compIndex[i]].push_back(compIndex[k]);
```

```
                        if (!(inDeg[compIndex[k]]++))
                              cntSrc--;
                        if (!(outDeg[compIndex[i]]++))
                              cntSnk--;
                  }
            }
      }
}
```

Strongly connected components (iterative tarjan)

```
#define mem(s,v) memset(s,v,sizeof(s))
#define FOR(i,a,b) for(int i=(a);i<(b);i++)
#define sz(v) (int)v.size()

int ind[MAX];
int lo[MAX];
int cid[MAX];

stack<int> st;
int nxt, ncm;
vector<vi> adj;

struct state {
      int idx, i, j;
};

stack<state> stk;
inline void tarjan(int idx) {
      start: int i, j;
      ind[idx] = lo[idx] = nxt++;
      st.push(idx);
      for (i = 0; i < sz(adj[idx]); ++i) {
            j = adj[idx][i];
            if (ind[j] == -1) {
                  {
                        state s = { idx, i, j };
                        stk.push(s);
                        idx = j;
                        goto start;
                  }
                  //tarjan(j);
                  after: lo[idx] = min(lo[idx], lo[j]);
            } else if (cid[j] == -1)
                  lo[idx] = min(lo[idx], lo[j]);
      }
      if (ind[idx] == lo[idx]) {
            int x;
            do {
                  x = st.top();
                  st.pop();
                  cid[x] = ncm;
```

```
            } while (x != idx);
            ncm++;
        }
        if (sz(stk)) {
            i = stk.top().i;
            j = stk.top().j;
            idx = stk.top().idx;
            stk.pop();
            goto after;
        }
    }

    inline void SCC() {
        nxt = ncm = 0;
        mem (ind, -1);
        mem (cid, -1);
        FOR (i , 0 , sz(adj)) {
            if (ind[i] == -1)
                tarjan(i);
        }
    }
```

Kth Shortest Path

```
    //Nodes appear in more than one path
    struct edge{
        int s,e,c; //start, end, cost
        bool operator<(const edge& e)const{return c < e.c;}
    };
    const int SIZE = 100; //max nodes number
    int N, start, end, K; //find the k-th shortest path from start to
    end
    int dist[SIZE][SIZE]; //this can be adjList instead of adjMatrix
    //Returns -1 of no k-th shortest path exist between start and end
    int getKthShortestPath(){
        multiset<edge> pq;    //first is cost and second is node
        edge e = {-1, start, 0};
        pq.insert(e);
        vector<int> reached[N];    //reached[i][j] is the cost of
    the j-th shortest path from start to i
        while(!pq.empty()){
            edge e = *pq.begin(); pq.erase(pq.begin());
            if(reached[e.e].size() >= K)continue;
            reached[e.e].push_back(e.c);
            for(int i = 0 ; i < N ; i++){
                if(dist[e.e][i] == -1)continue;
                edge ne = {e.e, i, e.c+dist[e.e][i]};
                pq.insert(ne);
            }
        }
        //no k-th path exist between start and end
```

```
        return reached[end].size() >= K ? reached[end].back() : -1;
}
//MAIN
memset(dist, -1, sizeof dist);
//set N, start, end, K, dist
int d = getKthShortestPath();
```

Topological Sort

```
#define MAXN 101
vector<vector<int> > adjl;
int in[MAXN];
int n;  // number of nodes
vector<int> res;
bool Topological()
{
        queue<int> q;
        rep(i,0,n)
                if(in[i]==0)
                        q.push(i);
        res.clear();
        int x;
        while(!q.empty())
        {
                x = q.front();
                q.pop();
                res.PB(x);
                rep(i,0,adjl[x].size())
                        if(!--in[adjl[x][i]])
                                q.push(adjl[x][i]);
        }
        return res.size()==n;
}
```

Disjoint Set

```
const int MX = 100;
int parent[MX];
int rank[MX];
int compCnt;
void init(int n) {
        memset(rank, 0, sizeof(rank));
        for (int i = 0; i < n; i++)
                parent[i] = i;
        compCnt = n;
}
int find(int c) {
        if (parent[c] == c)
                return c;
        return parent[c] = find(parent[c]);
}
void merge(int c1, int c2) {
        int p1 = find(c1);
```

```
	int p2 = find(c2);
	if (p1 == p2)
		return;
	if (rank[p1] == rank[p2])
		rank[p1]++;
	else if (rank[p2] > rank[p1])
		swap(p1, p2);
	parent[p2] = p1;
	compCnt--;
}
```

Euler Tour

```
//Fence USACO
vector< list<int> > adj_list;
vector<int> res_nodes;
 vector<list<int>::iterator >  mat[501][501];

void  euler(int from)
{
	while(adj_list[from].size())
	{
		int to = adj_list[from].back();
		adj_list[from].pop_back();
		adj_list[to].erase( mat[to][from].back() );
		mat[from][to].pop_back();
		mat[to][from].pop_back();
		euler(to);
	}
	res_nodes.push_back(from);
}
void add_edge(int x,int y)
{
		adj_list[x].push_back(y);
}
int main()
{
	int N=0;
	int n,x,y;
	vector< pair<int,int> > v;
	cin>>n;
	while(n--)
	{
		cin>>x>>y;
		v.push_back( make_pair(x,y) );
		N = max( max(N,x) , y)+1;
	}
	adj_list.resize(N);
	vector<int> degree(N);
	int m = 1000000;
	for(n=0;n<(int)v.size();n++)
	{

		degree[ x = v[n].first  ]++;
		degree[ y = v[n].second ]++;
		add_edge(x,y);
```

```
                add_edge(y,x);
                m=min(m,x);
                m = min(m,y);
        }
        x = m;
        for(n=N-1;n>=0;n--)
        {
                if(degree[n]%2)
                        x = n;
                adj_list[n].sort();
                adj_list[n].reverse();
                list<int>::iterator  it;
                for(it=adj_list[n].begin();it!=adj_list[n].end();it++)
                        mat[n][*it].push_back(it);
        }
        euler(x);
        for(n=(int)res_nodes.size()-1;n>=0;n--)
                cout<<res_nodes[n]<<endl;
        return 0;
}
```

Stable Marriage Problem

```
// wr[w][m] --> the precedence of man number "m" with respect to
woman number "w", the less value the more important that man to
the woman
vector<vector<int> > wr;

// mp[m] --> has the a deque which contains the women in order of
importance to this man "m", the first woman is the most important
one to this man "m"
vector<deque<int> > mp;

// queue of mans, Initially contains all mans indi ces
queue<int> unMatchedMen;

// "wm" contains the result, such that wm[w] --> the man index
who is married to this woman "w"
// "mw" contains the result, such that mw[m] --> the women index
who is married to this man "m"
vector<int> wm, mw;

// this algorithm depends on the adaptive greedy approach
void stableMarrageProblem() {
      while (unMatchedMen.size()) {
            int mi = unMatchedMen.front();
            unMatchedMen.pop();
            while (1) {
                  int wi = mp[mi].front();
                  mp[mi].pop_front();
                  if (wm[wi] == -1) {
                        wm[wi] = mi;
                        mw[mi] = wi;
                        break;
```

```
                } else {
                        int mdi = wm[wi];
                        if (wr[wi][mi] < wr[wi][mdi]) {
                                wm[wi] = mi;
                                mw[mi] = wi;
                                unMatchedMen.push(mdi);
                                mw[mdi] = -1;
                                break;
                        }
                }
        }
    }
}
```

Bi – Connectivity

//Don't forget tor resize the vectors

```
vector<int> dfs_low, dfs_num,dfs_parent;
int dfsNumberCounter;
vector<vector<int> > AdjList;
void articulationPointAndBridge(int u) {
    dfs_low[u] = dfs_num[u] = dfsNumberCounter++; // dfs_low[u]
<= dfs_num[u]
    for (int i = 0; i < AdjList[u].size(); i++) {
        int v = AdjList[u][i];
        if (dfs_num[v] == -1) { // a tree edge
            dfs_parent[v] = u; // parent of this children is
me
            if (u == dfsRoot) //// special case special case
                t++; // count children of root
            articulationPointAndBridge(v);
            if (u != dfsRoot && dfs_low[v] >= dfs_num[u])
//// for articulation point for articulation point
                articulation_vertex[u] = true; // store
this information first
            if (dfs_low[v] > dfs_num[u]) // for bridge
                printf(" Edge (%d, %d) is a bridge\n", u,
v);
            dfs_low[u] = min(dfs_low[u], dfs_low[v]); //
update dfs_low[u]
        } else if (v != dfs_parent[u]) // a back edge and not
direct cycle
dfs_low[u] = min(dfs_low[u], dfs_num[v]); // update dfs_low[u]
    }
}
```

Mimimun cycle mean

```
//Finds the minimimun cycle mean in the graph represented by
weight, if no cycle found it returns INF
//Note that the graph represented by weight must be strongly
connected (i.e. there is a path
```

```
//from each node i to each node j). if it isn't then run the SCC
algorithm to find the components
//and then run MMC on each component and take the minimum
//If there is an edge from i to j then weight[i][j] = weight of
that edge, else weight[i][j]=INF
//O(n*m) = O(n^3) where n is the number of nodes and m is the
number of edges
double MMC(vector<vector<int> > weight) {

	//Initialize
	int s = 0, k, u, v, n = weight.size(); //n = nodes num
	//d[a][b] hwa el distance from 0 to node b using exactly a
edges
	vector<vector<int> > d(n + 1, vector<int>(n, INF + 1));
	d[0][s] = 0;
	//Compute the distances
	for (k = 1; k <= n; k++)
		for (v = 0; v < n; v++)
			for (u = 0; u < n; u++)
				if (weight[u][v] < INF)
					d[k][v] = min(d[k][v], d[k - 1][u] +
weight[u][v]);
	//Compute lambda using Karp's theorem
	double lamda = INF;
	for (u = 0; u < n; u++) {
		double currentLamda = -1;
		for (int k = 0; k < n; k++)
			if (d[n][u] < INF && d[k][u] < INF)
				currentLamda = max(currentLamda,
					1.0 * (d[n][u] - d[k][u]) / (n -
k));
		if (currentLamda != -1)
			lamda = min(lamda, currentLamda);
	}
	return lamda;
}
```

Geometry, 2D

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

typedef complex<long double> point;
#define sz(a) ((int)(a).size())
#define all(n) (n).begin(),(n).end()
#define EPS 1e-9
#define OO 1e9
```

```
#define X real()
#define Y imag()
#define vec(a,b) ((b)-(a))
#define polar(r,t) ((r)*exp(point(0,(t))))
#define angle(v) (atan2((v).Y,(v).X))
#define length(v) ((long double)hypot((v).Y,(v).X))
#define lengthSqr(v) (dot(v,v))
#define dot(a,b) ((conj(a)*(b)).real())
#define cross(a,b) ((conj(a)*(b)).imag())
#define rotate(v,t) (polar(v,t))
#define rotateabout(v,t,a)  (rotate(vec(a,v),t)+(a))
#define reflect(p,m) ((conj((p)/(m)))*(m))
#define normalize(p) ((p)/length(p))
#define same(a,b) (lengthSqr(vec(a,b))<EPS)
#define mid(a,b) (((a)+(b))/point(2,0))
#define perp(a) (point(-(a).Y,(a).X))
#define colliner pointOnLine

enum STATE {
      IN, OUT, BOUNDRY
};
```

Counter Clockwise (determine position of a point relative to two other points) (not from multisystem's lib)

```
typedef complex<double> P;
namespace std {
bool operator <(const P& a, const P& b) {
      return real(a) != real(b) ? real(a) < real(b) : imag(a) <
imag(b);
}
}
int ccw(P a, P b, P c) {
      b -= a;
      c -= a;
      if (cross(b, c) > 0)
            return +1; // counter clockwise
      if (cross(b, c) < 0)
            return -1; // clockwise
      if (dot(b, c) < 0)
            return +2; // c--a--b on line
      if (norm(b) < norm(c))
            return -2; // a--b--c on line
      return 0;
}
```

intersect

```
bool intersect(const point &a, const point &b, const point &p,
const point &q, point &ret) {

      //handle degenerate cases
      double d1 = cross(p - a, b - a);
```

```
	double d2 = cross(q - a, b - a);
	ret = (d1 * q - d2 * p) / (d1 - d2);
	if (fabs(d1 - d2) > EPS)
		return 1;
	return 0;
}
```

pointOnLine

```
bool pointOnLine(const point& a, const point& b, const point& p)
{
	return fabs(cross(vec(a,b),vec(a,p))) < EPS;
}
```

pointOnRay

```
inline bool pointOnRay (point a, point b, point p)
{
	return dot(vec(a,b),vec(a,p)) > -EPS && pointOnLine(a, b, p);

}
```

pointOnSegment

```
inline bool pointOnSegment (point a, point b, point p)
{
	return dot(vec(a,b),vec(a,p)) > -EPS && pointOnLine(a, b, p)
			&& dot(vec(b,a),vec(b,p)) > -EPS;
}
```

pointLineDist

```
long double pointLineDist(const point& a, const point& b, const
point& p) {
	if (same(a,b))
		return hypot(a.X - p.X, a.Y - p.Y);

	return fabs(cross(vec(a,b),vec(a,p)) / length(vec(a,b)));
}
```

pointSegmentDist

```
long double pointSegmentDist(const point& a, const point& b,
const point& p) {
	if (dot(vec(a,b),vec(a,p)) < EPS)
		return length(vec(a,p));
	if (dot(vec(b,a),vec(b,p)) < EPS)
		return length(vec(b,p));
	return pointLineDist(a, b, p);
}
```

lineLatticePointsCount

```
int lineLatticePointsCount(int x1, int y1, int x2, int y2) {
```

```
	return abs(__gcd(x1 - x2, y1 - y2)) + 1;
}
```

triangleAreaBH

```
long double triangleAreaBH(long double b, long double h) {
	return b * h / 2;
}
```

triangleArea2sidesAngle

```
long double triangleArea2sidesAngle(long double a, long double b,
long double t) {
	return fabs(a * b * sin(t) / 2);
}
```

triangleArea2anglesSide

```
long double triangleArea2anglesSide(long double t1, long double
t2,
		long double s) {
	return fabs(s * s * sin(t1) * sin(t2) / (2 * sin(t1 +
t2)));
}
```

triangleArea3sides

```
long double triangleArea3sides(long double a, long double b, long
double c) {
	long double s((a + b + c) / 2);
	return sqrt(s * (s - a) * (s - b) * (s - c));
}
```

triangleArea3points

```
long double triangleArea3points(const point& a, const point& b,
const point& c) {
	return fabs(cross(a,b) + cross(b,c) + cross(c,a)) / 2;
}
```

cosRule

```
//get angle opposite to side a
long double cosRule(long double a, long double b, long double c)
{
	// Handle denom = 0
	long double res = (b * b + c * c - a * a) / (2 * b * c);
	if (res > 1)
		res = 1;
	if (res < -1)
		res = -1;
	return acos(res);
}
```

sinRuleAngle

```
long double sinRuleAngle(long double s1, long double s2, long
double a1) {
	// Handle denom = 0
	long double res = s2 * sin(a1) / s1;
	if (res > 1)
		res = 1;
	if (res < -1)
		res = -1;
```

```
	return asin(res);
}
```

sinRuleSide

```
long double sinRuleSide(long double s1, long double a1, long
double a2) {
	// Handle denom = 0
	long double res = s1 * sin(a2) / sin(a1);
	return fabs(res);
}
```

circleLineIntersection

```
int circleLineIntersection(const point& p0, const point& p1,
const point& cen,
		long double rad, point& r1, point & r2) {

	if (same(p0,p1)) {
		if (fabs(lengthSqr(vec(p0,cen)) - (rad * rad)) < EPS)
{
			r1 = r2 = p0;
			return 1;
		}
		return 0;
	}
	long double a, b, c, t1, t2;
	a = dot(p1-p0,p1-p0);
	b = 2 * dot(p1-p0,p0-cen);
	c = dot(p0-cen,p0-cen) - rad * rad;
	double det = b * b - 4 * a * c;
	int res;
	if (fabs(det) < EPS)
		det = 0, res = 1;
	else if (det < 0)
		res = 0;
	else
		res = 2;
	det = sqrt(det);
	t1 = (-b + det) / (2 * a);
	t2 = (-b - det) / (2 * a);
	r1 = p0 + t1 * (p1 - p0);
	r2 = p0 + t2 * (p1 - p0);
	return res;
}
```

circleCircleIntersection

```
int circleCircleIntersection(const point &c1, const long
double&r1,
		const point &c2, const long double&r2, point &res1,
point &res2) {
	if (same(c1,c2) && fabs(r1 - r2) < EPS) {
		res1 = res2 = c1;
		return fabs(r1) < EPS ? 1 : OO;
	}
	long double len = length(vec(c1,c2));
```

```
      if (fabs(len - (r1 + r2)) < EPS || fabs(fabs(r1 - r2) -
len) < EPS) {
            point d, c;
            long double r;
            if (r1 > r2)
                  d = vec(c1,c2), c = c1, r = r1;
            else
                  d = vec(c2,c1), c = c2, r = r2;
            res1 = res2 = normalize(d) * r + c;
            return 1;
      }
      if (len > r1 + r2 || len < fabs(r1 - r2))
            return 0;
      long double a = cosRule(r2, r1, len);
      point c1c2 = normalize(vec(c1,c2)) * r1;
      res1 = rotate(c1c2,a) + c1;
      res2 = rotate(c1c2,-a) + c1;
      return 2;
}
```

circle2

```
void circle2(const point& p1, const point& p2, point& cen, long
double& r) {
      cen = mid(p1,p2);
      r = length(vec(p1,p2)) / 2;
}
```

circle3

```
bool circle3(const point& p1, const point& p2, const point& p3,
point& cen,
            long double& r) {
      point m1 = mid(p1,p2);
      point m2 = mid(p2,p3);
      point perp1 = perp(vec(p1,p2));
      point perp2 = perp(vec(p2,p3));
      bool res = intersect(m1, m1 + perp1, m2, m2 + perp2, cen);
      r = length(vec(cen,p1));
      return res;
}
```

circlePoint

```
STATE circlePoint(const point & cen, const long double & r, const
point& p) {
      long double lensqr = lengthSqr(vec(cen,p));
      if (fabs(lensqr - r * r) < EPS)
            return BOUNDRY;
      if (lensqr < r * r)
            return IN;
      return OUT;
}
```

tangentPoints

```
int tangentPoints(const point & cen, const long double & r, const
point& p, point &r1, point &r2) {
      STATE s = circlePoint(cen, r, p);
```

```
	if (s != OUT) {
		r1 = r2 = p;
		return s == BOUNDRY;
	}
	point cp = vec(cen,p);
	long double h = length(cp);
	long double a = acos(r / h);
	cp = normalize(cp) * r;
	r1 = rotate(cp,a) + cen;
	r2 = rotate(cp,-a) + cen;
	return 2;
}
```

minimum enclosing circle

```
//init p array with the points and ps with the number of points
//cen and rad are result circle
//you must call random_shuffle(p,p+ps); before you call mec
#define MAXPOINTS 100000
point p[MAXPOINTS], r[3], cen;
int ps, rs;
long double rad;

void mec() {
	if (rs == 3) {
		circle3(r[0], r[1], r[2], cen, rad);
		return;
	}
	if (rs == 2 && ps == 0) {
		circle2(r[0], r[1], cen, rad);
		return;
	}
	if (!ps) {
		cen = r[0];
		rad = 0;
		return;
	}
	ps--;
	mec();
	if (circlePoint(cen, rad, p[ps]) == OUT) {
		r[rs++] = p[ps];
		mec();
		rs--;
	}
	ps++;

}
```

polygonArea

```
//to check if the points are sorted anti-clockwise or clockwise
//remove the fabs at the end and it will return -ve value if
clockwise
```

```
long double polygonArea(const vector<point>&p) {
	long double res = 0;
	for (int i = 0; i < sz(p); i++) {
		int j = (i + 1) % sz(p);
		res += cross(p[i],p[j]);
	}
	return fabs(res) / 2;
}
```

polyginCentroid

```
// return the centroid point of the polygon
// The centroid is also known as the "centre of gravity" or the
"center of mass". The position of the centroid
// assuming the polygon to be made of a material of uniform
density.
point polyginCentroid(vector<point> &polygon) {
	point res(0, 0);
	long double a = 0;

	for (int i = 0; i < (int) polygon.size(); i++) {
		int j = (i + 1) % polygon.size();

		res.X += (polygon[i].X + polygon[j].X) * (polygon[i].X
* polygon[j].Y
				- polygon[j].X * polygon[i].Y);

		res.Y += (polygon[i].Y + polygon[j].Y) * (polygon[i].X
* polygon[j].Y
				- polygon[j].X * polygon[i].Y);

		a += polygon[i].X * polygon[j].Y - polygon[i].Y *
polygon[j].X;
	}

	a *= 0.5;
	res.X /= 6 * a;
	res.Y /= 6 * a;

	return res;
}
```

picksTheorm

```
int picksTheorm(vector<point>& p) {
	long double area = 0;
	int bound = 0;
	for (int i = 0; i < sz(p); i++) {
		int j = (i + 1) % sz(p);
		area += cross(p[i],p[j]);
		point v = vec(p[i],p[j]);
		bound += abs(__gcd((int) v.X, (int) v.Y));
	}
	area /= 2;
	area = fabs(area);
```

```
	return round(area - bound / 2 + 1);
}
```

picksTheorm

```
//count interior
int picksTheorm(int a, int b) {
	return a - b / 2 + 1;
}
```

polygonCut

```
void polygonCut(const vector<point>& p, const point&a, const
point&b, vector<point>& res) {
	res.clear();
	for (int i = 0; i < sz(p); i++) {
		int j = (i + 1) % sz(p);
		bool in1 = cross(vec(a,b),vec(a,p[i])) > EPS;
		bool in2 = cross(vec(a,b),vec(a,p[j])) > EPS;
		if (in1)
			res.push_back(p[i]);
		if (in1 ^ in2) {
			point r;
			intersect(a, b, p[i], p[j], r);
			res.push_back(r);
		}
	}
}
```

convexPolygonIntersect

```
//assume that both are anti-clockwise
void convexPolygonIntersect(const vector<point>& p, const
vector<point>& q,
		vector<point>& res) {
	res = q;
	for (int i = 0; i < sz(p); i++) {
		int j = (i + 1) % sz(p);
		vector<point> temp;
		polygonCut(res, p[i], p[j], temp);
		res = temp;
		if (res.empty())
			return;
	}
}
```

voronoi

```
void voronoi(const vector<point> &pnts, const vector<point>&
rect,
		vector<vector<point> > &res) {
	res.clear();
	for (int i = 0; i < sz(pnts); i++) {
		res.push_back(rect);
		for (int j = 0; j < sz(pnts); j++) {
			if (j == i)
				continue;
			point p = perp(vec(pnts[i],pnts[j]));
			point m = mid(pnts[i],pnts[j]);
```

```
                vector<point> temp;
                polygonCut(res.back(), m, m + p, temp);
                res.back() = temp;
            }
        }
}
```

pointInPolygon

```
STATE pointInPolygon(const vector<point>& p, const point &pnt) {
        point p2 = pnt + point(1, 0);
        int cnt = 0;
        for (int i = 0; i < sz(p); i++) {
            int j = (i + 1) % sz(p);
            if (pointOnSegment(p[i], p[j], pnt))
                return BOUNDRY;
            point r;
            intersect(pnt, p2, p[i], p[j], r);
            if (!pointOnRay(pnt, p2, r))
                continue;
            if (same(r,p[i]) || same(r,p[j]))
                if (fabs(r.Y - min(p[i].Y, p[j].Y)) < EPS)
                    continue;
            if (!pointOnSegment(p[i], p[j], r))
                continue;
            cnt++;
        }
        return cnt & 1 ? IN : OUT;
}
```

sortAntiClockWise

```
struct cmp {
        point about;
        cmp(point c) {
            about = c;
        }
        bool operator()(const point& p, const point& q) const {
            double cr = cross(vec(about,p),vec(about,q));
            if (fabs(cr) < EPS)
                return make_pair(p.Y, p.X) < make_pair(q.Y, q.X);
            return cr > 0;
        }
};

void sortAntiClockWise(vector<point>& pnts) {
        point mn(1 / 0.0, 1 / 0.0);
        for (int i = 0; i < sz(pnts); i++)
            if (make_pair(pnts[i].Y, pnts[i].X) < make_pair(mn.Y,
mn.X))
                mn = pnts[i];

        sort(all(pnts),cmp(mn)) ;
}
```

convexHull

```
void convexHull(vector<point> pnts, vector<point> &convex) {
	sortAntiClockWise(pnts);
	convex.clear();
	convex.push_back(pnts[0]);
	if (sz(pnts) == 1)
		return;
	convex.push_back(pnts[1]);
	for (int i = 2; i <= sz(pnts); i++) {
		point c = pnts[i % sz(pnts)];
		while (sz(convex) > 1) {
			point b = convex.back();
			point a = convex[sz(convex) - 2];
			if (cross(vec(b,a),vec(b,c)) < -EPS)
				break;
			convex.pop_back();
		}
		if (i < sz(pnts))
			convex.push_back(pnts[i]);
	}
}
```

Geometry, 3D

Distance On Sphere

```
/* Spherical distance from longitude & latitude */
double SphericalDist(double p_long, double p_lat, double q_long,
double q_lat,double r) {
double a = (1.0 - cos(q_lat - p_lat)) / 2, b = cos(p_lat) *
cos(q_lat)* (1.0 - cos(q_long - p_long)) / 2;
double c = 2 * atan2(sqrt(a + b), sqrt(1 - a - b));
return r * c; // more accurate
}
```

3D Point

```
#define EPS 1e-9
double ONE = 1;
struct point3D {
	double v[3];
	point3D() {
		for (int i = 0; i < 3; ++i) {
			this->v[i] = 0;
		}
	}
	point3D(double v[3]) {
		for (int i = 0; i < 3; ++i) {
			this->v[i] = v[i];
		}
	}
	double& operator [](int idx) {
```

```
        return idx < 3 ? v[idx] : (ONE = 1);
}
double operator [](int idx) const {
        return idx < 3 ? v[idx] : 1;
}
double& x() {
        return v[0];
}
double& y() {
        return v[1];
}
double& z() {
        return v[2];
}
point3D operator +(const point3D& t) const {
        point3D ret;
        for (int i = 0; i < 3; ++i) {
                ret.v[i] = v[i] + t.v[i];
        }
        return ret;
}
point3D operator -(const point3D& t) const {
        point3D ret;
        for (int i = 0; i < 3; ++i) {
                ret.v[i] = v[i] - t.v[i];
        }
        return ret;
}
point3D operator *(const double& t) const {
        point3D ret;
        for (int i = 0; i < 3; ++i) {
                ret.v[i] = v[i] * t;
        }
        return ret;
}
point3D operator /(const double& t) const {
        point3D ret;
        for (int i = 0; i < 3; ++i) {
                ret.v[i] = v[i] / t;
        }
        return ret;
}
double Length() {
        double sum = 0;
        for (int i = 0; i < 3; ++i) {
                sum += v[i] * v[i];
        }
        return sqrt(sum);
}
double Dot(const point3D& t) const {
        double sum = 0;
        for (int i = 0; i < 3; ++i) {
```

```
                        sum += v[i] * t.v[i];
                }

                return sum;
        }
        point3D Cross(const point3D& t) const {
                double arr[] = { v[1] * t.v[2] - v[2] * t.v[1],
v[2] * t.v[0] - v[0]
                                * t.v[2], v[0] * t.v[1] - v[1] *
t.v[0] };
                return point3D(arr);
        }
        point3D Normalize() {
                return point3D(v) / Length();
        }
};
```

4x4 Transformation Matrix

```
struct matrix {
        double arr[4][4];
        matrix operator *(const matrix& m) const {
                matrix ret;
                memset(ret.arr, 0, sizeof(ret.arr));
                for (int i = 0; i < 4; ++i) {
                        for (int j = 0; j < 4; ++j) {
                                for (int k = 0; k < 4; ++k) {
                                        ret.arr[i][j] +=
arr[i][k] * m.arr[k][j];
                                }
                        }
                }
                return ret;
        }
        point3D operator *(const point3D& m) const {
                point3D ret;
                for (int i = 0; i < 4; ++i) {
                        for (int j = 0; j < 4; ++j) {
                                ret[i] += arr[i][j] * m[j];
                        }
                }
                return ret;
        }
        double& operator()(int i, int j) {
                return arr[i][j];
        }
        const double& operator()(int i, int j) const {
                return arr[i][j];
        }
};
```

4x4 Identity Matrix

```
matrix Identity() {
        matrix ret;
        for (int i = 0; i < 4; ++i) {
                for (int j = 0; j < 4; ++j) {
                        ret(i, j) = i == j;
                }
        }
        return ret;
}
```

3D Translation Matrix

```
matrix translate(const point3D& v, int dir = 1) {
        matrix ret = Identity();
        for (int i = 0; i < 3; ++i) {
                ret(i, 3) = v[i] * dir;
        }
        return ret;
}
```

3D Rotation around Z Axis Matrix

```
matrix rotateZ(double angle) {
        matrix ret = Identity();
        ret(0, 0) = ret(1, 1) = cos(angle);
        ret(0, 1) = -(ret(1, 0) = sin(angle));
        return ret;
}
```

3D Transform coordinate system Matrix

```
matrix transformSystem(const point3D& u, const point3D& v, const
point3D& w) {
        matrix ret = Identity();
        for (int j = 0; j < 3; ++j) {
                ret(0, j) = u[j];
                ret(1, j) = v[j];
                ret(2, j) = w[j];
        }
        return ret;
}
```

3D Inverse Transform coordinate system Matrix

```
matrix ItransformSystem(const point3D& u, const point3D& v, const
point3D& w) {
        matrix ret = Identity();

        for (int j = 0; j < 3; ++j) {
                ret(j, 0) = u[j];
                ret(j, 1) = v[j];
                ret(j, 2) = w[j];
        }
```

```
        return ret;
}
```

3D Get Perpendicular on two Vectors

```
void getPrep(point3D & w, point3D & v, point3D & u) {
        w = w.Normalize();
        for (int i = 0; i < 3; ++i) {
                if (fabs(w[i]) > EPS) {
                        int j = (i + 1) % 3;
                        int k = (i + 2) % 3;
                        v[i] = w[j];
                        v[j] = -w[i];
                        v[k] = 0;
                        v = v.Normalize();
                        break;
                }
        }
        u = v.Cross(w);
}
```

3D Rotation around General line Matrix

```
matrix rotate(const point3D& p, const point3D& q, double angle) {
        point3D w((p - q).Normalize()), u, v;
        getPrep(w, v, u);

        return translate(p, 1) * ItransformSystem(u, v, w) *
rotateZ(angle)
                        * transformSystem(u, v, w) * translate(p,
-1);
}
```

Line Plane Intersection

```
bool linePlaneIntersect(const point3D& p, const point3D& q, const
point3D& pp,
                const point3D& N, point3D& ret) {
        double d = (q - p).Dot(N);
        if (fabs(d) < EPS)
                return false;

        double t = (pp - p).Dot(N) / d;
        ret = p + (q - p) * t;
        return true;
}
```

Calculate the intersection of a line (not line segment) and a sphere

```
/*   -There are potentially two points of intersection given by
 p = p1 + mu1 (p2 - p1)
 p = p1 + mu2 (p2 - p1)
```

```
 -To apply this to two dimensions, that is, the intersection of a
line and a circle
 simply remove the z component from the above mathematics.*/

//If mu isn't between 0 and 1 then the intersection point isn't
between p1,p2
bool intersectLineSphere(point3D p1, point3D p2, point3D sc,
double r,
           double& mu1, double& mu2) {

      double a, b, c;
      double bb4ac;
      point3D dp;

      dp.x() = p2.x() - p1.x();
      dp.y() = p2.y() - p1.y();
      dp.z() = p2.z() - p1.z();
      a = dp.x() * dp.x() + dp.y() * dp.y() + dp.z() * dp.z();
      b = 2
                  * (dp.x() * (p1.x() - sc.x()) + dp.y() * (p1.y()
- sc.y())
                              + dp.z() * (p1.z() - sc.z()));
      c = sc.x() * sc.x() + sc.y() * sc.y() + sc.z() * sc.z();
      c += p1.x() * p1.x() + p1.y() * p1.y() + p1.z() * p1.z();
      c -= 2 * (sc.x() * p1.x() + sc.y() * p1.y() + sc.z() *
p1.z());
      c -= r * r;
      bb4ac = b * b - 4 * a * c;
      if (fabs(a) < EPS || bb4ac < 0) {
            mu1 = 0;
            mu2 = 0;
            return false;
      }

      mu1 = (-b + sqrt(bb4ac)) / (2 * a);
      mu2 = (-b - sqrt(bb4ac)) / (2 * a);

      return true;
}
```

Tetrahedron centroid

```
point3D tetra_center(const point3D & a, const point3D & b, const
point3D & c,
                  const point3D & d) {
        return (a + b + c + d) / 4;
}
```

Tetrahedron volume

```
double tetra_volume(const point3D & a, const point3D & b, const
point3D & c,
                  const point3D & d) {
```

```
        return fabs((a - d).Dot((b - d).Cross(c - d))) / 6;
}
```

Spherical To Cartesian Coordiantes

```
/*Note that rho represents the distance from the origin,
 -phi (aka latitude) is the angle (in radians) between the vector
from the origin to the point represneted by this coordinate and
the z-axis
 theta (aka longitude) is the angle (in radians) from the
positive xz-plane to the point*/
struct spherical {
      double rho, phi, theta;
};
cartesian spherical2cartesian(spherical sp) {
      cartesian cp;
      cp.x = sp.rho * cos(sp.phi) * cos(sp.theta);
      cp.y = sp.rho * cos(sp.phi) * sin(sp.theta);
      cp.z = sp.rho * sin(sp.phi);
      return cp;
}
spherical cartesian2spherical(cartesian cp) {
      spherical sp;
      sp.rho = sqrt(cp.x * cp.x + cp.y * cp.y + cp.z * cp.z);
      sp.phi = asin(cp.y / sp.rho);
      sp.theta = cp.x >= 0 ? acos(cp.z / (sp.rho * cos(sp.phi)))
: -acos(
                  cp.z / (sp.rho * cos(sp.phi)));
      return sp;
}
```

Data structures and string algorithms

RMQ

```
struct Node { //RMQ (O(Log n))//
      int start, end, minimum;
      bool operator <(const Node& t) const {
            return minimum < t.minimum;
      }
} A[MAXSIZE];
vector<int> Val;
int RMQ(int curNodeInd, int i, int j) {
      if (i == j)
            return i;
      if (A[curNodeInd].start >= i && A[curNodeInd].end <= j)
            return A[curNodeInd].minimum;
      if (A[curNodeInd].start > j || A[curNodeInd].end < i)
            return -1;
      int r = RMQ(curNodeInd * 2 + 1, i, j);
      int l = RMQ(curNodeInd * 2 + 2, i, j);
      if (r == -1)
```

```
            return l;
      if (l == -1)
            return r;
      if (Val[r] < Val[l])
            return r;
      return l;
}
int precompute(int cur, int s, int t) {
      A[cur].start = s;
      A[cur].end = t;
      if (s == t) {
            return A[cur].minimum = t;
      }
      int r = precompute(cur * 2 + 1, s, (s + t) / 2);
      int l = precompute(cur * 2 + 2, ((s + t) / 2) + 1, t);
      return A[cur].minimum = Val[r] < Val[l] ? r : l;
}
///// RMQ (O(1))///////
int MIN[MAXN][LOGMAXN], MAX[MAXN][LOGMAXN], A[MAXN];
void processRMINQ(int N) {
      int i, j;
      for (i = 0; i < N; i++)
            MIN[i][0] = i;
for (j = 1; 1 << j <= N; j++)
for (i = 0; i + (1 << j) - 1 < N; i++)
if (A[MIN[i][j - 1]] < A[MIN[i + (1 << (j - 1))][j - 1]])
MIN[i][j] = MIN[i][j - 1];
else MIN[i][j] = MIN[i + (1 << (j - 1))][j - 1];
}
int RMinQ(int i, int j) {
      int k = log(j - i + 1) / log(2);
      return min(A[MIN[i][k]], A[MIN[j - (1 << k) + 1][k]]);
}
```

LCA

```
vector<int> sTree;
int n;
vector<int> levels;
void comp(int S = 0, int E = n - 1, int ind = 0) {
      if (S == E) {
            sTree[ind] = S;
            return;
      }
      comp(S, (S + E) / 2, ind * 2 + 1);
      comp((S + E) / 2 + 1, E, ind * 2 + 2);
      sTree[ind] =
                  levels[sTree[ind * 2 + 1]] < levels[sTree[ind * 2
+ 2]] ?
                              sTree[ind * 2 + 1] : sTree[ind * 2 +
2];
}
```

```
int RMQ(int qS, int qE, int rS = 0, int rE = n - 1, int n = 0) {
	if (qS > rE || qE < rS)
		return -1;
	if (rS >= qS && rE <= qE)
		return sTree[n];
	int s1 = RMQ(qS, qE, rS, (rS + rE) / 2, n * 2 + 1);
	int s2 = RMQ(qS, qE, (rS + rE) / 2 + 1, rE, n * 2 + 2);
	return s1 == -1 ? s2 : (s2 == -1 ? s1 : (levels[s1] <
levels[s2] ? s1 : s2));
}
int N;
vector<vector<int> > g;
vector<int> tour;
vector<int> tourInd;
void dfs(int n, int l) {
	int ind = tour.size();
	tour.push_back(n);
	levels.push_back(l);
	for (int i = 0; i < g[n].size(); i++) {
		dfs(g[n][i], l + 1);
		tour.push_back(n);
		levels.push_back(l);
	}
	tourInd[n] = ind;
}
void generate(int root) {
	levels.clear();
	tour.clear();
	tourInd = vector<int>(N + 1);
	dfs(root, 0);
	n = levels.size();
	sTree = vector<int>(1 << int(2 + 1e-9 + log(n) / log(2)));
	comp();
}
int getQuery(int t1, int t2) {
	if (tourInd[t1] < tourInd[t2])
		return tour[RMQ(tourInd[t1], tourInd[t2])];
	return tour[RMQ(tourInd[t2], tourInd[t1])];
}
```

LCA on DAG

```
#define MX 1000
typedef vector<vector<int> > vii;
#define pb push_back
int in1[MX];
int in2[MX];
int out[MX];
vii g, gr;
int n, m;
bitset<MX> des[MX];
bitset<MX> anc[MX];
```

```
int ind[MX];
bitset<MX> vis[MX];
int mat[MX][MX];
void calc1() {
	mem(des, 0);
	queue<int> q;
	rep(i,n)
		if (!out[i])
			q.push(i);
	while (!q.empty()) {
		int t = q.front();
		q.pop();
		des[t][t] = 1;
		rep(i,sz(g[t]))
			des[t] |= des[g[t][i]];
		rep(i,sz(r[t])) {
			out[gr[t][i]]--;
			if (!out[gr[t][i]])
				q.push(gr[t][i]);
		}
	}
}
void calc2() {
	mem(anc, 0);
	queue<int> q;
	rep(i,n)
		if (!in1[i])
			q.push(i);
	int cur = 0;
	while (!q.empty()) {
		int t = q.front();
		ind[t] = cur++;
		q.pop();
		anc[t][t] = 1;
		rep(i,sz(gr[t]))
			anc[t] |= anc[gr[t][i]];
		rep(i,sz(g[t])) {
			in1[g[t][i]]--;
			if (!in1[g[t][i]])
				q.push(g[t][i]);
		}
	}
}
void calc3() {
	mem(anc, 0);
	queue<int> q;
	rep(i,n)
		if (!in2[i]) {
			q.push(i);
			rep(j,n)
				if (des[i][j]) {
					mat[i][j] = mat[j][i] = i;
```

```
                              vis[i][j] = vis[j][i] = 1;
                        }
                  }
            while (!q.empty()) {
                  int t = q.front();
                  q.pop();
                  rep(i,sz(gr[t])) {
                        rep(j,n) {
                              if (des[t][j])
                                    mat[t][j] = mat[j][t] = t;
                              else if (des[j][t])
                                    mat[t][j] = mat[j][t] = j;
                              else {
                                    if (!vis[j][t] ||
ind[mat[gr[t][i]][j]] > ind[mat[j][t]])
                                          mat[t][j] = mat[j][t] =
mat[gr[t][i]][j];
                              }
                              vis[t][j] = vis[j][t] = 1;
                        }
                  }
                  rep(i,sz(g[t])) {
                        in2[g[t][i]]--;
                        if (!in2[g[t][i]])
                              q.push(g[t][i]);
                  }
            }
}
void init() {
      g = vector<vector<int> >(n);
      gr = vector<vector<int> >(n);
      mem(in1, 0);
      mem(in2, 0);
      mem(out, 0);
}
void addEdge(int from, int to) {
      g[from].pb(to);
      gr[to].pb(from);
      in1[to]++;
      in2[to]++;
      out[from]++;
}
void calc() {
      calc1();
      calc2();
      mem(vis, 0);
      calc3();
}
```

BIT

```
//insert 5 3 9, put 1 at 5, 3 and 9. add(5, 1), add(3, 1), add(9,
1);
//find(3) returns 9, find(2) returns 5, find(1) returns 3 //find
is lower bound
//get(9) returns 3, get(5) returns 2, get(3) returns 1

struct BIT {
	vector<long long> v;
	BIT(int s) {
		resize(s);
	}
	void clear() {
		v.clear();
	}
	BIT() {
	}
	void resize(int s) {
		s = 1 << (int) ceil(log(1.0 * s) / log(2.) + EPSILON);
		v.resize(s);
	}
	long long get(int i) {
		i++;
		long long r = 0;
		while (i) {
			r += v[i - 1];
			i -= i & -i;
		}
		return r;
	}
	void add(int i, long long val) {
		i++;
		while (i <= (int) v.size()) {
			v[i - 1] += val;
			i += i & -i;
		}
	}
	int find(long long val) {
		int s = 0;
		int m = v.size() >> 1;
		while (m) {
			if (v[s + m - 1] < val)
				val -= v[(s += m) - 1];
			m >>= 1;
		}
		return s;
	}
};
```

2D BIT

```
int arr[R][C], mat[R][C];
```

```
void add(int i, int jj, int v) {
	i++;
	jj++;
	while (i <= R) {
		int j = jj;
		while (j <= C) {
			arr[i - 1][j - 1] += v;
			j += (j & -j);
		}
		i += (i & -i);
	}
}
int get(int i, int jj) {
	int v = 0;
	i++;
	jj++;
	while (i) {
		int j = jj;
		while (j) {
			v += arr[i - 1][j - 1];
			j -= (j & -j);
		}
		i -= (i & -i);
	}
	return v;
}
int get2D(int b, int l, int t, int r) {
	int v = 0;
	v += get(t, r);
	v -= get(t, l - 1);
	v -= get(b - 1, r);
	v += get(b - 1, l - 1);
	return v;
}
```

Suffix Arrays

```
#include<iostream>
#include<cstdio>
using namespace std;
#define Max_N 1000
```

O(N^2 lg N)

```
// buildSA O(n^2 log(n) )
char str[Max_N];
int suffix[Max_N];
struct comp {
	bool operator()(int a, int b) const {
		return strcmp(str + a, str + b) < 0;
	}
};
void buildSA() {
```

```
	int n;
	for (n = 0; n == 0 || str[n - 1]; n++)
		suffix[n] = n;
	sort(suffix, suffix + n, comp());
}
```

O(N (lg N)^2)

```
// buildSA O(n(logn)^2)
char str[Max_N];
int suffix[Max_N];
int group[Max_N];
int tg[Max_N];

struct comp {
	int h;
	comp(int h) :
		h(h) {
	}

	bool operator ()(const int& s1, const int& s2) const {
		return group[s1] < group[s2] || group[s1] == group[s2]
&& group[s1 + h]
				< group[s2 + h];
	}
};

void buildSA() {
	int n;
	for (n = 0; n == 0 || str[n - 1]; n++) {
		suffix[n] = n;
		group[n] = str[n];
	}
	sort(suffix, suffix + n, comp(0));
	tg[0] = tg[n - 1] = 0;
	for (int h = 1; tg[n - 1] != n - 1; h <<= 1) {
		comp c(h);
		sort(suffix, suffix + n, c);
		for (int i = 1; i < n; ++i) {
			tg[i] = tg[i - 1] + c(suffix[i - 1], suffix[i]);
		}
		for (int i = 0; i < n; ++i) {
			group[suffix[i]] = tg[i];
		}
	}
}
```

O(N lg N)

```
// buildSA O(nlogn)
char str[Max_N];
int suffix[Max_N];
int group[Max_N];
```

```
int tg[Max_N < 128 ? 128 : Max_N];
int newSuffix[Max_N];
int gstart[Max_N];

void buildSA() {
      int n;
      memset(tg, -1, (sizeof tg[0]) * 128);
      for (n = 0; n == 0 || str[n - 1]; n++) {
            newSuffix[n] = tg[str[n]];
            tg[str[n]] = n;
      }
      int ng = -1, j = 0;
      for (int i = 0; i < 128; ++i) {
            if (tg[i] != -1) {
                  gstart[++ng] = j;
                  int cur = tg[i];
                  while (cur != -1) {
                        suffix[j++] = cur;
                        group[cur] = ng;
                        cur = newSuffix[cur];
                  }
            }
      }
      tg[0] = tg[n - 1] = 0;
      newSuffix[0] = suffix[0];
      for (int h = 1; tg[n - 1] != n - 1; h <<= 1) {
            for (int i = 0; i < n; ++i) {
                  j = suffix[i] - h;
                  if (j < 0)
                        continue;
                  newSuffix[gstart[group[j]]++] = j;
            }
            for (int i = 1; i < n; ++i) {
                  bool newgroup = group[newSuffix[i - 1]] <
group[newSuffix[i]]
                              || group[newSuffix[i - 1]] ==
group[newSuffix[i]]
                                          && group[newSuffix[i - 1] +
h] < group[newSuffix[i]
                                                      + h];
                  tg[i] = tg[i - 1] + newgroup;
                  if (newgroup)
                        gstart[tg[i]] = i;
            }
            for (int i = 0; i < n; ++i) {
                  suffix[i] = newSuffix[i];
                  group[suffix[i]] = tg[i];
            }
      }
}
```

LCP

```
int rank[Max_N];
int lcp[Max_N];

void buildLCP() {
	int n;
	for (n = 0; n == 0 || str[n - 1]; n++)
		rank[suffix[n]] = n;

	int c = 0;
	for (int i = 0; i < n; i++) {
		if (rank[i]) {
			int j = suffix[rank[i] - 1];
			while (str[i + c] == str[j + c])
				c++;
		}
		lcp[rank[i]] = c;

		if (c)
			c--;
	}
}


struct cmp {
	int k;
	cmp(int _k) {
		k = _k;
	}
	bool operator ()(const int &i, const int &j)const {
		return str[i+k]<str[j+k];
	}
};

//if u search for small strings in a large string use suffix array with
this method to search for these small strings using binary search
bool search(char *cur) {
	int s = 0, e = strlen(str) + 1;
	int f = 1;
	for (int j = 0; cur[j]; ++j) {
		s = lower_bound(suffix+s, suffix+e,
cur-str, cmp(j)) - suffix;

e = upper_bound(suffix+s, suffix+e, cur-str,
cmp(j)) - suffix;
		if(s >= e) {
			f = 0;
			break;
		}
	}
	return f;
}
```

Suffix Tree

```
struct edge {
	int to, s, e;
	edge(int to, int s, int e) :
			to(to), s(s), e(e) {
	}
	edge() {
	}
};
struct _hash {
	int operator ()(const pair<int, char>& t) const {
		return t.first * 257 + t.second;
	}
};
char str[MAXSIZE];
int size, strNum, mx = 0, nnodes;
hash_map<pair<int, char> , edge, _hash> g;
typedef hash_map<pair<int, char> , edge, _hash>::iterator iter;
vector<int> res, f;
bool getEdge(int s, char t, int& kd, int&pd, int&sd) {
	if (s == -1) {
		sd = kd = pd = 0;
		return true;
	}
	iter it = g.find(make_pair(s, t));
	if (it == g.end())
		return false;
	kd = it->second.s;
	pd = it->second.e;
	sd = it->second.to;
	return true;
}
pair<int, int> canonize(int s, int k, int p) {
	if (p < k)
		return make_pair(s, k);
	int kd, pd, sd;
	getEdge(s, str[k], kd, pd, sd);
	while (pd - kd <= p - k) {
		k += pd - kd + 1;
		s = sd;
		if (k <= p)
			getEdge(s, str[k], kd, pd, sd);
	}
	return make_pair(s, k);
}
void init() {
	g.clear();
	f.clear();
	g.resize(size * 2);
	f.reserve(size * 2);
	nnodes = 1;
```

```
	f.push_back(-1);
}
pair<bool, int> test_and_split(int s, int k, int p, char t) {
	int kd, pd, sd;
	if (k <= p) {
		getEdge(s, str[k], kd, pd, sd);
		if (t == str[kd + p - k + 1])
			return make_pair(true, s);
		int r = nnodes++;
		f.push_back(-1);
		g[make_pair(s, str[kd])] = edge(r, kd, kd + p - k);
		g[make_pair(r, str[kd + p - k + 1])] = edge(sd, kd + p
- k + 1, pd);
		return make_pair(false, r);
	}
	return make_pair(getEdge(s, t, kd, pd, sd), s);
}
pair<int, int> update(int s, int k, int i) {
	int oldr = 0;
	pair<bool, int> temp = test_and_split(s, k, i - 1, str[i]);
	while (!temp.first) {
		int r = temp.second;
		int rd = nnodes++;
		f.push_back(-1);
		g[make_pair(r, str[i])] = edge(rd, i, size);
		if (oldr)
			f[oldr] = r;
		oldr = r;
		pair<int, int> c = canonize(f[s], k, i - 1);
		s = c.first;
		k = c.second;
		temp = test_and_split(s, k, i - 1, str[i]);
	}
	if (oldr)
		f[oldr] = s;
	return make_pair(s, k);
}
void insert() {
	size = strlen(str) - 1;
	pair<int, int> temp(0, 0); // s,k
	int i = 0;
	init();
	while (str[i]) {
		temp = update(temp.first, temp.second, i);
		temp = canonize(temp.first, temp.second, i++);
	}
}

vector<vector<char> > adj;
vector<pair<int, char> > parent;
void constructAdjacency() {
	adj.clear();
```

```
	adj.resize(f.size());
	parent.clear();
	parent.resize(f.size());
	parent[0] = make_pair(-1, -1);
	iter it = g.begin();
	for (; it != g.end(); it++) {
		adj[it->first.first].push_back(str[it->second.s]);
		parent[it->second.to] = make_pair(it->first.first,
str[it->second.s]);
	}
}
void sortAdjacency() {
	for (int i = 0; i < adj.size(); i++)
		sort(adj[i].begin(), adj[i].end());
}
int n, m, s2;
vector<int> bestNode;
int len[100], strInd[MAXSIZE];
vector<pair<int, int> > que;
void bfs() {
	int i, sz;
	que.clear();
	que.push_back(make_pair(0, 0));
	for (int ind = 0; ind < que.size(); ind++)
		for (i = 0; i < adj[que[ind].first].size(); i++) {
			iter it = g.find(make_pair(que[ind].first,
adj[que[ind].first][i]));
			que.push_back(
					make_pair(it->second.to,
						que[ind].second + it-
>second.e - it->second.s + 1));
		}
}
void findLongest() {
	int best = -1;
	vector<bitset<100> > has(f.size());
	iter it; // empty string is not counted as common
substring, to count it, make i >= 0 in the for loop, and ensure
that it != g.end()
	for (int i = que.size() - 1; i > 0; i--) {
		it = g.find(parent[que[i].first]);
		int ind = strInd[it->second.s];
		if (strInd[it->second.s] != strInd[it->second.e + 1])
			has[que[i].first][ind] = 1;
		if (i != 0)
			has[parent[que[i].first].first] |=
has[que[i].first];
		if (has[que[i].first].count() > strNum / 2) {
			if (que[i].second > best)
				bestNode.clear(), best = que[i].second;
			if (que[i].second == best)
				bestNode.push_back(que[i].first);
```

```
            }
      }
}
void printPrefix(int node, string &s) {
      if (node != 0) {
            printPrefix(parent[node].first, s);
            iter it = g.find(parent[node]);
            for (int i = it->second.s; i <= it->second.e; i++)
                  s += str[i];
      }
}
```

KMP

```
//put strings in the arrays and call match, result in vector res
#define MAX 1000010
char pat[MAX];
char text[MAX];
int pre[MAX];
vector<int> res;
//int dp[MAX][256];
int getlen(int l, char c) {
      //if(dp[l][c]!=-1)return dp[l][c];
      while (l && pat[l] != c)
            l = pre[l - 1];
      if (pat[l] == c)
            l++;

      return /*dp[l][c]=*/l;
}
/* To check if the prefix ending at i is periodic or not use this
if sttmnt
      d = prefix_len / period_length_in_it
      which means d is the number of repetitions of a substring
forming the prefix ending @i
if ((i + 1) % (i + 1 - pre[i]) == 0)
            int d = (i + 1) / (i + 1 - pre[i]);
*/
void compute_pre() {
      pre[0] = 0;
      int l = 0;
      if (pat[0])
            for (int i = 1; pat[i]; ++i) {
                  l = getlen(l, pat[i]);
                  pre[i] = l;
            }
}
void match() {
      compute_pre();
      res.clear();
      int l = 0;
      for (int i = 0; text[i]; ++i) {
```

```
            l = getlen(l, text[i]);
            if (!pat[l]) {
                  res.push_back(i - l + 1);
                  l = pre[l - 1]; //l=0 if no overlap is allowed
            }
      }
}
```

Rabin Karp

V1 //coach says I suck

```
//ll md[3] = {2000000063, 2000000087, 2000000089};
//ll mdi[3] = {622568113, 661478628, 1712062333};
//ll bs = 257; ll md=2147483629;
ll bs = 53;
ll mdi = 1053482535;
ll pow(ll n, ll p) {
      if (p == 0)
            return 1;
      ll t = pow(n, p / 2) % md;
      if (p % 2)
            return (((t * t) % md) * n) % md;
      return ((t * t) % md);
}
ll addDigit(ll n, ll val, ll ind) {
      ll temp = (pow(bs, ind) * val) % md;
      return (n + temp) % md;
}
ll shiftLeft(ll n) {
      return (n * bs) % md;
}
ll shiftRight(ll n) {
      return (n * mdi) % md;
}
ll removeDigit(ll n, ll val, ll ind) {
      ll temp = (pow(bs, ind) * val) % md;
      return (n + md - temp) % md;
}
```

V2

```
#define BASE 128LL
#define BASEINV 1453125008LL
#define MOD 2000000011LL
ll addCharAt(int ind, char v, ll pvHashV) {
      return ((pow(BASE, (ll) ind) * v) % MOD + pvHashV) % MOD;
}
ll removeCharAt(int ind, char v, ll pvHashV) {
      return (MOD - (((pow(BASE, (ll) ind) * v) % MOD)) % MOD +
pvHashV) % MOD;
}
ll shiftL(ll pvHash) {
```

```
	return (pvHash * BASE) % MOD;
}
ll shiftR(ll pvHash) {
	return (pvHash * BASEINV) % MOD;
}
```

Aho

```
#define MOD 1000000009
typedef hash_map<pair<int, char> , int, hashing>::iterator
trieitr;
struct node {
	vector<char> children;
	int fail;
	vector<int> mpind;
};

hash_map<pair<int, char> , int, hashing> trie;
vector<node> nodes;

void insert(const char* str, int sid) {
	int cur = 0;
	for (; *str; str++) {
		trieitr itr = trie.find(make_pair(cur, *str));
		int nind;
		if (itr == trie.end()) {
			nind = nodes.size();
			nodes[cur].children.push_back(*str);
			nodes.push_back(node());
			trie[make_pair(cur, *str)] = nind;
		} else
			nind = itr->second;
		cur = nind;
	}
	nodes[cur].mpind.push_back(sid);
}
int dp2[302][128];
int getnode(int node, char c) {
	int&x = dp2[node][c];
	if (x != -1)
		return x;
	while (trie.find(make_pair(node, c)) == trie.end())
		node = nodes[node].fail;
	return x = trie[make_pair(node, c)];
}

void build_failure() {
	queue<int> q;
	//loop over your alphabet
	for (char c = 'a'; c <= 'z'; c++) {
		trieitr itr = trie.find(make_pair(0, c));
		if (itr == trie.end())
```

```
                trie[make_pair(0, c)] = 0;
            else {
                q.push(itr->second);
                nodes[itr->second].fail = 0;
            }
        }
        while (!q.empty()) {
            int cur = q.front();
            q.pop();
            node& n = nodes[cur];
            for (int i = 0; i < n.children.size(); i++) {
                char ch = n.children[i];
                trieitr itr = trie.find(make_pair(cur, ch));
                int next = itr->second;
                nodes[next].fail = getnode(nodes[cur].fail, ch);
                node& nn = nodes[nodes[next].fail];
        nodes[next].mpind.insert(nodes[next].mpind.end(),
nn.mpind.begin(),nn.mpind.end());
                q.push(next);
            }
        }
}

void build(vector<string>& vs) {
        trie.clear();
        nodes.clear();
        nodes.push_back(node());
        for (int i = 0; i < vs.size(); i++)
            insert(vs[i].c_str(), i);
        build_failure();
}
```

Treap

```
struct node {
        node *left, *right;
        int value, freq, priority, size;
        static node* sentinel;
        node() {
            memset(this, 0, sizeof *this); //initialize all member
variables to zero }
            node(int v) {
                value = v; freq = size = 1;
                priority = rand(); left = right = sentinel;}
            void update() {
                size = freq + left->size + right->size;
            }
        }
        ;
        node* node::sentinel = new node();
        node* rotateRight(node* Q) {
            node* P = Q->left;
```

```
        Q->left = P->right;
        P->right = Q;
        Q->update();
        P->update();
        return P;
    }
    node* rotateLeft(node* P) {
        node* Q = P->right;
        P->right = Q->left;
        Q->left = P;
        P->update();
        Q->update();
        return Q;
    }
    node* balance(node* root) {
        if (root->left->priority > root->priority)
            root = rotateRight(root);
        else if (root->right->priority > root->priority)
            root = rotateLeft(root);
        return root;
    }
    node* insert(node* root, int val) {
        if (root == node::sentinel)
            return new node(val);
        if (val == root->value) {
            root->freq++;
            root->size++;
            return root;
        }
        if (val < root->value)
            root->left = insert(root->left, val);
        else
            root->right = insert(root->right, val);
        root->update();
        root = balance(root);
        return root;
    }
    int lower_bound(node* root, int x) { //number of elements
less than x in the tree
        if (root == node::sentinel)
            return 0;
        if (x == root->value)
            return root->left->size;
        return (x < root->value) ? lower_bound(root->left, x)
                    : root->left->size + root->freq +
lower_bound(root->right, x);
    }
    node* remove(node* root, int v) {
        if (root == node::sentinel)
            return root;
        if (v < root->value)
            root->left = remove(root->left, v);
```

```
            else if (v > root->value)
                  root->right = remove(root->right, v);
            else {
                  if (root->freq > 1) {
                        root->freq--;
                        root->size--;
                        return root;
                  }
                  if (root->left == node::sentinel)
                        root = root->right;
                  else if (root->right == node::sentinel)
                        root = root->left;
                  else {
                        if (root->left->priority < root->right-
>priority)
                              root = rotateRight(root);
                        else
                              root = rotateLeft(root);
                        root = remove(root, v);
                  }
            }
            root->update();
            return root;
      }
      int upper_bound(node* root, int x) {
            //number of elements less than or equal to x in the
tree
            if (root == node::sentinel)
                  return 0;
            if (x == root->value)
                  return root->left->size + root->freq;
            return (x < root->value) ? upper_bound(root->left, x)
                        : root->left->size + root->freq +
upper_bound(root->right, x);
      }
      int getByIndex(node* root, int idx) {
            if (idx < root->left->size)
                  return getByIndex(root->left, idx);
            if (idx >= root->left->size + root->freq)
                  return getByIndex(root->right,
                              idx - (root->left->size + root-
>freq));
            return root->value;
      }
};
```

KD-Tree

```
#define Type long long
#define DIMS 3

struct point {
```

```
	Type a[DIMS];point(Type aa, Type bb, Type cc) {
		a[0] = aa;
		a[1] = bb;
		a[2] = cc;
	}
	point() {
	}
	bool operator <(const point& aa) const {
		return a[0] < aa.a[0] || (a[0] == aa.a[0] && a[1] <
aa.a[1]) || (a[0]
				== aa.a[0] && a[1] == aa.a[1] && a[2] <
aa.a[2]);
	}
};
set<point> ss;
vector<point> v;

struct node;
node* nil;
struct node {
	Type di[DIMS];
	node*l, *r;
	node() :
		l(nil), r(nil) {
	}
	node(Type a, Type b, Type c, node*left, node*right) {
		di[0] = a;
		di[1] = b;
		di[2] = c;
		l = left;
		r = right;
	}
};

int n;

struct cmp {
	static int d;
	bool operator()(const point&a, const point&b) const {
		return a.a[d] < b.a[d];
	}
};
int cmp::d = 0;

node* build(int st, int en, int depth) {
	if (en < st)
		return nil;
	if (en == st)
		return new node(v[st].a[0], v[st].a[1], v[st].a[2],
nil, nil);
	cmp::d = depth % DIMS;
	sort(v.begin() + st, v.begin() + en + 1, cmp());
```

```
	int med = (en + st) / 2;
	node*r = new node();
	r->di[0] = v[med].a[0];
	r->di[1] = v[med].a[1];
	r->di[2] = v[med].a[2];
	r->l = build(st, med - 1, depth + 1);
	r->r = build(med + 1, en, depth + 1);
	return r;
}
point p;

Type distSq(node*cur) {
	Type r = 0;
	for (int i = 0; i < DIMS; ++i)
		r += (cur->di[i] - p.a[i]) * (cur->di[i] - p.a[i]);
	return r;
}
Type mc;
//finds nearest neighbour to point p
void dfs(node*cur, Type& mn, int depth) {
	if (cur == nil)
		return;
	Type d = distSq(cur);
	if (d == mn)
		mc++;
	if (d < mn && !(cur->di[0] == p.a[0] && cur->di[1] ==
p.a[1] && cur->di[2]
			== p.a[2]))
		mn = d, mc = 1;
	int di = depth % DIMS;
	if (cur->di[di] > p.a[di]) {
		dfs(cur->l, mn, depth + 1);
		if (mn < (cur->di[di] - p.a[di]) * (cur->di[di] -
p.a[di]))
			return;
		dfs(cur->r, mn, depth + 1);
	} else {
		dfs(cur->r, mn, depth + 1);
		if (mn < (cur->di[di] - p.a[di]) * (cur->di[di] -
p.a[di]))
			return;
		dfs(cur->l, mn, depth + 1);
	}
}
```

Letter tree

```
const int MAX = 128; //if MAX is big range but not all values are
used, use a map instead of static array
struct tree {
	tree* child[MAX];
	tree() {
```

```
            memset(child, 0, sizeof(child));
      }
      void insert(vector<int>& vec, int index) {
            if (index == vec.size())
                  return;
            if (child[vec[index]] == 0)
                  child[vec[index]] = new tree();
            child[vec[index]]->insert(vec, index + 1);
      }
      int count() {
            int c = 0;
            for (int i = 0; i < MAX; i++)
                  if (child[i] != 0)
                        c += child[i]->count();
            return c + 1;
      }
};
//MAIN
//tree t;
//t.insert(vec, 0)
//int c = t.count
```

Letter Tree(Hashing)

```
edge edges[maxE]; // memseted with -1
bool isLeaf [maxN];//memseted with 0

edge& getEdge(int ind,unsigned char c)
{
        int i = ((ind<<8)+c)%maxE; // da el hashing
        while(edges[i].from!=-1)
        {
          if(edges[i].from == ind && edges[i].c == c) break;
          i = ++i%maxE;
        }
        return edges[i];
}

void insert(const char* str,int ind = 0)
{
        if(!*str)
        {
                isLeaf[ind] = true;
                return ;
        }

        edge& e = getEdge(ind,*str);
        if(e.from == -1)
        {
                e.from = ind;
                e.to = cN ++;
                e.c = *str;
```

```
        }
        insert(str+1,e.to);
}

bool traverse(const char* str,int ind = 0)
{
        if(!*str)
                return isLeaf[ind] ;

        edge& e = getEdge(ind,*str);
        if(e.from == -1)
                return false;
        return traverse(str+1,e.to);
}
```

Letter Tree(Hashing-using hashmap)

```
#include<ext/hash_map>
using namespace __gnu_cxx;
//#define MAXNODES 1000000
int nNodes = 1; // root
struct hashh
{
      int operator()(const pair<int,char> &p)const
      {
            return p.first*31+p.second;
      }
};

hash_map<pair<int,char>,int ,hashh> edges; //from , char, to
//bool isLeaf[MAXNODES];
vector<bool> isLeaf(1);
vector<multiset<int> > crab; // fot dfs implementations

void insert(const char* str)
{
      int cur = 0;
      for(const char* s = str ; *s ; s++)
      {
            hash_map<pair<int,char>,int ,hashh>::iterator it;
            it = edges.find(make_pair(cur,*s-'a'));
            if(it==edges.end())
      isLeaf.push_back(0),cur = edges[make_pair(cur,*s-'a')]=
nNodes++;
            else
                  cur = it->second;
      }
      isLeaf[cur]=1;
}

bool find(const char* str)
```

```
{
	int cur = 0;
	for(const char* s = str ; *s ; s++)
	{
		hash_map<pair<int,char>,int ,hashh>::iterator it;
		it = edges.find(make_pair(cur,*s-'a'));
		if(it==edges.end())
			return false;
		else
			cur = it->second;
	}
	return isLeaf[cur];
}

// dfs on the tree
int dfs(int cur)
{
	hash_map<pair<int,char>,int ,hashh>::iterator it;
	int ret = -(1<<28);
	if(isLeaf[cur])ret = 0;
	for(int c=0;c<26;c++)
	{
		it=edges.find(make_pair(cur,c));
		if(it==edges.end())
			continue;
		if(crab[c].size())
		{
			int x = *crab[c].rbegin();
			crab[c].erase(--crab[c].end());
			ret = max(ret,x+dfs(it->second));
			crab[c].insert(x);
		}
	}
	return ret;
}
```

Quad Tree

```
/*Very useful in many cases. One case is compressing of binary
image.
Imagine we partition a grid into 4 sections, if there is a region
full of same color [0, 1]
we do not need to process that region again.*/
struct QuadTree {
        bool isMixed;
        int val;
        QuadTree* childs[4];
        QuadTree() : isMixed(1) {}
        QuadTree(int v) : isMixed(0), val(v) {}
        QuadTree* getChild(int i) {
            if(isMixed) return child[i];
            return this;
```

```
        }
};
/*In comparing 2 trees, one tree may not have same structure as
second one.
One nice trick to make them seems similar, is define get
function.*/
```

Mathematically oriented part

Extended GCD

```
//ax+by=gcd(a,b)
int eGCD(int a, int b, int &x, int &y) {
      x = 1;
      y = 0;
      int nx = 0, ny = 1;
      int t, r;
      while (b) {
            r = a / b;
            t = a - r * b;
            a = b;
            b = t;
            t = x - r * nx;
            x = nx;
            nx = t;
            t = y - r * ny;
            y = ny;
            ny = t;
      }
      return a;
}
//ax+by=c
bool solveLDE(int a, int b, int c, int &x, int &y, int &g) {
      g = eGCD(a, b, x, y);
      x *= c / g;
      y *= c / g;
      return (c % g) == 0;
}
//(a*mi)%m=1
int modInv(int a, int m) {
      int mi, r;
      eGCD(a, m, mi, r);
      return (mi + m) % m;
}
//(a*x)%b=c
bool solve(ll a, ll b, ll c, ll &x) {
      ll y, g;
      if (solveLDE(a, b, c, x, y, g) && a * x + b * y == c) {
            if (x < 0) {
                  x += (abs(x) / b) * b;
                  if (x < 0)
```

```
                        x += b;
                }
                return 1;
        }
        return 0;
}
```

modInv (using eGCD)

```
//(a*mi)%m=1
int modInv(int a, int m) {
        int mi, r;
        eGCD(a, m, mi, r);
        return (mi + m) % m;
}
```

modInv (using fast power)

```
long long PowerMod(int x, int p, int m) {
        if (!p)
                return 1;
        long long y = PowerMod(x, p >> 1, m);
        y = y%m;
        y = (y * y) % m;
        if (p & 1)
                y = (y * x) % m;
        return y;
}

long long modInv(int x, int m) {
        if (__gcd(x, m) != 1)
                throw string("not coprimes ya 7aywan");
        return PowerMod(x, m - 2, m);
}
```

Chinese remainder theorem

```
ll EGCD(ll r0, ll r1, ll &x0, ll &y0) {
        ll y1 = x0 = 1, x1 = y0 = 0;
        while (r1) {
                ll r = (r0 / r1), t;
                t = r0 - r * r1, r0 = r1, r1 = t;
                t = x0 - r * x1, x0 = x1, x1 = t;
                t = y0 - r * y1, y0 = y1, y1 = t;
        }
        return r0;
}
bool solveLDE(ll a, ll b, ll c, ll &x, ll &y, ll &g) {
        g = EGCD(a, b, x, y);
        ll m = c / g;
        x *= m;
        y *= m;
        return !(c % g);
}
```

```
bool CRT(const vector<ll> &r, const vector<ll> &m, ll &R, ll &M)
{
	R = r[0], M = m[0];
	for (int i = 1; i < (int) r.size(); i++) {
		ll m1 = M, m2 = m[i], r1 = R, r2 = r[i];
		ll p1, p2, g;
		if (!solveLDE(m1, m2, r2 - r1, p1, p2, g))
			return false;
		ll mod = m2 / g;
		p1 = ((p1 % mod) + mod) % mod;
		M = m1 / g * m2;
		R = (p1 * m1) % M + r1;
		R = ((R % M) + M) % M;
	}
	return 1;
}
```

Euler Toitent

```
int fi(int n) {
	int result = n;
	for (int i = 2; i * i <= n; i++) {
		if (n % i == 0)
			result -= result / i;
		while (n % i == 0)
			n /= i;
	}
	if (n > 1)
		result -= result / n;
	return result;

}
```

Modular Linear Equation Solver

```
//returns the solutions (values of x) to the equation ax=b mod n.
(n //must be positive)
vector<int> modularLinearEquationSolver(int a, int b, int n){
	vector<int> result;
	triple t = extendedEuclid(a,n);
	if(b%t.d == 0){ //if d|b
		//we have solutions
		int x0 = (t.x*(b/t.y)) % n;
		while(x0 < 0)x0+=n;
		for(int i = 0 ; i < t.d ; i++){
			int s = (x0 + i*(n/t.d)) % n;
			while(s < 0)s+=n;
			result.push_back(s);
		}
	}
	return result;
}
```

Farey genrate all fractions On pairs with num. and dum. less than n

```
//sorted
void farey(int n) {
// genrate all fractions On pairs with num. and dum. less than n sorted
	int a = 0, b = 1, c = 1, d = n;
	v.push_back(make_pair(a, b));
	while (c < n) {
		int k = int((n + b) / d), ob = b, oa = a;
		a = c, b = d, c = k * c - oa, d = k * d - ob;
		v.push_back(make_pair(a, b));
	}
}
```

Continued Fractions of Rationales x=a0+(1/ (a1 + (1 / (a2 + ...)))

```
//where ai is positive integer
vector<int> contFract(int m, int n) {
	vector<int> ans;
	while (n) {
		ans.push_back(m / n);
		m %= n;
		m ^= n ^= m ^= n;
	}
	return ans;
}
```

Catalan numbers The number of distinct binary trees of n nodes

```
long long catalan(int n) {
	return (n == 1) ?
			1 :
			((2 * (n - 1) + 2) * (2 * (n - 1) + 1) *
catalan(n - 1))
				/ ((n) * (n + 1));
}
```

Prime Factors & Divisors

```
void factorize(int n, vector<pair<int, int> > &result)
// n to get it's prime we byrga3 vector of pair for all numbers
{
	result.clear();
	int i, d = 1;
	for (i = 2; i * i <= n; i += d, d = 2) {
		if (n % i == 0)
			result.push_back(make_pair(i, 0));
		while (n % i == 0) {
			n /= i;
			result.back().second++;
		}
	}
	if (n != 1)
		result.push_back(make_pair(n, 1));
```

```
	return result;
} // worst square root(n)

vector<pair<int, int> > primeFactors;
vector<int> divisors;
void getDivisors2(int i, int d) { // index , divisor till now // number
of divisors = powers+1 * b3d
	if (i == primeFactors.size()) {
		divisors.push_back(d);
		return;
	}
	for (int j = 0; j <= primeFactors[i].second; j++) {
		getDivisors2(i + 1, d);
		d *= primeFactors[i].first;
	}
}
```

Factorize to divisors using folding

```
long long FastPower(long long a, long long n){
	if (n==0) return 1;
	long long x=FastPower(a,n/2);
	if (n%2==0) return x*x;
	return x*x*a;
}

void Factorize(long long m,vector<long long>& p,vector<long
long>& q){//m=p1^q1*p2^q2..,np=size of p,q (1-indexed)
	long long a=m; long long i=2;
	while((unsigned long long)i*i<=(unsigned long long)a) {
		if (a%i==0) {
			p.push_back(i); q.push_back(1); a/=i;
			while(a%i==0) {q.back()++; a/=i;}
		}
		i+=i==2?1:2;
	}
	if (a>1) { p.push_back(a); q.push_back(1);}
}

long long numOfDiv(long long n, vector<long long>& p, vector<long
long>& q) {

	Factorize(n, p, q);
	long long r = 1;
	foreach(i,q)
		r *= (*i) + 1;
	return r;
}

long long numOfDiv(long long n) {
	vector<long long> p;
	vector<long long> q;
```

```
	return numOfDiv(n, p, q);
}

vector<long long>& getDivisors(long long n, vector<long long>&
res,
		vector<long long>& p, vector<long long> &q) {
	res.resize(0);
	long long r = numOfDiv(n, p, q);
	for (int i = 0; i < (int) (r); ++i) {
		long long d = 1;
		long long a = i;
		for (int j = 0; j < (int) (q.size()); ++j) {
			long long x = a % (q[j] + 1);
			a /= (q[j] + 1);
			d *= FastPower(p[j], x);
		}
		res.push_back(d);
	}
	sort(res.begin(), res.end());
	return res;
}

vector<long long>& getDivisors(long long n, vector<long long>&
res) {
	vector<long long> p;
	vector<long long> q;
	return getDivisors(n, res, p, q);
}
```

Sieve

```
void sieve(bool prime[], int N) {
//determines if numbers between 0 and N-1 are prime or not
	memset(prime, -1, N * sizeof(prime[0]));
	prime[0] = prime[1] = false;
	int sqrtN = (int) sqrt((double) N);
	for (int i = 2; i <= sqrtN; i++)
		if (prime[i])
			for (int j = i * i; j < N; j += i)
				prime[j] = false;
}
```

nCk

```
// Use it with large values but small difference < 10e6 (take care
OVERFLOW)
unsigned long long nCr(unsigned long long n, unsigned long long r) {
	if (n == r)
		return 1;
	return nCr(n - 1, r) * n / (n - r);
}
```

Recursive combinations O(N^2)

```
//based on pascal's formula
const int MAX = 31;
int comb[MAX][MAX];
void calcCombinations() {
	comb[0][0] = 1;
	for (int i = 1; i <= MAX; i++) {
		comb[i][0] = 1;
		comb[i][i] = 1;
		for (int j = 1; j < i; j++)
			comb[i][j] = comb[i - 1][j] + comb[i - 1][j - 1];
	}
}
```

Efficient combinations

```
//divides by the gcd before multiplication
long gcd(long a, long b) {
	if (a % b == 0)
		return b;
	else
		return gcd(b, a % b);
}
void Divbygcd(long& a, long& b) {
	long g = gcd(a, b);
	a /= g;
	b /= g;
}
long C(int n, int k) {

	if (n < k)
		return 0;
	long numerator = 1, denominator = 1, toMul, toDiv, i;
	if (k > n / 2)
		k = n - k; // use smaller k
	for (i = k; i; i--) {
		toMul = n - k + i;
		toDiv = i;
		Divbygcd(toMul, toDiv); // always divide before
multiply
		Divbygcd(numerator, toDiv);
		Divbygcd(toMul, denominator);
		numerator *= toMul;
		denominator *= toDiv;
	}
	return numerator / denominator;
}
```

-ve Base Conversion

```
string ConvertToNegativeBase(int x, int b) {
//abs(b) is between 2, 10
	bool sign = false;
```

```
	if (b > 0 && x < 0) {
		sign = true;
		x = abs(x);
	}
	string str = get(x, b);
	if (sign)
		str = "-" + str;
	return str;
}
```

System Of Linear Equation Moded Top of Form

```
long long mod(long long a, long long m)
{
    a%m;
    if(a<0) a+=m;
    return a%m;
}
const int MAXSIZE = 100;
int modNumber;
int size;
long long rhs[MAXSIZE], lhs[MAXSIZE][MAXSIZE];

void print()
{
    for(int i = 0 ; i < size ; i ++)
    {
        for(int j = 0 ; j < size ; j ++)
        {
            cout<<lhs[i][j]<<" ";
        }
        cout<<rhs[i];
        cout<<endl;
    }
}
void solveSystemOfLinearEquationModed()
{
        for(int i = 0 ; i<size ; i++)
    {
                long long mInv = modInv(lhs[i][i] , modNumber);
                rhs[i] = (((rhs[i]%modNumber) *
(mInv%modNumber))%modNumber);

                for(int j = 0 ; j<size ; j++)
                        lhs[i][j] = (((lhs[i][j]%modNumber) *
(mInv%modNumber))%modNumber);
                 //print();
        for(int j = 0 ; j< size ; j++)
        {
                        if(j == i) continue;

            long long adI = modNumber - lhs[j][i];
```

```
            rhs[j] = ((rhs[j] % modNumber) + ((rhs[i] %
modNumber) * (adI % modNumber) )% modNumber)%modNumber;

            for(int k = 0 ; k<size ;  k++)
                lhs[j][k] = ((lhs[j][k] % modNumber) +
((lhs[i][k] % modNumber) * (adI % modNumber) )%
modNumber)%modNumber;
          // print();
                }
        }
}
```

Solve System of Linear Equations (Gaussian)

```
#define rep(i,n) for(int  i=0;i<(int)(n);++i)
long double ZERO=0;
const long double INF=1/ZERO,EPSILON=1e-12;
#define rep2(i,a,b) for(int i=(a);i<=((int)b);++i)
#define
foreach(it,c)for(__typeof((c).begin())it=(c).begin();it!=(c).end();++it)
#define sz(v) ((int)((v).size()))

enum SOL {
      NOSOL, UNIQUE, INFINIT
};

bool zeros(vector<long double> & row) {
      foreach(i,row)
      {
            if(fabs(*i-0)>EPSILON)
            return false;
      }
      return true;
}
bool swapRow(int i, vector<vector<long double> > &mat) {
      rep2(j,i+1,mat.size()-1) {
            if (fabs(mat[j][i] - 0) > EPSILON) {
                  swap(mat[i], mat[j]);
                  return true;
            }
      }
      return false;
}
void devideRow(int i, long double x, vector<vector<long double> > &mat) {
      foreach(j, mat[i]) * j /= x;
}
void makeZero(int i, int j, vector<vector<long double> > &mat) {
      long double d = -mat[j][i];
      rep(k,mat[i].size()) {
            mat[j][k] += d * mat[i][k];
      }
}
```

```
SOL solveLinearEquation(vector<vector<long double> > &mat) {
	rep(i,mat.size()) {
		if (zeros(mat[i])) {
			mat.erase(mat.begin() + i);
			i--;
			continue;
		}
		if (i == (int) mat[i].size())
			break;
		if (fabs(mat[i][i] - 0) < EPSILON)
			if (!swapRow(i, mat))
				return NOSOL;
		devideRow(i, mat[i][i], mat);
		rep(j,mat.size()) {
			if (i == j)
				continue;
			makeZero(i, j, mat);
		}
	}
	if (mat.size() + 1 == mat[0].size())
		return UNIQUE;
	if (mat.size() + 1 < mat[0].size())
		return INFINIT;
	return NOSOL;
}
```

Matrix Power

```
// very optimized code
#define MSIZE 101
int MOD;
struct matrix
{
	long long m[MSIZE][MSIZE];
	int r,c;

	matrix(int r_,int c_ ){
		r = r_,c = c_;
	}


};

// good code
void mul(const matrix&a,const matrix&b,matrix&c)
{
	for(int i=0;i<a.r;i++)
		for(int j=0;j<b.c;j++)
		{
			c.m[i][j]=0;
			for(int k=0;k<a.c;k++)
```

```
      c.m[i][j]+=((a.m[i][k]%MOD)*(b.m[k][j]%MOD))%MOD,c.m[i][j]%
=MOD;

            }
      c.r=a.r;
      c.c=b.c;
}
void mp(const matrix &a,const long long &p,matrix& res)
{
      int x=log(p)/log(2)+1+1e-9;
      matrix t(a.r,a.c),*t2=&t,*t1=&res;
      res.r=res.c=a.r;
      for(int i=0;i<res.c;i++)
            for(int j=0;j<res.c;j++)
                  res.m[i][j]=i==j;
      for(;x>=0;x--)
      {
            mul(*t1,*t1,*t2);
            swap(t1,t2);
            if(p&(1ll<<x))
            {
                  mul(*t1,a,*t2);
                  swap(t1,t2);
            }

      }
      res=*t1;
}
```

Integer roots for polynomial given coefficients

```
#define big long long
big a[100000];// the polynomial coefficients, a[0] is the
coefficient of the constant term
int n; //the polynomial degree
big MAX_COEFFICIENT; // the largest possible absolute value of a
coefficient
bool check(big x) {
      big d = 0;
      for (int i = n; i >= 0; i--) {
            d = d * x + a[i];
            if (abs(x) != 1 && abs(d) > 2 * MAX_COEFFICIENT)
                  return false;
      }
      return d == 0;
}
set<big> getIntegerRoots() {
      set<big> res;
      if (a[0] == 0)
            res.insert(0);
      int f = 0;
```

```
	while (a[f] == 0)
		f++; //specify constant term of the polynomial
	set<big> div;
	div = divisors(abs(a[f]));
//divisors of constant term, these are the possible roots
	vector<big> vv(div.begin(), div.end());
	for (int i = 0; i < vv.size(); i++) {
		if (check(vv[i]))
			res.insert(vv[i]);
		if (check(-vv[i]))
			res.insert(-vv[i]);
	}
	return res;
}

//MAIN
//Set a, n, MAX_COEFFICIENT
set<big> roots = getIntegerRoots();
```

Prime power in !N

```
long long count_p_in_nfact(long long p, long long n) {
	long long res = 0;
	long long q = p;
	while (q <= n) {
		res += n / q;
		q *= p;
	}
	return res;
}
```

Numerical Integration

Simpsons

```
template<class T>
long double simpson(long double(*f)(T data, const long double&x),
T& d,
		long double a, long double b, int n = 100) {
	long double h = (b - a) / n;
	long double h2 = 0.5 * h;
	long double bound = a + (n - 0.25) * h;
	long double integral = (*f)(d, a) + 4.0 * (*f)(d, a + h2);

	for (a += h; a < bound; a += h)
		integral += 2.0 * (*f)(d, a) + 4.0 * (*f)(d, a + h2);

	return h * (integral + (*f)(d, a)) / 6;
}
```

Adaptive Simpsons

```
//Adaptive Simpson works if there is no horizontal lines in the
curve
```

```
double adaptiveSimpsonsAux(double(*f)(const double&), double a,
double b,
            double epsilon, double S, double fa, double fb, double
fc, int bottom) {
            double c = (a + b) / 2, h = b - a;
      double d = (a + c) / 2, e = (c + b) / 2;
      double fd = f(d), fe = f(e);
      double Sleft = (h / 12) * (fa + 4 * fd + fc);
      double Sright = (h / 12) * (fc + 4 * fe + fb);
      double S2 = Sleft + Sright;
      if (bottom <= 0 || fabs(S2 - S) <= 15 * epsilon)
            return S2 + (S2 - S) / 15;
      return adaptiveSimpsonsAux(f, a, c, epsilon / 2, Sleft, fa,
fc, fd,
                  bottom - 1) + adaptiveSimpsonsAux(f, c, b,
epsilon / 2, Sright, fc,
                  fb, fe, bottom - 1);
}

//
// Adaptive Simpson's Rule
//
double adaptiveSimpsons(double(*f)(const double&), // ptr to
function
            double a, double b, // interval [a,b]
            double epsilon, // error tolerance
            int maxRecursionDepth) { // recursion cap
      double c = (a + b) / 2, h = b - a;
      double fa = f(a), fb = f(b), fc = f(c);
      double S = (h / 6) * (fa + 4 * fc + fb);
      return adaptiveSimpsonsAux(f, a, b, epsilon, S, fa, fb, fc,
                  maxRecursionDepth);
}
```

Simplex

```
/*
 Simplex algorithm for solving linear programming problems.
 O(N^3), where N is the number of variables

 Testing Field: TopCoder(PreciousStones,Mixture), UVA(10498)
 References:
 -
http://en.wikibooks.org/wiki/Operations_Research/The_Simplex_Meth
od
 */

#include<cmath>
#include<vector>
using namespace std;

enum Type {
```

```
	LE, GE, EQ
}; //respectively, less than or equal, greater than or equal,
equal.
enum Result {
	OK, UNBOUND, UNFEASIBLE
};
enum OFType {
	MAX, MIN
}; //objective funtion type (maximize or minimize)

#define INF 1e30
#define EPS 1e-9
#define LD	long double	//Percision does matter in this
algorithm
struct SimplexModel {

	/*****Data Structures*****/
	//Constraints
	vector<vector<LD> > lhs; //matrix of constraints
coefficients
	vector<LD> rhs; //right hand side of constraints
	vector<Type> constraintTypes; //type of constraint (greater
than or equal, equal ... etc)

	//Objective Function
	vector<LD> of; //coefficients of variables in objective
function
	OFType oftype;

	//Variables
	vector<bool> unRestricted; //unRestricted[i] is true iff
variable[i] can be -ve

	//Values of variable in the solution (output only, don't
fill)
	vector<LD> solution;

	//Internal use data structures (don't fill from outside)
	int nVar, nCon; //number of variables/constraints
	vector<int> negativePart; //index of negative part of
unrestricted variables
	vector<int> positivePart; //index of positive part of
unrestricted variables
	vector<bool> isNegativePart; //isNegativePart[i] = true iff
variable i is x2 in (x=x1-x2)
	vector<int> basic; //indicies of variables in the current
solution (initially slacks and artificials)
	vector<bool> isArtificial; //isArtificial[i] = true iff
variable[i] is artificial

	/*****Data Structures*****/
```

```
	/******Methods******/

	//Add new variable to the model (used to add slacks,
artificials, negative parts and surpluses) and return its index
	int addVariable() {

		//Add variable to LHS
		for (int i = 0; i < lhs.size(); i++)
			lhs[i].push_back(0);

		//Add varaible to Objective funtion
		of.push_back(0);
		isArtificial.push_back(false); //default value, might
be modified later
		isNegativePart.push_back(false); //default value,
might be modified later
		positivePart.push_back(0);
		//Return variable index
		return nVar++;
	}

	//Standardize model
	void standardize() {

		//Initialize internal data structures
		nVar = unRestricted.size();
		nCon = lhs.size();
		negativePart.resize(nVar);
		positivePart.resize(nVar);
		isNegativePart.clear();
		isNegativePart.resize(nVar, false);
		basic.clear();
		solution.clear();
		solution.resize(nVar, 0);
		isArtificial.clear();
		isArtificial.resize(nVar, false);

		int i, j, varIdx;

		//Objective function should be max
		if (oftype == MIN) {
			for (i = 0; i < nVar; i++)
				of[i] *= -1;
			oftype = MAX;
		}

		//Handle unresitricted variables (set x to x1-x2)
		for (i = 0; i < unRestricted.size(); i++) {
			if (!unRestricted[i])
				continue;
			varIdx = addVariable();
			for (j = 0; j < nCon; j++)
```

```
                    lhs[j][varIdx] = -lhs[j][i];
                of[varIdx] = -of[i];
                negativePart[i] = varIdx;
                positivePart[varIdx] = i;
                isNegativePart[varIdx] = true;
            }

            //Standardize contstraints
            for (i = 0; i < nCon; i++) {

                if (rhs[i] < 0) {
                    rhs[i] *= -1;
                    for (j = 0; j < nVar; j++)
                        lhs[i][j] *= -1;
                    if (constraintTypes[i] != EQ)
                        constraintTypes[i] =
constraintTypes[i] == GE ? LE : GE; //modify GE to LE and vice
versa
                }

                //Add basic variable (variable in the initial
solution, that is slack or artifical)
                int basicVar = addVariable();
                basic.push_back(basicVar);
                lhs[i][basicVar] = 1;

                switch (constraintTypes[i]) {
                case GE:
                    varIdx = addVariable(); //add surplus
                    lhs[i][varIdx] = -1;
                case EQ:
                    isArtificial[basicVar] = true;
                    of[basicVar] = -INF;
                    break;
                }
                constraintTypes[i] = EQ;
            }
        }

        //Solve model using Simplex algorithm
        Result solve() {

            //Standardize
            standardize();

            //Solve
            int i, j, k;
            LD z, ratio, cmz;

            while (true) {
                //Compute z, c-z and Select pivot column
                int pivotCol = 0;
```

```
                LD bestCMZ = -INF;
                for (j = 0; j < nVar; j++) {
                      z = k = 0;
                      for (i = 0; i < basic.size(); i++)
                            z += of[basic[i]] * lhs[k++][j];
                      cmz = of[j] - z;

                      pivotCol = (cmz > bestCMZ) ? j : pivotCol;
                      bestCMZ = max(cmz, bestCMZ);
                }

                //Check if no more improvement
                if (fabs(bestCMZ) < EPS)
                      break;

                //Compute ratio and Select pivot row
                int pivotRow = 0;
                LD bestRatio = INF;
                for (i = 0; i < nCon; i++) {
                      if (lhs[i][pivotCol] < EPS)
                            continue; //avoid division by zero
                      ratio = rhs[i] / lhs[i][pivotCol];
                      if (ratio < 0)
                            ratio = INF; //to avoid selecting
negative ratios

                      pivotRow = ratio < bestRatio ? i :
pivotRow;
                      bestRatio = min(bestRatio, ratio);
                }

                if (bestRatio >= INF)
                      return UNBOUND; //unbounded solution (can
achieve infinite profit)

                //Update table
                basic[pivotRow] = pivotCol;

                //Set coeff of new basic to 1
                LD pivot = lhs[pivotRow][pivotCol];
                for (i = 0; i < nVar; i++)
                      lhs[pivotRow][i] /= pivot;
                rhs[pivotRow] /= pivot;

                //Set coeff of pivotCol to 0
                for (i = 0; i < nCon; i++) {
                      if (i == pivotRow)
                            continue;
                      LD val = -lhs[i][pivotCol];
                      for (j = 0; j < nVar; j++)
                            lhs[i][j] += lhs[pivotRow][j] * val;
                      rhs[i] += rhs[pivotRow] * val;
```

```
                    }
               }

               //Compute solution
               for (i = 0; i < basic.size(); i++) {
                    if (isArtificial[basic[i]] && fabs(rhs[i]) > EPS)
                         return UNFEASIBLE;
                    if (basic[i] < solution.size())
                         solution[basic[i]] += rhs[i];
                    else if (isNegativePart[basic[i]])
                         solution[positivePart[basic[i]]] += -
rhs[i];
               }

               return OK;
          }
          /******Methods******/
};
////////////////
#include<numeric>
class PreciousStones {
public:
     LD value(vector<int> silver, vector<int> gold) {
          int i, j, N = silver.size();
          int nCon = N + 1;
          int nVar = N;

          SimplexModel model;
          //Objective funtion
          for (i = 0; i < silver.size(); i++)
               model.of.push_back(silver[i]);
          model.oftype = MAX;

          //Constraints
          model.unRestricted.resize(nVar, false);
          model.constraintTypes.resize(nCon, LE);
          model.lhs.resize(nCon, vector<LD> (nVar, 0));
          for (i = 0; i < N; i++) {
               model.rhs.push_back(1);
               model.lhs[i][i] = 1;
          }
          model.rhs.push_back(accumulate(gold.begin(),
gold.end(), 0));
          for (i = 0; i < N; i++)
               model.lhs.back()[i] = silver[i] + gold[i];

          Result r = model.solve();

          LD d = 0;
          for (i = 0; i < model.solution.size(); i++)
               d += model.solution[i] * silver[i];
          return d;
```

```
	}
};
```

Closest Pair of Points O(N lg N)

```
#define type double
#define MapIterator map<type, multiset<type> >::iterator
#define SetIterator multiset<type>::iterator

const int SIZE = 10000; //Maximum number of points
type x[SIZE], y[SIZE]; //Coordinates of points
int N; //Number of points
double INF = INT_MAX;
double getClosestPair() {
	map<type, multiset<type> > points;
	for (int i = 0; i < N; i++)
		points[x[i]].insert(y[i]);
	double d = INF;
	for (MapIterator xitr1 = points.begin(); xitr1 != points.end();
xitr1++){
	for (SetIterator yitr1 = (*xitr1).second.begin(); yitr1!=
(*xitr1).second.end(); yitr1++) {
	type x1 = (*xitr1).first, y1 = *yitr1;
	MapIterator xitr3 = points.upper_bound(x1 + d);
	for (MapIterator xitr2 = xitr1; xitr2 != xitr3; xitr2++)
	{
		type x2 = (*xitr2).first;
		SetIterator yitr2 = (*xitr2).second.lower_bound(y1 - d);
		SetIterator yitr3 = (*xitr2).second.upper_bound(y1 + d);
		for (SetIterator yitr4 = yitr2; yitr4 != yitr3; yitr4++) {
			if (xitr1 == xitr2 && yitr1 == yitr4)
				continue; //same point		type y2 = *yitr4;
			d = min(d, hypot(x1 - x2, y1 - y2));
		}
	}
	}
	}
	return d;
}
```

Fraction class

```
#define ABS(x) ((x)>=0?(x):-(x))
struct frac {
	long long n, d;
	frac(const long long& N, const long long &D = 1) :
		n(N), d(D) {
		long long g = gcd(ABS(n), ABS(d));
		if (!g) {
			this->n = this->d = 0;
			return;
		}
		n /= g;
```

```
            d /= g;
            if (n == 0)
                  d = 1;
            if (d < 0)
                  n *= -1, d *= -1;
            if (d == 0)
                  n = 1;
      }
      bool operator<(const frac &f) const {
            return n * f.d < d * f.n;
      }
      frac operator*(const frac &f) const {
            return frac(n * f.n, d * f.d);
      }
      frac operator/(const frac&f) const {
            return frac(n * f.d, d * f.n);
      }
      frac operator-(const frac &f) const {
            return frac(n * f.d - d * f.n, d * f.d);
      }
      frac operator+(const frac &f) const {
            return frac(n * f.d + d * f.n, d * f.d);
      }
};
```

Permutations

```
int getIndex(char * str) {
      int res = 0;
      if (!*str)
            return 1;
      bool vis[26] = { 0 };
      for (char * s = str + 1; *s; s++)
            if (!vis[*s - 'a'] && *s < *str) {
                  vis[*s - 'a'] = 1;
                  int count[26] = { 0 };
                  int chars[26];
                  int size = 0, len = 0;
                  for (char * ss = str; *ss; ss++) {
                        if (ss == s)
                              continue;
                        if (!(count[*ss - 'a']++))
                              chars[size++] = *ss - 'a';
                        len++;
                  }
                  long long f = 1;
                  for (int i = len; i > 1; i--) {
                        f *= i;
                        for (int j = 0; j < size; j++) {
                              int & r = count[chars[j]];
                              while (r > 1 && f % r == 0) {
                                    f /= r;
```

```
                                    r--;
                              }
                        }
                  }
                  res += f;
            }
      return res + getIndex(str + 1);
}

typedef vector<int> vi;
// p should contain numbers (0)->(n-1)
// returns the permutation number of p (0 indexed)
int permToIndex(vi p) {
      if (sz(p) <= 1)
            return 0;
      if (sz(p) == 2)
            return p[0];
      int f = 1;
      for (int i = 1; i < sz(p); i++)
            f *= i;
      vi r = p;
      r.erase(r.begin());
      for (int i = 0; i < sz(r); i++)
            if (r[i] > p[0])
                  r[i]--;
      return f * p[0] + permToIndex(r);
}

#define pb push_back
// j is the permutaion number
// d is the number of elements in the permutaion
// returns the jth permutaion
vi indexToPerm(int j, int d) {
      if (d == 1) {
            vi ret;
            ret.pb(0);
            return ret;
      }
      int f = 1;
      for (int i = 2; i < d; i++)
            f *= i;
      vi r(d);
      r[0] = j / f;
      vi t = indexToPerm(j % f, d - 1);
      for (int i = 0; i < sz(t); i++)
            if (t[i] >= r[0])
                  t[i]++;
      int ff = 0, tt = 1;
      rep(i,sz(t))
            r[tt++] = t[ff++];
      return r;
}
```

kthRoot

```
ll kthRoot(ll n, ll k)  // return integer kth root for n
{       // Also can be done by binary search for accurate results
        double root = pow((double)n, 1.0 / (double) k); // will
have percision errors
        ll realRoot = (ll)(root-1);

        while(1) {
                ll a = realRoot + 1, p = 1;
                for(int j = 0; j < k; j++)      // compute a^k
                {
                        if(p > n / a)   // we exceed n, this also
detect overflow
                                return realRoot;
                        p *= a;
                }
                ++realRoot;
        }
}
```

Josephus cycle

```
// Assume cycle [1 - n], and we kill mth, then 2mth..
// all sent arguments are 1-based
int joseph_lastKilled(int n, int m, int firstKilled = 1) {
        int k = 0;
        for(int i = 2; i<=n; k=(k+m)%i, i++);   // k represent
last killed person when cycle length=i
        k = (k-(m-firstKilled)+10000*n)%n;              // shift
the k, note: M may be > n
        while(k < 0)    k += n;
        return k;
}
int JosephCycle(int n, int m, int k) // using segment tree
{       // JosephCycle(5, 2, 3) = 5, after how many iter, k will
die
    int cur = 1;

    build(1, n,1 );     //  build tree from 1-n
    for(int i = n;i > 0; i--)   // UNTILL i > 0
    {
        cur=(cur+m-1)%i;
        if(cur == 0) cur = i;   // I think this is done because
it is 1-based
        // cur the index to be killed starting from START.
        if( del(1, n, cur, 1) == k ) return n-i+1;
    }
    return -1;  // must not happen
}
// test if any element in range n/2 is killed in n/2 iteration
bool JosephCycleTest(int n, int m) {    // test first n/2 kill
operation
```

```
        for(int cur = 0, i = n; i >n/2 ; i--) {
                cur = (cur+m-1)%i;
                if(cur < n/2)   return false;   // 0-based
compare parameters
        }
        return true;
}
```

build_bellNumbers

```
const ll MAX_BELL = 1000;
ll bell[MAX_BELL] = {1};
ll rows[2][MAX_BELL] = {1}, p = 0;
// number of partitions of a set of size n
// E.g. set{1, 2, 3, 4} can be divided {{1}, {3,2, 4}} or {{2, 4}
{1, 3}}
// NOTE: partitions {{1, 2},{3, 4}} and {{3, 4}, {2, 1}} are
counted once. NO ORDER ISSUES
void build_bellNumbers() {      // O(n^2)
        build_nCk();
        for(i, 1, MAX_BELL) repi(k, 0, i)      bell[i] += C[i-
1][k] * bell[k];
}

void build_bellNumbers2() {     // O(n*(n+1)/2) // bell triangle
        repi(i, 1, MAX_BELL) {
                p = !p, bell[i-1] = rows[p][0] = rows[!p][i-1];
             repi(j, 1, i+1) rows[p][j] = rows[p][j-1] +
rows[!p][j-1];
        }
}
```

fast_Fibonacci O(log(n))

```
int fast_Fibonacci(int n){
    int i=1, h=1, j=0, k=0, t;
    while (n > 0) {
                if (n%2 == 1)
                        t = j*h, j = i*h + j*k + t, i = i*k + t;
        t = h*h, h = 2*k*h + t, k = k*k + t, n = n/2;
    }
    return j;
    /*          Golden Mean
        double d = sqrt(5);
        double b=pow( (1+d)/2, n);
        double c=pow( (1-d)/2, n);
        cout<<(b-c)/d;
    */
}
```

repeating_digits_after_decimal_point_from_rational_number

```
int numBeforeRepeat(int n, int d) {
    int c2=0, c5=0;
```

```
    if (n == 0) return 1;
    while (d%2==0) d/=2, c2++;
    while (d%5==0) d/=5, c5++;
    while (n%2==0) n/=2, c2--;
    while (n%5==0) n/=5, c5--;
    if (c2 > c5)
        return c2 > 0 ? c2 : 0;
    return c5 > 0 ? c5 : 0;
}
void repeating_fractions_from_rational_mumber(int n, int d)
{       // you can apply it, to any base, but keep n, d in
decimal base
    cout<<n/d<<'.', n%=d;
    int m=numBeforeRepeat(n,d);
    for(int i=0;i<m;i++)
        n*=10, cout<<n/d, n%=d;
    int count = 0, r = n;
    if(r!=0)
    {
        do
        {
            n*=10, cout<<n/d, n%=d, count++;
        } while (n!=r);
        cout<<"\nThe last "<<count<<" digits repeat forever";
    }
}
```

FFT

```
typedef complex<double> Complex;
const Complex I(0, 1);
void fft(double theta, vector<Complex> &a) {
      int n = a.size();
      for (int m = n; m >= 2; m >>= 1) {
            int mh = m >> 1;
            for (int i = 0; i < mh; i++) {
                  Complex w = exp(i * theta * I);
                  for (int j = i; j < n; j += m) {
                        int k = j + mh;
                        Complex x = a[j] - a[k];
                        a[j] += a[k];
                        a[k] = w * x;
                  }
            }
            theta *= 2;
      }
      int i = 0;
      for (int j = 1; j < n - 1; j++) {
            for (int k = n >> 1; k > (i ^= k); k >>= 1) {
            }
            if (j < i)
                  swap(a[i], a[j]);
```

```
	}
}
void fft(vector<Complex> &a) {
	int n = ceil(log(a.size()) / log(2));
	a.resize(1 << n);
	fft(2 * PI / a.size(), a);
}
void ifft(vector<Complex> &a) {
	int n = ceil(log(a.size()) / log(2));
	a.resize(1 << n);
	fft(-2 * PI / a.size(), a);
	for (int i = 0; i < a.size(); i++)
		a[i] /= a.size();
}
char a[11001], b[11001], c[22203];
void mul() {
	int sa = strlen(a);
	int sb = strlen(b);
	int sc = sa + sb + 1;
	vector<Complex> A(sc), B(sc), C;
	for (int i = sa - 1, j = 0; i >= 0; i--)
		A[j++] = a[i] - '0';
	for (int i = sb - 1, j = 0; i >= 0; i--)
		B[j++] = b[i] - '0';
	fft(A);
	fft(B);
	C.resize(A.size());
	for (int i = 0; i < A.size(); i++)
		C[i] = A[i] * B[i];
	ifft(C);
	for (int i = 0; i < C.size() - 1; i++) {
		int cr = round(C[i].real()) / 10;
		C[i] = fmod(round(C[i].real()), 10.0);
		C[i + 1] += cr;
	}
	int i = C.size() - 1, j;
	while (i >= 0 && fabs(C[i].real()) < 1e-9)
		i--;
	if (i < 0) {
		c[0] = '0', c[1] = 0;
		return;
	}
	for (j = 0; i >= 0; j++, i--)
		c[j] = round(C[i].real()) + '0';
	c[j] = 0;
}
```

Other

Binary Search

```
inline double b_s(double s, double e) {
	for(double size=(e-s)/2; size>1e-9;size*=.5)//not less than
1e-15
		if (valid(s + size))
			s += size;
	return s;
}

//Searching for the last True in TTTTTTTTTTTTTFFFFFFFFFFFFF
pattern
//change val to !val to search for last False in
FFFFFFFFFTTTTTTTTT pattern
int b_s(int s, int e) {
	while (s < e) {
		int mid = s + (e - s + 1) / 2;
		if (valid(mid))
			s = mid;
		else
			e = mid - 1;
	}
	return s;
}

//Searching for the first True in FFFFFFFFTTTTTTTT pattern
//change val to !val to search for the first False in
TTTTTTTTFFFFFFF pattern
inline int bs(int s, int e) {
	while (s < e) {
		int mid = (s + (e - s) / 2);
		if (valid(mid))
			e = mid;
		else
			s = mid + 1;
	}
	return s;
}
```

Ternary Search

```
double ternarySearch(double left, double right) {
	while (right - left > EPS) {
		double g = left + (right - left) / 3, h = left + 2 * (right
- left) / 3;
		if (f(g) < f(h))
			right = h;
		else
			left = g; // change < to > if the fn inc then dec
	}
	return f((left + right) / 2);
```

```
}
```

Max Empty Rectangle

```
const int MAX = 3000;
class MaxEmptyRect {
private:
	int W, H, N;
	vector<int> dCols[MAX + 2];
	int pLeft[MAX + 1], pRight[MAX + 1], pTop[MAX + 1];
	int best;
public:
	MaxEmptyRect(vector<pair<int, int> > points, int height, int width) {

		H = height;
		W = width;
		N = points.size();
		best = 0;
		memset(pLeft, 0, sizeof(pLeft));
		memset(pRight, 0, sizeof(pRight));
		memset(pTop, 0, sizeof(pTop));
		int i;
		for (i = 0; i < N; i++) {
			int r = points[i].first, c = points[i].second;
			dCols[r].push_back(c);
		}
		for (i = 0; i <= H; i++) {
			dCols[i].push_back(0);
			dCols[i].push_back(W + 1);
			sort(dCols[i].begin(), dCols[i].end());
		}
		int k;
		for (i = 1; i <= H; i++) {
			k = 0;
			for (int j = 1; j <= W; j++) {
				if (dCols[i][k + 1] == j) {
					k++;
					pTop[j] = i;
					continue;
				}
				if (pTop[j] + 1 == i) {
					pLeft[j] = dCols[i][k];
					pRight[j] = dCols[i][k + 1];
				} else {
					pLeft[j] = dCols[i][k] > pLeft[j] ? dCols[i][k] :
pLeft[j];
					pRight[j] = dCols[i][k + 1] < pRight[j] ?
dCols[i][k + 1]
							: pRight[j];
				}

				int area = (i - pTop[j]) * (pRight[j] - pLeft[j] - 1);
```

```
                        best = area > best ? area : best;
                    }
                }
            }

            int getMaxEmptyArea() {
                return best;
            }
};

//MAIN
MaxEmptyRect m(vec, l, w);
cout << m.getMaxEmptyArea() << endl;

Max empty rectangle, On border, O(N^2)
#define point pair<int,int>
class MaxEmptyRect {
private:
      vector<point> P;
      int l,w;
      int best;
      void update(int a) {best = a > best ? a : best;}
      void split(int i, int y0, int y1) {
            if(l*(y1-y0) < best)return;
            int px,py;
            if (y0==y1) return;
            if (i==P.size())update(l*(y1-y0));
            else {
                  px=P[i].first;
                  py=P[i].second;
                  if (y0<=py && py<=y1) {
                        update( px*(y1-y0) );
                        split(i+1,y0,py);
                        split(i+1,py,y1);
                  } else split(i+1,y0,y1);
            }
      }
      void sweep() {

            int i,j, y0,y1, pix,piy,pjx,pjy;
            for (i=0; i<P.size(); i++) {
                  y0=0; y1=w;
                  pix=P[i].first; piy=P[i].second;
                  for (j=i+1; j<P.size(); j++) {
                        pjx=P[j].first; pjy=P[j].second;
                        if (y0<=pjy && pjy <=y1) {
                              update( (pjx-pix)*(y1-y0) );
                              if (pjy<piy)y0=pjy;
                              else if (pjy>piy)y1=pjy;
                              else break;
                        }
                  }
```

```
                if (j==P.size())update( (l-pix)*(y1-y0) );
            }
        }

public:
    MaxEmptyRect(vector<pair<int, int> > points, int height, int width) {
        P = points;
        l = height;
        w = width;
        best = 0;
        sort(P.begin(),P.end());
        split(0,0,w);
        sweep();
    }

    int getArea() {return best;}
};

//MAIN
MaxEmptyRect m(vec, l, w);
cout << m.getArea() << endl;
```

LIS O(N lg K)

```
vector<int> find_lis(vector<int> a) {//Finds longest strictly
increasing subsequence. O(n log k) algorithm.
    vector<int> b, p(a.size());
    int u, v;
    if (a.size() < 1)
        return b;
    b.push_back(0);
    for (int i = 1; i < (int) a.size(); i++) {
        if (a[b.back()] < a[i]) {
            p[i] = b.back();
            b.push_back(i);
            continue;
        }
        for (u = 0, v = b.size() - 1; u < v;) {
            int c = (u + v) / 2;
            if (a[b[c]] < a[i])
                u = c + 1;
            else
                v = c;
        }
        if (a[i] < a[b[u]]) {
            if (u > 0)
                p[i] = b[u - 1];
            b[u] = i;
        }
    }
    for (u = b.size(), v = b.back(); u--; v = p[v])
        b[u] = v;
```

```
	return b;
}
```

2 SAT

```
int n, c;
vector<vector<int> > adjList;

int getvar(int p_id) {
	return p_id * 2;
}

int getnotvar(int var) {
	return var ^ 1;
}

void add_or(int a, int b) {
	adjList[getnotvar(b)].push_back(a);
	adjList[getnotvar(a)].push_back(b);
}

vector<vector<int> > comps;
vector<int> comp_id;
vector<bool> issrc, issnk;
vector<vector<int> > compadj;
vector<int> indeg;

void buildCompGraph() {
	issrc = vector<bool> (comps.size(), 1);
	issnk = vector<bool> (comps.size(), 1);
	indeg = vector<int> (comps.size(), 0);
	compadj = vector<vector<int> > (comps.size());

	rep (i , adjList.size()) {
		rep (k , adjList[i].size()) {
			int j = adjList[i][k];
			int ii = comp_id[i], jj = comp_id[j];
			if (ii != jj) {
				issnk[ii] = issrc[jj] = 0;
				indeg[jj]++;
				compadj[ii].push_back(jj);
			}
		}
	}
}

vector<int> notcomp;
vector<int> compval;
void getsol() {
	notcomp = vector<int> (comps.size());
	compval = vector<int> (comps.size(), -1);
```

```
	rep(i , adjList.size()) {
		int j = getnotvar(i);
		int ii = comp_id[i], jj = comp_id[j];
		notcomp[ii] = jj;
	}

	queue<int> q;
	rep(i , comps.size()) {
		if (issrc[i])
			q.push(i);
	}

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

		rep (k , compadj[i].size()) {
			int j = compadj[i][k];
			if (!--indeg[j])
				q.push(j);
		}
		if (compval[i] == -1)
			compval[i] = 0, compval[notcomp[i]] = 1;
	}
}
```

Algorithm X

```
//Code for Sudoku 16* 16
#define fo(i,n) for(i=0;i<(n);++i)

typedef vector<int> vi;
typedef vector<string> vs;
typedef vector<double> vd;
#define sz(x) ((int)(x).size())
#define all(x) x.begin(),x.end()
#define pb(x) push_back(x)

// dancing links with pointers used when only one test case
#define MAXROW 16*16*16
#define MAXCOLS 16*16*5
int fcol; // fixed constraints
int cols; // count of columns (constraints)
vector<vector<int> > adj;
struct node {
	node* lf, *rt, *up, *dn;
	int id;
	union {
		node* hdr;
		int cnt;
	};
```

```
      inline void set(node* l, node* r, node* u, node* d, int
idx, node* h) {
            lf = (l), rt = (r), up = (u), dn = (d), id = (idx),
hdr = (h); // el coach 2al mtshlsh el akwas

            lf->rt = this;
            rt->lf = this;
            up->dn = this;
            dn->up = this;
      }
      inline void coverLR() {
            lf->rt = rt;
            rt->lf = lf;
      }
      inline void coverUD() {
            up->dn = dn;
            dn->up = up;
      }
      inline void unCoverLR() {
            lf->rt = this;
            rt->lf = this;
      }
      inline void unCoverUD() {
            up->dn = this;
            dn->up = this;
      }
      inline void coverCol() {
            coverLR();
            for (node* x = dn; x != this; x = x->dn)
                  for (node* y = x->rt; y != x; y = y->rt) {
                        y->coverUD();
                        y->hdr->cnt--;
                  }
      }
      inline void unCoverCol() {
            for (node* x = up; x != this; x = x->up)
                  for (node* y = x->lf; y != x; y = y->lf) {
                        y->unCoverUD();
                        y->hdr->cnt++;
                  }
            unCoverLR();
      }
};
node* root;
inline node* selectMinC() {
      node* mn = NULL;
      int mnCnt = INT_MAX;
      for (node* tmp = root->rt; tmp->id < fcol && tmp != root;
tmp = tmp->rt)
            if (tmp->cnt < mnCnt)
                  mn = tmp, mnCnt = tmp->cnt;
      return mn;
```

```
}
int solCnt;
int sol[MAXROW];
inline bool algoX() {
	node* mn = selectMinC();
	if (!mn)
		return true; // turn into false if all solutions
required
	mn->coverCol();
	for (node* x = mn->dn; x != mn; x = x->dn) {
		for (node* y = x->rt; y != x; y = y->rt)
			y->hdr->coverCol();
		sol[solCnt++] = x->id;

		if (algoX())
			return true;
		solCnt--;
		for (node* y = x->lf; y != x; y = y->lf)
			y->hdr->unCoverCol();
	}
	mn->unCoverCol();
	return false;

}
node* hdrs[MAXCOLS];
inline void build() {
	solCnt = 0;
	root = new node();
	root->set(root, root, root, root, 0, 0);
	for (int i = 0; i < cols; i++) {
		hdrs[i] = new node();
		hdrs[i]->set(root->lf, root, hdrs[i], hdrs[i], i, 0);
	}
	for (int i = 0; i < sz(adj); i++) {
		node* fn;
		for (int k = 0; k < sz(adj[i]); k++) {
			int j = adj[i][k];
			if (k)
				(new node())->set(fn->lf, fn, hdrs[j]->up,
hdrs[j], i, hdrs[j]);
			else {
				fn = new node();
				fn->set(fn, fn, hdrs[j]->up, hdrs[j], i,
hdrs[j]);
			}
			hdrs[j]->cnt++;
		}
	}
}
inline void init(int n) {
	adj.clear();
	adj.resize(n);
```

```
}
char b[16][17];
int main() {
	char* t = "";
	while (1) {
		if (scanf(" %c", &b[0][0]) == EOF)
			break;
		int cnt = b[0][0] != '-';
		for (int i = 0; i < 16; i++)
			for (int j = i == 0; j < 16; j++)
				scanf(" %c", b[i] + j), cnt += b[i][j] !=
'-';

		int cell = 0;
		int rws = 16 * 16;
		int cls = rws + 16 * 16;
		int bxs = cls + 16 * 16;
		int fxd = bxs + 16 * 16;
		cols = fcol = fxd + cnt;
		init(16 * 16 * 16);
		for (int i = 0; i < 16; i++)
			for (int j = 0; j < 16; j++)
				for (int k = 0; k < 16; k++) {
					int rnk = (i * 16 + j) * 16 + k;
					adj[rnk].pb(cell+i*16+j);
					adj[rnk].pb(rws+i*16+k);
					adj[rnk].pb(cls+j*16+k);
					int bxi = i / 4;
					int bxj = j / 4;
					int bi = bxi * 4 + bxj;
					adj[rnk].pb(bxs+bi*16+k);
					if (b[i][j] == k + 'A')
						adj[rnk].pb(fxd++);
				}

		build();
		algoX();

		for (int l = 0; l < solCnt; l++) {
			int k = sol[l] % 16;
			sol[l] /= 16;
			int j = sol[l] % 16;
			int i = sol[l] / 16;
			b[i][j] = k + 'A';
		}
		printf(t);
		t = "\n";
		for (int i = 0; i < 16; i++)
			printf("%s\n", b[i]);
	}
	return 0;
}
```

Partitioning

```
//this means that u start value s then s + i*DIVIDE_RANGE such
that s + i*DIVIDE_RANGE < e
//every time the range will be divided by DIVIDE_RANGE and so on
#define DIVIDE_RANGE 10 //the termination of the delta value once
it's almost zero depending of the problem
#define TERMINATE 1e-9
long double get_best_using_partitioning(long double start, long
double end) {
	long double delta = (end - start) / DIVIDE_RANGE, res = oo;
	long double best;
	while (delta > TERMINATE) {
		for (long double current = start; current <= end;
current += delta) {
			long double temp = solve(current);
			if (temp < res) {
				res = temp;
				best = current;
			}
		}
		start = best - delta;
		end = best + delta;
		delta /= DIVIDE_RANGE;
	}
	return best;
}
```

Expressions and Parsing

```
//#define __put_brackets_in_tree
struct ExpParsing {
	enum TYPE {
		OP, NUM, VAR, BRAC, SEMICOLON, LN, EOE
	};
	typedef pair<TYPE, string> TOKEN;
	queue<TOKEN> TOKS;
	map<string, TYPE> reservedWords; // saving all reserved
words
	map<string, int> vars; // saving all variables with thier
values
	void reserved(TOKEN &t) { // take token if its one of
reserved words it adapt its type
		map<string, TYPE>::iterator it =
reservedWords.find(t.second);
		if (it != reservedWords.end())
			t.first = it->second;
	}
	void Tokinize(const char* exp) { // parsing the statment to
tokens
		TOKEN t;
		for (const char* c = exp; *c; c++) {
			if (isspace(*c))
```

```
                    continue;
                switch (*c) {
                case '+':
                case '-':
                case '=':
                case '/':
                case '*':
                case '%':
                case '^':
                    t.second = string(1, *c);
                    t.first = OP;
                    break;
                    // case ':' : t.second=string(1,*c);
t.first=OP; if(*(c+1)=='=') { t.second+=*(++c); }
                case '(':
                case ')':
                    t.second = string(1, *c);
                    t.first = BRAC;
                    break;
                case ';':
                    t.second = string(1, *c);
                    t.first = SEMICOLON;
                    break;
                default:
                    t.second = string(1, *c);
                    if (isdigit(*c++)) {
                        t.first = NUM;
                        while (isdigit(*c) || *c == '.')
                            t.second += *(c++);
                    } else {
                        t.first = VAR;
                        while (isalnum(*c) || *c == '_')
                            t.second += *(c++);
                    }
                    c--; // on for loop there is c++
                    reserved(t);
                } // if this token is a reserved word it will
adapt its type
                TOKS.push(t);
            }
            t.first = EOE; // end of expression ( to avoid RTE )
            t.second = "";
            TOKS.push(t);
        }
        struct NODE { // if (memory limit exceed) Destructor
recommended
            TOKEN r;
            NODE* lf, *rt, *p;// parent
            NODE() :
                lf(0), rt(0), p(0) {
            }
            NODE(TOKEN _r, NODE* _lf, NODE* _rt) :
```

```
                r(_r), lf(_lf), rt(_rt), p(0) {
        }
    };
    // NODE* expr(); // declration (out of struct)
    NODE* base() {
        TOKEN t = TOKS.front();
        TOKS.pop();
        NODE* n;
        switch (t.first) {
        case NUM:
        case VAR:
            return new NODE(t, 0, 0);
        case BRAC:
            n = expr();
            TOKS.pop();
#ifdef __put_brackets_in_tree
            return new NODE(make_pair(BRAC,string("")),n,0);
#else
            return n;
#endif
        case OP:
            return new NODE(t, 0, base()); // unary minus
        case LN:
            TOKS.pop();
            n = expr();
            TOKS.pop();
            return new NODE(t, n, 0);
        default:
            ;
        }
        return n;
    }
    NODE* factor() {
        NODE* b = base();
        TOKEN t = TOKS.front();
        if (t.second != "^") {
            return b;
        }
        TOKS.pop();
        return new NODE(t, b, factor());
    }
    NODE* term_(NODE* n) {// term'
        TOKEN t = TOKS.front();
        if (t.second == "*" || t.second == "/" || t.second ==
"%") {
            TOKS.pop();
            return term_(new NODE(t, n, factor()));
        }
        return n;
    }
    NODE* term() {
        return term_(factor());
```

```
	}
	NODE* expr_(NODE* n) { // expr'
		TOKEN t = TOKS.front();
		if (t.second == "+" || t.second == "-") {
			TOKS.pop();
			return expr_(new NODE(t, n, term()));
		}
		return n;
	}
	NODE* expr() {
		return expr_(term());
	}
	int eval(NODE* t) {
		if (t == 0)
			return 0;
		int res;
		switch (t->r.first) {
		case OP:
			switch (t->r.second[0]) {
			case '+':
				return eval(t->lf) + eval(t->rt);
			case '-':
				return eval(t->lf) - eval(t->rt);
			case '/':
				return eval(t->lf) / eval(t->rt);
			case '*':
				return eval(t->lf) * eval(t->rt);
			case '%':
				return eval(t->lf) % eval(t->rt);
			case '^':
				return (int) pow((double) eval(t->lf),
(double) eval(t->rt));
			}
		case NUM:
			sscanf(t->r.second.c_str(), "%d", &res);
			return res;
		case VAR:
			return vars[t->r.second];
		case BRAC:
			return eval(t->lf);
		}
		return 0;
	}
	void statement() {
		TOKEN var = TOKS.front();
		TOKS.pop();
		if (TOKS.empty())
			return; // if its empty line
		TOKS.pop();
		NODE* tree = expr();
		// actual main for vary according to problem statement
		//most of expression be on this BNF
```

```
            //EXP -> TERM E'
            //E' -> + TERM E'| - TERM E'
            //TERM -> FACTOR T'
            //T' -> * FACTOR T' | / FACTOR T' | e
            //FACTOR -> BASE^FACTOR|BASE
            //BASE -> VAR| NUM| (EXP)
            vars[var.second] = eval(tree);
            TOKS.pop(); // for semicolon --Remove it if there is
no semicolons--
            TOKS.pop();
      } // for EOE
      NODE* deff(NODE* t) { // deffrentiation
            NODE* t1, *t2, *t3, *t4, *t5;
            switch (t->r.first) {
            case OP:
                  switch (t->r.second[0]) {
                  case '-':
                        if (!t->lf) {
                              if (t->rt->r.first == NUM)
                                    return new NODE(make_pair(NUM,
string("0")), 0, 0);
                              return new NODE(t->r, 0, deff(t->rt));
                        }
                  case '+':
                        return new NODE(t->r, deff(t->lf), deff(t-
>rt));
                  case '*':
                        t1 = new NODE(make_pair(OP, string("*")),
deff(t->lf), t->rt);
                        t2 = new NODE(make_pair(OP, string("*")),
t->lf, deff(t->rt));
                        t3 = new NODE(make_pair(OP, string("+")),
t1, t2);
                        return new NODE(make_pair(BRAC,
string("")), t3, 0);
                  case '/':
                        t1 = new NODE(make_pair(OP, string("*")),
deff(t->lf), t->rt);
                        t2 = new NODE(make_pair(OP, string("*")),
t->lf, deff(t->rt));
                        t3 = new NODE(make_pair(OP, string("-")),
t1, t2);
                        t4 = new NODE(make_pair(BRAC, string("")),
t3, 0);
                        t5 = new NODE(make_pair(OP, string("^")),
t->rt,
                              new NODE(make_pair(NUM,
string("2")), 0, 0));
                        return new NODE(make_pair(OP, string("/")),
t4, t5);
                  }
            case NUM:
```

```
                return new NODE(make_pair(NUM, string("0")), 0,
0);
          case VAR:
                return new NODE(make_pair(NUM, string("1")), 0,
0);
          case BRAC:
                return new NODE(make_pair(BRAC, string("")),
deff(t->lf), 0);
          case LN:
                t1 = new NODE(make_pair(BRAC, string("")),
deff(t->lf), 0);
                t2 = new NODE(make_pair(BRAC, string("")), t->lf,
0);
                return new NODE(make_pair(OP, string("/")), t1,
t2);
          }
     }
     string print(NODE* t) {
          if (!t)
                return "";
          string res;
          switch (t->r.first) {
          case OP:
                return print(t->lf) + t->r.second + print(t->rt);
          case NUM:
          case VAR:
                return t->r.second;
          case BRAC:
                return "(" + print(t->lf) + ")";
          case LN:
                return "ln(" + print(t->lf) + ")";
          }
     }
     map<TOKEN, int> prec, notass;// for precedence and
associativity
     void setprec_Ass() {
          prec[make_pair(OP, string("+"))] = 1;
          prec[make_pair(OP, string("-"))] = 1;
          prec[make_pair(OP, string("*"))] = 2;
          prec[make_pair(OP, string("/"))] = 2;
          notass[make_pair(OP, string("+"))] = 0;
          notass[make_pair(OP, string("-"))] = 1;
          notass[make_pair(OP, string("*"))] = 0;
          notass[make_pair(OP, string("/"))] = 1;
     }
     string printWithOutBraces(NODE* t) {
          if (!t)
                return "";
          bool br = 0;
          switch (t->r.first) {
          case OP:
                if (t->p && prec[t->p->r] > prec[t->r])
```

```
                    br = 1;
                if (t->p && prec[t->p->r] == prec[t->r] && t->p->rt == t
                        && notass[t->p->r])
                    br = 1;
                return (br ? "(" : "") + printWithOutBraces(t->lf) + t->r.second
                        + printWithOutBraces(t->rt) + (br ? ")" : "");
            case NUM:
            case VAR:
                return t->r.second;
            }
        }
        string printWithOutBracesAfter3aks(NODE* t) {
            if (!t)
                return "";
            bool br = 0;
            switch (t->r.first) {
            case OP:
                if (t->p && prec[t->p->r] > prec[t->r])
                    br = 1;
                return (br ? "(" : "") + printWithOutBracesAfter3aks(t->lf)
                        + t->r.second + printWithOutBracesAfter3aks(t->rt)
                        + (br ? ")" : "");
            case NUM:
            case VAR:
                return t->r.second;
            }
        }
        void makeParents(NODE* t) {
            if (t->lf)
                makeParents(t->lf), t->lf->p = t;
            if (t->rt)
                makeParents(t->rt), t->rt->p = t;
        }
        // if you call e3ks, then you must remove the printWithout
        void e3ks(NODE* n, int par_prec) {
            if (n->r.first != OP || prec[n->r] != par_prec)
                return;
            char *ops = "+-*/";
            int ind = find(ops, ops + 4, n->r.second[0]) - ops;
            ind = (ind / 2) * 2 + !(ind % 2);
            n->r.second[0] = ops[ind];
            e3ks(n->lf, par_prec);
        } //e3ks(n->rt, par_prec);
        void zabat_el_non_ass(NODE *n) { // distribute - and / operators (which are non-associative) on the other operators
            if (!n)
                return;
```

```
            if (n->r.second == "-" || n->r.second == "/")
                  e3ks(n->rt, prec[n->r]);
            zabat_el_non_ass(n->lf);
            zabat_el_non_ass(n->rt);
      }
};
```

Other others

Consecutive integers that sum to a given value

```
#define big long long
vector<pair<big, big> > whichSums(big target) {

      big n = (-1 + sqrt(1 + 8 * target)) / 2, i;
      vector<pair<big, big> > res;
      for (i = 1; i <= n; i++) {
            if (i % 2) {
                  if (target % i == 0)
                        res.push_back(
                                    make_pair(target / i - i / 2,
target / i + i / 2));
            } else if ((2 * target - i) % (2 * i) == 0)
                  res.push_back(
                              make_pair((2 * target - i) / (2 * i) -
i / 2 + 1,
                                          (2 * target - i) / (2 * i)
+ i / 2));
      }
      return res;
}
```

Calculating the palindrome substrings

```
int isP[2500][2500]; //2500 is the max string length
string all; //all the text
int isPalin(int start, int end) {
      if (start == end)
            return isP[start][end] = 1;
      if (end == start + 1)
            return isP[start][end] = (all[start] == all[end]) ? 1 : 0;
      if (isP[start][end] != -1)
            return isP[start][end];
      if (all[start] != all[end])
            return isP[start][end] = 0;
      isP[start][end] = isPalin(start + 1, end - 1);
      return isP[start][end];
}
//MAIN
//memset(isP, -1, sizeof(isP));
//for(int i = 0; i < all.size(); i++)for(int j = i; j < all.size();
j++)isPalin(i,j);
```

Permutation Cycles (disjoint cycles)

```
vector<vector<int> > getCycles(vector<int> vec) {
	vector<bool> visited(vec.size(), false);
	vector<vector<int> > cycles;
	while (true) {
		int start = -1, i;
		for (i = 0; i < vec.size(); i++)
			if (!visited[i]) {
				start = i;
				break;
			}
		if (start == -1)
			break;
		i = start;
		vector<int> cycle;
		while (true) {
			cycle.push_back(i);
			visited[i] = true;
			i = vec[i];
			if (i == start)
				break;
		}
		cycles.push_back(cycle);
	}
	return cycles;
}
```

Flatten rectangles

```
struct rect {
	int lx, ly, ux, uy, color;
	bool operator<(const rect& r) const {
		return lx < r.lx || (lx == r.lx && ly < r.ly)
				|| (lx == r.lx && ly == r.ly && ux < r.ux)
				|| (lx == r.lx && ly == r.ly && ux == r.ux
&& uy < r.uy);
	}
};

bool valid(rect M) {
	return (M.ux <= M.lx || M.uy <= M.ly) ? false : true;
}
vector<rect> intersect(vector<rect> vec, rect N) {
	set<rect> result;
	for (int i = 0; i < vec.size(); i++) {
		rect M = vec[i];
		//N doesn't intersect M
		if (N.lx >= M.ux || N.ux <= M.lx || N.ly >= M.uy ||
N.uy <= M.ly) {
			result.insert(M);
			continue;
		}
```

```
            rect r[4] = { { M.lx, M.ly, N.lx, M.uy, M.color }, {
N.ux, M.ly, M.ux,
                        M.uy, M.color }, { max(N.lx, M.lx), N.uy,
min(N.ux, M.ux), M.uy,
                        M.color }, { max(N.lx, M.lx), M.ly,
min(N.ux, M.ux), N.ly,
                        M.color } };

            for (int j = 0; j < 4; j++)
                  if (valid(r[j]))
                        result.insert(r[j]);
      }
      result.insert(N);
      vector<rect> v;
      for (set<rect>::iterator itr = result.begin(); itr !=
result.end(); itr++)
            v.push_back(*itr);
      return v;
}
vector<rect> flatten(vector<rect> vec) {
      vector<rect> result;
      for (int i = 0; i < vec.size(); i++)
            result = intersect(result, vec[i]);
      return result;
}
```

next_permuatation in java

```
void next_permutation(int[] arr) {
      int N = arr.length;
      int i = N - 1;
      while(arr[i-1] >= arr[i]) i = i-1;int j = N;while(arr[j-1] <= arr[i-1]) j = j-1;int temp = arr[i-
1];arr[i-1] = arr[j-1];arr[j-1] = temp;i++; j = N;
      while (i < j) {temp = arr[i-1];arr[i-1] = arr[j-1];arr[j-1] = temp;i++;j--;}}
}
```

Date

```
bool isLeap(int year) {
      return (year % 4 == 0 && year % 100 != 0) || year % 400 ==
0;
}
int days[] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
};
struct date {
      int year, month, day;
      date() {
      }
      date(int dd, int mm, int yy) {
            year = yy;
            month = mm;
            day = dd;
      }
```

```
	bool operator <(const date &d) const {
		if (year != d.year)
			return year < d.year;
		if (month != d.month)
			return month < d.month;
		return day < d.day;
	}
	bool operator>(const date &d) const {
		if (year != d.year)
			return year > d.year;
		if (month != d.month)
			return month > d.month;
		return day > d.day;
	}
	bool operator <=(const date &d) const {
		if (year != d.year)
			return year < d.year;
		if (month != d.month)
			return month < d.month;
		return day <= d.day;
	}
	bool operator>=(const date &d) const {
		if (year != d.year)
			return year > d.year;
		if (month != d.month)
			return month > d.month;
		return day >= d.day;
	}
	bool operator ==(const date &d) const {
		return year == d.year && month == d.month && day ==
d.day;
	}
	void next() {
		int dd = days[month];
		if (month == 2 && isLeap(year))
			dd++;
		day++;
		if (day > dd) {
			month++;
			day = 1;
			if (month > 12) {
				year++;
				month = 1;
			}
		}
	}
	void prev() {
		day--;
		if (day < 1) {
			month--;
			if (month < 1) {
				year--;
```

```
                    month = 12;
                }
                day = days[month];
                if (month == 2 && isLeap(year))
                    day++;
            }
        }
        string toString() {
            stringstream S;
            S << day << "/" << month << "/" << year;
            return S.str();
        }
};
```

Solving defragmentation problem using segment trees

```
const int SIZE = 200000; // 2*( 1<< ((int)(log2(50000))+1) );
struct node {
    int from, to; //segment this node is responsible for
    int left, right, big; //size of left, right, biggest spaces
with segment
    int state; //1 for empty, 0 for full, 2 for mixed
} nodes[SIZE];
int N, M, MAX_NODE = 0;
void createTree(int node, int from, int to) {
    nodes[node].from = from;
    nodes[node].to = to;
    nodes[node].state = 1;
    nodes[node].big = nodes[node].right = nodes[node].left = to
- from + 1;
    MAX_NODE = max(MAX_NODE, node);
    if (from == to)
        return; //leaf
    createTree(2 * node, from, (from + to) / 2);
    createTree(2 * node + 1, (from + to) / 2 + 1, to);
}
int query(int node, int size) {
    if (nodes[node].big < size)
        return 0;
    if (nodes[node].left >= size)
        return nodes[node].from;
    if (nodes[2 * node].big >= size)
        return query(2 * node, size);
    if (nodes[2 * node].right + nodes[2 * node + 1].left >=
size)
        return nodes[2 * node].to - nodes[2 * node].right + 1;
    return query(2 * node + 1, size);
}
void propagateState(int node) {
    nodes[2 * node].state = nodes[2 * node + 1].state =
nodes[node].state;
```

```
      nodes[2 * node].left = nodes[2 * node].right = nodes[2 *
node].big =
                  nodes[node].state * (nodes[2 * node].to - nodes[2
* node].from + 1);
      nodes[2 * node + 1].left = nodes[2 * node + 1].right =
                  nodes[2 * node + 1].big = nodes[node].state
                              * (nodes[2 * node + 1].to - nodes[2 *
node + 1].from + 1);
}
void modify(int node, int from, int to, int val) {
      if (nodes[node].from > to || nodes[node].to < from)
            return;
      if (nodes[node].from >= from && nodes[node].to <= to) {
            nodes[node].state = val;
            nodes[node].big = nodes[node].left = nodes[node].right
= val
                        * (nodes[node].to - nodes[node].from + 1);
            return;
      }
      if (nodes[node].state != 2) //Make sure children are
consistent with me if i'm not mixed
            propagateState(node);
      modify(2 * node, from, to, val);
      modify(2 * node + 1, from, to, val);

      nodes[node].state =
                  nodes[2 * node].state != nodes[2 * node +
1].state ?
                              2 : nodes[2 * node].state;
      nodes[node].left =
                  nodes[2 * node].state != 1 ?
                              nodes[2 * node].left :
                              nodes[2 * node].left + nodes[2 * node
+ 1].left;
      nodes[node].right =
                  nodes[2 * node + 1].state != 1 ?
                              nodes[2 * node + 1].right :
                              nodes[2 * node + 1].right + nodes[2 *
node].right;
      nodes[node].big = max(nodes[2 * node].big, nodes[2 * node +
1].big);
      nodes[node].big = max(nodes[node].big,
                  nodes[2 * node].right + nodes[2 * node +
1].left);
}
```

String utilities

```
bool isVowel(char t)
{
      t = tolower(t);
```

```
     if(t == 'a' || t == 'i' || t == 'u' || t == 'o' || t == 'e')
return true;
     return false;
}
string toLower(string t)
{
     for(int i = 0 ; i < t.size() ; i ++)
     {
             t[i] = tolower(t[i]);
     }
     return t;
}
bool replace(string& str, string fr, string to)
{
  int pos;
  if ((pos = str.find(fr)) != -1) {
    str = str.substr(0, pos) + to + str.substr(pos +
fr.length());

    return true;
  }
  return false;
}
vector<string> split(string t, char c)
{
        string m = "";
        vector<string> res;
        for(int i = 0 ; i < sz(t) ; i ++)
        {
                if(t[i] == c&&m!="")
                {
                        res.push_back(m);
                        m = "";
                }
                else
                        m+=t[i];
        }
        if(m!="")
                res.push_back(m);
        return res;
}
string toUpper(string t)
{

     for(int i = 0 ; i < t.size() ; i ++)
     {
             t[i] = toupper(t[i]);
     }

     return t;
}
```

```
int toDecimal(string s, int base)
{
    int v, i, result = 0;
    for(i = 0 ; i < s.size() ; i++)
    {
          if(s[i]>='0' && s[i] <= '9')  v = s[i] - '0';
          else v = s[i]-'A'+10;
          result = result*base+v;
    }
    return result;
}
int StoI(string s)
{
    int v, i, result = 0;
    for(i = 0 ; i < s.size() ; i++)
    {
           v = s[i] - '0';
          result = result*10+v;
    }
    return result;
}
string toBase(int num, int base)
{
       if(num ==0) return "0";
       string str;
       while(num!=0)
       {
        int nlet = num%base;
        num/= base;
        if(nlet<0)//for negative base
            num++,nlet+=(-1*base);
        if(nlet<10) str += (nlet+'0');
        else str += (nlet-10+'A');

       }
       reverse(str.begin(),str.end());
       return str;
}
string ItoS(int num )
{
       if(num == 0) return "0";
       string str;
       while(num!=0)
       {
        int nlet = num%10;
        str += (nlet+'0');
        num/= 10;
       }
       reverse(str.begin(),str.end());
```

```
        return str;
}
```

Int utilities

```
double dis(int x1,int y1,int x2,int y2)
{
                return sqrt(pow((double)abs(x1-
x2),2)+pow((double)abs(y1-y2),2));
}

int gcd (int x,int y)
{
    if(y==0)return x; return gcd(y,x%y);
}
int lcm (int x,int y)
{
    return x/gcd(x,y)*y;
}
int oo = 1000000001;
int C[203][203];
void buildnCr(int n) {
    for(int i = 0; i <  n ; i++)
      for(int j = 0 ;j < n  ; j++)
           C[i][j] = (j == 0) ? 1 : ( (i == 0) ? 0 : C[i-1][j-
1]+C[i-1][j]);
}
```

merge vector of pairs

```
( remove intersection ) make larger pairs .. v must contain 1
element at least
void merge(vector<pair<double, double> >& v, vector<pair<double,
double> >& ans)       {

      ans.clear();
      sort(v.begin(), v.end());
      pair<double, double> cur = v[0];

      for(int i=1;i<v.size();i++)
           if(v[i].first >= cur.first && v[i].first <=
cur.second)
                cur.second = max(cur.second, v[i].second);
           else ans.push_back(cur), cur = v[i];

      ans.push_back(cur);
}
```

Loop on all subsets of 1s for a certain number s

```
for(int i=s;i;i=(i-1)&s);
```

numDigits 1000 has four digits

```
int numDigits(int n) {
    return (int)log10(n)+1;
}
```

Roll die

```
////////////////////
string dir = "NSEW";    // you can rotate a die in 4 directions
//0=top 1=bottom 2=left 3=right 4=front 5=back
int rot[][6] = {
        // roll ON y-axix
        {4, 5, 2, 3, 1, 0}, // N
        {5, 4, 2, 3, 0, 1}, // S
        // roll ON x-axix
        {2, 3, 1, 0, 4, 5}, // E
        {3, 2, 0, 1, 4, 5}, // W
        // move AROUND z-axix
        {0, 1, 5, 4, 2, 3},
        {0, 1, 4, 5, 3, 2}
};

string roll(string die, char d) {       // assume d in dir
        string ndie = "";
        int idx = (int)dir.find( toupper(d) );
        lp(i, 6)    ndie += die[ rot[idx][i] ];
        return ndie;
}
// u should in paper, determine how is initial die, E.g. 163452
// not each two faces sum = 7
////////////////////////////////

// Generate all rotation of a Die
int rotLEFT[]={0,1,4,5,3,2};
int rotDOWN[]={4,5,2,3,1,0};

void rotate(string s, set<string> & rots) {
   if (rots.find(s) != rots.end()) return;
   rots.insert(s);

   string rot1 = "", rot2 = "";
   rep(i, 6)    rot1 += s[rotLEFT[i]];
   rep(i, 6)    rot2 += s[rotDOWN[i]];
   rotate(rot1, rots);
   rotate(rot2, rots);
}
// dice is 6 faces E.g. 012345 [top, bottom, left, right, front
back]
string getNormalDiceForm(string die) {
        set<string> allRotations;
```

```
        rotate(die, allRotations);      // generate all, and take
first
        return  *(allRotations.begin());
}
```

Time to string

```
string toTime(int total_sec) //120 sec is 2 minutes
{
    int days    =  total_sec / (60*60*24);
    int hours   =  total_sec / (60*60)-days*24;
    int minutes = (total_sec / 60) % 60;
    int sec     =  total_sec % 60;
    string period = " AM";
    if(hours > 12) hours -= 12, period = " PM";
    return toStr(days, 2)    +':'+ toStr(hours, 2)+':' +
                   toStr(minutes, 2) +':'+ toStr(sec, 2) +
period;
}
```

Month names

```
string months[12] = {"JANUARY", "FEBRUARY", "MARCH", "APRIL",
"MAY", "JUNE", "JULY", "AUGUST",
                                          "SEPTEMBER", "OCTOBER",
"NOVEMBER", "DECEMBER"};
```

Number names and fromNumTOWords and fromWordsTONum

```
string nums[20] = {
          "", "one", "two", "three", "four", "five", "six",
"seven",
          "eight", "nine", "ten", "eleven", "twelve","thirteen",

"fourteen","fifteen","sixteen","seventeen","eighteen","nineteen"
};

string tenths[10] = {
                "", "", "twenty", "thirty", "forty", "fifty",
"sixty",
                "seventy", "eighty", "ninety"
};

string fromNumTOWords(int num) //10 is ten
{
    if(num == 0)    return "zero";

    string res = "", thos = "", hund = "", tens = "";
    if(num < 0) num *= -1, res += "negative";

    int nThousands = num/1000;
    int nHundreds  = (num%1000)/100;
    int nTenths    = num - 1000*nThousands-100*nHundreds;
```

```
    if(nThousands) thos += fromNumTOWords(nThousands)+ "
thousand";
    if(nHundreds)  hund += nums[nHundreds] + " hundred";
    if(nTenths)    tens = (nTenths < 20 ) ? nums[nTenths] :
                  tenths[nTenths/10]+' '+nums[nTenths%10];

    res += thos + (( (nThousands) ? " " : "")) + hund;
    res += ( (nThousands||nHundreds) && nTenths) ? " and " : "";

    return res +tens;
}

int fromWordsTONum(string line)//ten is 10
{
    map<string, int> value;

    for(int i=1; i<20;i++) value[ nums[i]   ] = i;
    for(int j=2; j<10;j++) value[ tenths[j] ] = 10*j;
    value["zero"] = 0, value["hundred"] = value["thousand"] = -1;

    string word;
    int answer = 0, tens = 0, negative = 0;

    istringstream iss(line);
    while (iss >> word)
    {
        if (word == "and")      continue;
        else if (word == "negative") negative = 1;
        else if (value[word] == -1)
        {
            if(word == "thousand")
                answer = 1000*(answer+tens), tens = 0;
            else
                answer += 100*tens, tens = 0;
        }
        else
            tens += value[word];
    }
    return (negative) ? (answer+tens)*-1 : (answer+tens);
}
```

Written numbers' suffixes (st, nd, rd, ...)

```
string formatPostfix(int n) {
        int temp, mod1, mod2;
        temp = n, mod1 = temp%10, temp/=10, mod2 = temp%10;
        if(mod2 == 1) return "th";
        if(mod1 == 1) return "st";
        if(mod1 == 2) return "nd";
        if(mod1 == 3) return "rd";
        return "th";
}
```

Return angle from hour hand to minute hand.

```
double clockAngle(int h, int m, int s = 0) {
        double exactM = m+s/60.0, exactH = h%12+exactM/60.0;
// 60 sec is 1 min, 30 sec is 0.5 min
        double mDeg = exactM*6.0;
// calc angle clockwise. Each minute is 360/60=6  degree
        double hDeg = exactH*30.0;
// calc angle clockwise. Each hour   is 360/12=30 degree
        if(hDeg <= mDeg)        return mDeg-hDeg;
// Draw. Simply it is difference
        return 360 - (hDeg-mDeg);
// Draw. Simply it is complement
}
```

add integers in other bases

```
string B = "0123456789ABCDEF";
int I(char c) { return B.find(c);}
string add(string a, string b, int base) {
        int mx = max(sz(a), sz(b));

        int C[200] = {0};

        while( sz(a) != mx)     a = "0" + a;
        while( sz(b) != mx)     b = "0" + b;

        reverse( all(a) );
        reverse( all(b) );

        for (int i = 0; i < mx; ++i) {
                int t = C[i] + I(a[i]) + I(b[i]);
                C[i]    = t % base, C[i+1] += t / base;
        }

        int i = mx;
        while(i > 0 && C[i] == 0)       i --;

        string ret = "";
        for (int j = i; j >= 0; --j) ret += B[ C[j] ];

        return ret;
}
```

decToBase

```
string decToBase(ll number, int base)
{
    if(number == 0)     return "0";
    string res = "", encode = "0123456789ABCDEF";

    while(number)
        res = encode[number % base] + res, number /= base;
```

```
    return res;
}
```

toDecimal

```
ll toDecimal(string number, int base) { // Watchout OVERFLOW
inputs
    string decode = "0123456789ABCDEF";
    ll res = 0;
    for (int i=0;i<number.size(); ++i)
        res *= base, res += decode.find(number[i]);
    return res;
}
```

roman_to_int

```
int value(char ch) {
    if(ch =='I') return 1;              if(ch =='V') return 5;
    if(ch =='X') return 10;             if(ch =='L') return 50;
    if(ch =='C') return 100;    if(ch =='D') return 500;
    return 1000;
}

int roman_to_int(string roman) {
    int i, num = 0, len = roman.size()-1;
    for(i=0;i<len;i++)
    {
        if(value(roman[i]) >= value(roman[i+1]))
            num += value(roman[i]);
        else
                num -= value(roman[i]);
    }
    num += value(roman[i]);
    return num;
}
```

int_to_roman

```
string int_to_roman(int num) {
        // bool valid = (num <= (4999||3999) && int_to_roman() ==
roman_to_int()
    string roman[] = //Largest integer possible 4999, some people
3999
    { "","I","II","III","IV","V","VI","VII","VIII","IX"
//1,2,3,4,..
        ,"X","XX","XXX","XL","L","LX","LXX","LXXX","XC"
//10,20,30,..
        ,"C","CC","CCC","CD","D","DC","DCC","DCCC","CM"
//100,200,300
        ,"M","MM","MMM","MMMM"  //1000,2000...
    };  //2222=2000+200+20+2 = MMCCXXII // 4444=MMMMCDXLIV

    string roman_number = "";
```

```
    int i, j, arr[50] = {0|;

    for(i=0; num; i++)  //cut it to thousands, hundreds, tens..
    {
        if(num%10 != 0) arr[i] = i*9 + (num%10);
        num /= 10;
    }

    for(j=0;j<i;j++)
        roman_number = roman[arr[j]] + roman_number;

    return roman_number;
}
```

grayCode

```
int inverseGray(int n) {
   int ish = 1, ans = n;

   while(true) {
           int idiv = ans >> ish;
           ans ^= idiv;
           if (idiv <= 1 || ish == 32)  return ans;
           ish <<= 1;
   }

}

void grayCode(int n) {
        lp(i, 1<<n)
                cout<<(i^(i>>1))<<"\n";
}
```

stirling1

```
// number of permutations of n elements with k permutation
cycles.
// E.g perm(1, 2, 3, 4) = 2, 1, 4, 3  has 2 cycles. {1, 2} , {3,
4}
ll stirling1(ll n, ll k) {
        if(k == 0)      return n == k;
        if(n == 0)      return 0;
        return          (n-1) *  stirling1(n-1, k) + stirling1(n-
1, k-1);
}
```

stirling2

```
// number of ways to partition a set of n elements into k groups.
// E.g. set{1, 2, 3, 4, 5} can be partioned to {1, 3, 5} {2, 4}
ll stirling2(ll n, ll k) {
        if(n == k || k == 1)    return 1;
```

```
        return          k * stirling2(n-1, k) + stirling2(n-1, k-
1);
}
```

num_digits_of_n_combination_k

```
int num_digits_of_n_combination_k(int n, int k) {
    double comb = 0;
    if(k > n/2) k = n-k;
    int i, j = k;
    for(i=n;i>n-k;i--) {
        comb += log10(i);
        for(;j>0;j--) {
            if(comb < 0)                    break;
                comb -= log10(j);
        }
    }
    return (int)(floor(comb)+1);
}
```

CONTEST STRATEGY

REGIONALS:

1- Sort problem set by length and assign to members starting with the fastest

2- Read the problem CAREFULLY, if it is an ACE code it directly on PC and go to step 8

3- Give yourself 5 minutes of thinking even if the problem is hard, you only need to understand the problem statement very well and think in a solution if possible

4- Describe the problem to the person who is better at the problem area, whom should listen very carefully and make sure he understands the problem very well,

5- This small meeting should decide one of the following: 1-the problem should be delayed 2-you should write the solution you came up with 3-you both stay for sometime thinking in a solution 4-you only should stay for sometime thinking in a solution

6- If you both decided that this problem is to be solved, the better of the two at the problem area will read the problem carefully (if not yet) write code on paper and get approval from the backup that the code is COMPLETE

7- Once the PC is free, copy ur code there, make sure you copy the input correctly from the problem statement and debug for the first 10 minutes to match the sample output, if more debugging is needed the backup should join for another 10 minutes, if still print and debug on paper

8- if you submit and got WA or TLE or RTA review the checklist, read the problem again, debug on paper or whatever for another 20 minutes, if u found bug(s)interrupt the man on the PC, write the testcase u suspect, run and make sure u get WA, then take backup of the code apply ur fix, run and make sure the output is correct and submit

9- if the offline debug took 20 minutes the backup should read the problem and the code and spend 10 minutes with you, if u couldn't get it leave the problem immediately and get back to it later

10- In the last hour don't start a new problem (unless you've no wrong submissions), sort problems by most solved and for each wrong submission the author and the backup (and the third if he isn't doing anything) should debug it.

HINTS FOR THE CONTEST

Hints: -Compete with problemset instead of team, use score board only to know which problems are solved

WA bugs:

-CHECK THE SPELLING OF OUTPUT STRINGS (Specillay s for plural and case sensitivity)-Repeat sample input cases in reverse order -Read the problem again, specially the input and output -Make sure you correctly initialize between testcases -Math operations like mod, floor and ceil works differentelly on positive and negative -Multiple edges between two nodes -Multiple spaces between input words -truncate or apporximate -double issues, watch for -0.0 (if the double is near than zero output zero) and don't use (==, <, >) directly -Multiple input items (same string in the input twice), use set or multiset -Input terminating condition and output format must equal to what the problem specified -Copy input correctly from problem statements -watch for special cases in the input -Integer and char overflow (multiplications & powers& Cross Products)!! -Make use you don't use a very large infinity and add things to it which may cause overflow -If you've a double and want to convert it to integer (multiply by 100000 or so), then add EPS first as the double 0.7 may be stored as. Watch out: "Input is a 32 integer bit" int x; cin>>x; if(x<0) x = -x; do(x); OVERFLOW: -2^31 should not be positived in int var.

0.69999999 -HashSet and HashMap don't sort, TreeSet and TreeMap do (C++ set and map are tree-based) -If the problem can be DPed then do it this way (safer than greedy) -After all, you may have got the problem the wrong way, let a fresh member read it and hear from him (don't affect him) -not a number(NAN) which comes from sqrt(-ve), (0/0) ,or acos(1.00000000001) or cos(-1.000000000001) for such case if the value is very close

to -1 or 1 make it 1. - reading by scanf("%d ",x) to remove '\n' can remove leading spaces on the next line

---

TLE bugs: -Note that Choosing all combination of N items is of order 2^N using recursion and N*(2^N) if using bitmasked loop -Use scanf instead of cin if u got TLE -avoid division, mod and multiplicatio operations if u got TLE -If the problem is DP, make sure you are using the smallest possible number of dimensions for the DP -incorrect input reading/termination (watch for empty lines)

Runtime bugs: -Index out of boundaries -Stack over flow -integer division by zero -Calling Integer.pareseInt with invalid string- incorrect input reading (getline)- empty lines in input

Presentataion error=Output formmat error: 1) Watch out diplayed lists 1 3 7 9 Do not display SPACE after last number(here 9)

2) Make sure from sepreating testscases. 2.1) Display blank line after each test case means there is a line between each test case even after the last test case. 2.2) Display blank line between test casse. --> Means ONLY between testcases

3) In C++ : memcpy and memset don't work normally with very large arrays

---

1- first hour is the hunt for ACES, don't interrupt the team members too much in this hour. if there is an interruption it should be for asking about something not for thinking with you in the idea.

2- the ACE problem is the addition, multiplication or sorting problem such that it's not harder than Div2-250 or the lines of code doesn't exceed 20 lines. **it's a must that the problem doesn't need the strategy and the problem can be solved inside the main**.

3- read the problem statement till the end, take your time to check the input and the output, and take care that the sample input and output may have the key to the problem solution.

4- make your code small, simple, smart

---

contest scenario: In the first hour do the following:

1- no interrupts, hunting for aces, and reading problems as much as you can.

2- read the problems to the end including the input and the output section and put a rough estimate for the problem.

3- never not to complete reading a problem to the end.

starting from secind hour:

1- all problems codes must be written on papers.

2- the written code should be written in a clean way.

3- the code should be scanned from the papers to the machine and compile.

starting from the third hour:

1- the score board is a good guide to see which problems you should solve.

2- schedule for the next 2 hours which problems to start with and which to delay.

in the last hour:

1- do not start coding a problem in the last hour unless you got accepted in all the other tried problems.

2- do your best to solve all the written problems.

«

WHY WRONG ANSWER

-CHECK THE SPELLING OF OUTPUT STRINGS (Specially s for plural and case sensitivity)

-Repeat sample input cases in reverse order

-Compete with problem set instead of team, use score board only to know which problems are solved

-Read the problem again, specially the input and output

-Make sure you correctly initialize between test cases

-Multiple edges between two nodes

-Multiple spaces between input words

-truncate or approximate

-double issues, watch for -0.0 (if the double is near than zero output zero) and don't use (==, <, >) directly

-Multiple input items (same string in the input twice), use set or multiset

-Input terminating condition and output format must equal to what the problem specified

-Copy input correctly from problem statements

-Watch for special cases in the input

-Integer and char overflow!!

-Make sure you don't use a very large infinity and add things to it which may cause overflow (E.g. in DP)

-Small infinity may be wrong (if it smaller than what u calc)

-overflow: multiplications(cross product) & powers & Base conversions & DP counting problems.

-Check CAREFULLY input stopping conditions. E.g. Input terminate with line START with # or CONTAINS #

-If you've a double and want to convert it to integer (multiply by 100000 or so), then add EPS first as the double 0.7 may be stored as 0.69999999

-HashSet and HashMap don't sort, TreeSet and TreeMap do (C++ set and map are tree-based)

-If the problem can be DPed then do it this way (safer than greedy)

-After all, you may have got the problem the wrong way, let a fresh member read it and hear from him (don't affect him)

-not a number(NAN) which comes from sqrt(-ve), (0/0) ,or cos(1.00000000001) or cos(-1.000000000001) for such case if the value is very close to -1 or 1 make it 1.

-make sure when u are flooring a -ve integer that u floor it to the nearest less integer, for example Floor(-2.3) = -3 but Floor(2.3)= 2.

- Other tricks:

- Word is "sequence of upper/lower case letters". then ali is 1 word, X-Ray is 2 words

- You will operate on string of letters (this do not mean Latin letters a-z, this is bigger)

- Given 2 integers i, j, find number of primes between them. input may be 10 20 OR 20 10

- Given N*M grid, Read N lines each start with M chars. E.g. 3*2

-- 1st line -> ab

-- 2nd line -> cdEXTRA // use to depend on read N, M, as RE may happen

-- 3rd line -> ef

- In multiset insert add new element, but delete removes ALL instances of element

-- if multiset contains (3 3 3 3 6 9) and u delete 3 -->will be (6, 9)

- Use to read input then process it, if u did not, do not BREAK wrongly while reading.

--lp(i, 5) { cin>>x; if(!valid(x)) { ok = 0; break;} --> What about output REMINDER?

- Geometry: Is polygon simple, convex, concave? Is there duplicate points? Does it matter?

- if you are using double the maximum eps you can use is 1e-11, if you need more precision you have to use long double instead.

- if the output is longlong make sure that you use cout not printf