1 TEMPLATE

```
#include <bits/stdc++.h>

using namespace std;

typedef long long     LL;
typedef pair<int,int> pii;

double PI  = acos(-1);
double EPS = 1e-7;
int INF    = 1000000000;
LL INFLL   = 1000000000000000000LL;

#define fi            first
#define se            second
#define mp            make_pair
#define pb            push_back

#define input(in)     freopen(in,"r",stdin)
#define output(out)   freopen(out,"w",stdout)

#define MIN(a, b)     (a) = min((a), (b))
#define MAX(a, b)     (a) = max((a), (b))

#define RESET(a, b)   memset(a,b,sizeof(a))
#define ALL(a)        (a).begin(), (a).end()
#define SIZE(a)       (int)a.size()
#define SORT(a)       sort(ALL(a))
#define UNIQUE(a)     (a).erase( unique( ALL(a) ),
(a).end() )
#define FOR(a, b, c)  for (int (a)=(b); (a)<=(c); (a)++)
#define FORD(a, b, c) for (int (a)=(b); (a)>=(c); (a)--)
#define FORIT(a, b)   for (__typeof((b).begin())
a=(b).begin(); a!=(b).end(); a++)

int mx[8] = {-1,1,0,0,-1,-1,1,1};
int my[8] = {0,0,-1,1,-1,1,-1,1};

// ----- //

int main()
{

}
```

2 DATA STRUCTURE

2.1 BALANCED BINARY SEARCH TREE (AVL TREE)

Source : Self

```
struct node
{
      int height,value,size;
      node *l,*r;
};
struct AVL
{
      node *root;
      AVL()
      {
            root = NULL;
      }
      int height(node *cur)
      {
            if (cur == NULL) return 0;
            else return cur->height;
      }
      int size(node *cur)
      {
            if (cur == NULL) return 0;
            else return cur->size;
      }
      int size()
      {
```

```
            return size(root);
      }
      void update(node *cur)
      {
            if (cur == NULL) return;
            cur->height = 1 + max(height(cur->l),
height(cur->r));
            cur->size = 1 + size(cur->l) + size(cur->r);
      }
      node* left_rotate(node *cur)
      {
            node* tmp = cur->l;
            cur->l = tmp->r;
            tmp->r = cur;
            update(cur);
            update(tmp);
            return tmp;
      }
      node* right_rotate(node *cur)
      {
            node* tmp = cur->r;
            cur->r = tmp->l;
            tmp->l = cur;
            update(cur);
            update(tmp);
            return tmp;
      }
      node* balance(node *cur)
      {
            if (cur == NULL) return cur;
            if (height(cur->l)-height(cur->r) == 2)
            {
                  node *tmp = cur->l;
                  if (height(tmp->l)-height(tmp->r) == -
1)
                  {
                        cur->l = right_rotate(tmp);
                  }
            return left_rotate(cur);
      }

      if (height(cur->l)-height(cur->r) == -2)
      {
            node *tmp = cur->r;
            if (height(tmp->l)-height(tmp->r) == 1)
            {
                  cur->r = left_rotate(tmp);
            }

            return right_rotate(cur);
      }
      update(cur);
      return cur;
}
node* insert(node *cur,int k)
{
      if (cur == NULL)
      {
            cur = new node;
            cur->l = cur->r = NULL;
            cur->height = 1;
            cur->value = k;
            cur->size = 1;
            return balance(cur);
      }
      else
      {
            if (k < cur->value)
            {
                  cur->l = insert(cur->l,k);
            }
            if (k > cur-> value)
            {
                  cur->r = insert(cur->r,k);
            }
            return balance(cur);
```

```
        }
    }
    void insert(int k)
    {
        root = insert(root,k);
    }

    node* erase(node *cur,int k)
    {
        if (cur == NULL) return cur;
        if (cur->value == k)
        {
            if (cur->l == NULL || cur->r == NULL)
            {
                node* tmp = cur->l;
                if (tmp == NULL) tmp = cur->r;
                delete cur;
                return balance(tmp);
            }
            else
            {
                node* tmp = cur->r;
                while(tmp->l)
                {
                    tmp = tmp->l;
                }
                cur->value = tmp->value;
                cur->r = erase(cur->r,tmp-
>value);
                return balance(cur);
            }
        }
        if (cur->value > k)
        {
            cur->l = erase(cur->l,k);
        }
        if (cur->value < k)
        {
            cur->r = erase(cur->r,k);
        }
        return balance(cur);
}
void erase(int k)
{
    root = erase(root,k);
}

int rank(node* cur,int k)
{
    if (cur == NULL) return 0;
    if (cur->value <= k)
        return size(cur->l)+1+rank(cur->r,k);
    else
        return rank(cur->l,k);
}
int rank(int k)
{
    return rank(root,k);
}
void preorder(node *cur)
{
    if (cur==NULL) return;
    preorder(cur->l);
    printf("%d ",cur->value);
    preorder(cur->r);
}
void preorder()
{
    preorder(root);
    cout << endl;
}
int kth(node* cur,int k)
{
    if (size(cur->l) >= k) return kth(cur->l,k);
    if (size(cur->l)+1 == k) return cur->value;
    return kth(cur->r,k-size(cur->l)-1);
```

```
	}
	int kth(int k)
	{
		return kth(root,k);
	}
};
```

3 TECHNIQUE

3.1 HEAVY LIGHT DECOMPOSITION

Source :
http://apps.tøpcoder.com/forums/?module=Thread&threadID=796128&start=0&mc=8

```
const int V = 100000;
vector<int> adj[V];     // adjacency list
int parent[V], heavy[V];
int depth[V], size[V];
int chain[V], head[V];
//Where chain[u] is u's chain number and head[u] is the
node closest to root in u's chain.

void DFS(int i)
{
    size[i] = 1;
    for (int k=0; k<adj[i].size(); ++k)
    {
        int j = adj[i][k];
        if (j == parent[i]) continue;

        parent[j] = i;
        depth[j] = depth[i] + 1;

        DFS(j);

        size[i] += size[j];
        if (heavy[i] == -1 || size[j] > size[heavy[i]])
            heavy[i] = j;
    }
}

void heavylight_DFS(int N)
{
    memset(heavy, -1, sizeof(heavy));

    parent[0] = -1;
    depth[0] = 0;
    DFS(0);

    int c = 0;
    for (int i=0; i<N; ++i)
        if (parent[i] == -1 || heavy[parent[i]] != i)
        {
            for (int k = i; k != -1; k = heavy[k])
                chain[k] = c, head[k] = i;
            c++;
        }
}

int q[V], *qf, *qb; // BFS queue

void heavylight_BFS(int N)
{
    qf = qb = q;
    parent[0] = -1;
    depth[0] = 0;
    *qb++ = 0;
    while (qf < qb)
        for (int i=*qf++, k=0; k<adj[i].size(); ++k)
        {
            int j = adj[i][k];
            if (j == parent[i]) continue;
            parent[j] = i;
            depth[j] = depth[i] + 1;
            *qb++ = j;
        }
```

```
    memset(size, 0, sizeof(size));
    memset(heavy, -1, sizeof(heavy));
    for (int k=N-1; k>0; --k)
    {
        int j = q[k], i = parent[q[k]];
        size[j]++;
        size[i] += size[j];
        if (heavy[i] == -1 || size[j] > size[heavy[i]])
            heavy[i] = j;
    }

    int c = 0;
    for (int i=0; i<N; ++i)
        if (parent[i] == -1 || heavy[parent[i]] != i)
        {
            for (int k = i; k != -1; k = heavy[k])
                chain[k] = c, head[k] = i;
            c++;
        }
}

int lca_1(int i, int j)
{
    while (chain[i] != chain[j])
        if (depth[head[i]] > depth[head[j]])
            i = parent[head[i]];
        else
            j = parent[head[j]];

    return depth[i] < depth[j] ? i : j;
}

int lca_2(int i, int j)
{
    while (chain[i] != chain[j])
    {
        if (depth[head[i]] > depth[head[j]])
            swap(i, j);
        j = parent[head[j]];
    }

    if (depth[i] > depth[j])
        swap(i, j);
    return i;
}

void look_inside(int N) {
     int i;

     printf("\n");
     printf("HEAVY: \n");
     printf("(i, j): i----(heavy edge)----j\n\n");

     for (i = 0; i < N; i++)
           printf("(%d, %d)\n", i, heavy[i]);

     printf("\n");
     printf("CHAIN: \n");
     printf("(i, j): Node i is in group (heavy-path
group) number j\n\n");

     for (i = 0; i < N; i++)
           printf("(%d, %d)\n", i, chain[i]);

     printf("\n");
     printf("HEAD: \n");
     printf("(i, j): Node i goes up all the way to the
highest node (j) which is in the same group\n\n");

     for (i = 0; i < N; i++)
           printf("(%d, %d)\n", i, head[i]);
}

int main() {
     int N, i, j;
```

```
	FILE *fin = fopen("input.txt", "r");

	fscanf(fin, "%d", &N);

	for (i = 0; i < N; i++)
		adj[i].clear();
	while (fscanf(fin, "%d%d", &i, &j) != EOF) {
		adj[i].push_back(j);
		adj[j].push_back(i);
	}

	//heavylight_DFS(N);
	heavylight_BFS(N);

	//printf("%d\n", lca_2(12, 16));

	//printf("%d\n", lca_2(16, 12));

	//printf("%d\n", lca_2(0, 7));

	//printf("%d\n", lca_2(0, 24));

	printf("%d\n", lca_1(6635, 8590));

	//look_inside(N); //I just added it into this
program in order to understand more about how it works
}
```

4 GRAPH

4.1 STRONGLY CONNECTED COMPONENT (TARJAN)

Source : http://www.jeslev.com/2014/02/dfs.html

```
vector<int> G[nro_nodes];
int vis[nro_nodes], comp[nro_nodes],stck[nro_nodes],
high[nro_nodes];
int t, num, ncomp;

void dfscc(int u){
  high[u] = vis[u] = num--;
  stck[t++] = u;
  for(int i=0;i<G[u].size();i++){
    int v = G[u][i];
    if(vis[v]==0){
       dfscc(v);
       high[u] = max(high[u], high[v]);
    }
    else if(vis[v]> vis[u] && comp[v]==0) high[u] =
max(high[u], vis[v]);
  }
  if(vis[u] == high[u]){
    ncomp++;
    do{
      int v = stck[--t];
      comp[v] = ncomp;
    }while(u != v);
  }
}

void tarjan(){
  memset(vis,0,sizeof(vis));
  memset(comp,0,sizeof(comp));
  ncomp = t = 0; num = nro_nodes;
  for(int i=0;i<nro_nodes;i++){
    if(vis[i]==0) dfscc(i);
  }
}
```

4.2 ARTICULATION POINT AND BRIDGE FINDING (TARJAN)

Source : http://www.jeslev.com/2014/02/dfs.html

```
void dfsbcc(int u,int p=-1){
  low[u] = vis[u] = ++t;
  int ch = 0;
  for(int i=0;i<E[u].size();i++){
    int e= E[u][i];
    int v = (from[e]==u)?to[e]:from[e];
    if(vis[v] == 0){
```

```
      stck[dt++] = e;
      dfsbcc(v,u);
      low[u] = min(low[u], low[v]);
      ch++;
      if(low[v]>=vis[u]){
        part[u]=1;
        nbcc++;
        do{
          int x =stck[--top];
          comp[x] = nbcc;
        }while(u!=v);
      }
      if(low[u]==vis[u])  bridge[e]=1;
    }
    else if(v!=p && vis[v]<vis[u]){
      stck[dt++] = e;
      low[u] = min(low[u],vis[v]);
    }
  }
  return ch;
}

void bcc(){
  memset(vis,0,sizeof(vis));
  memset(comp,0,sizeof(comp));
  memset(part,0,sizeof(part));
  memset(bridge,0,sizeof(brige));
  memset(low,0,sizeof(low));
  memset(stck,0,sizeof(stck));
  t = 0; dt=0; nbcc=0;
  for(int i=0;i<nro_nodes;i++){
    if(vis[i]==0) part[i] = dfsbcc(i)>=2;
  }
}

main(){
  /**
   cin>>u>>v;
    E[u].pb(current_edge);
    E[v].pb(current_edge);
    from[current_edge] = u;
    to[current_edge] = v;

  */
  bcc();
}
```

4.3 Maximum Flow (Dinic's)

Source : Self

```
const int MAXN = 490005;

struct Edge
{
    int to,rev,flow,cap;
};

vector<Edge> adjFlow[MAXN];

struct Dinic
{
    int n,s,t;
    int dist[MAXN],cur[MAXN];


    void AddEdge(int u,int v,int cap)
    {
        Edge i = {v,adjFlow[v].size(),0,cap};
        Edge j = {u,adjFlow[u].size(),cap,cap}; /*
directed   flows */
        //Edge j = {u,adjFlow[u].size(),0,cap}; /*
undirected flows */
        adjFlow[u].pb(i);
        adjFlow[v].pb(j);
    }

    Dinic(int _n,int _s,int _t)
    {
```

```
        n = _n;
        s = _s;
        t = _t;
        FOR(a,1,n) adjFlow[a].clear();
    }

    bool BuildLevelGraph()
    {
        fill(dist+1,dist+n+1,-1);
        queue<int> q;
        q.push(s),dist[s] = 0;

        while(!q.empty())
        {
            int u = q.front();
            q.pop();
            FOR(a,0,SIZE(adjFlow[u])-1)
            {
                Edge &nx = adjFlow[u][a];
                if (dist[nx.to] == -1 && nx.flow <
nx.cap)
                {
                    dist[nx.to] = dist[u]+1;
                    q.push(nx.to);
                }
            }
        }

        return (dist[t] != -1);
    }

    int BlockingFlow(int u,int f)
    {
        if (u == t) return f;
        FOR(a,cur[u],SIZE(adjFlow[u])-1)
        {
            cur[u] = a;
            Edge &nx = adjFlow[u][a];
            if (dist[nx.to] == dist[u]+1 && nx.flow <
nx.cap)
            {
                int ret =
BlockingFlow(nx.to,min(f,nx.cap-nx.flow));
                if (ret > 0)
                {
                    nx.flow += ret;
                    adjFlow[nx.to][nx.rev].flow -= ret;
                    return ret;
                }
            }
        }
        return 0;
    }

    int MaxFlow()
    {
        int res = 0;
        while(BuildLevelGraph())
        {
            fill(cur+1,cur+n+1,0);
            while(int ret = BlockingFlow(s,INF))
                res += ret;
        }
        return res;
    }
};
```

4.4 Maximum Matching Bipartite Graph (Hopcraft Karp)

Source : https://sites.google.com/site/indy256/algo_cpp/hopcroft_karp

```
#include <algorithm>
#include <iostream>

using namespace std;

const int MAXN1 = 50000;
const int MAXN2 = 50000;
```

```
const int MAXM = 150000;

int n1, n2, edges, last[MAXN1], prev[MAXM], head[MAXM];
int matching[MAXN2], dist[MAXN1], Q[MAXN1];
bool used[MAXN1], vis[MAXN1];

void init(int _n1, int _n2) {
    n1 = _n1;
    n2 = _n2;
    edges = 0;
    fill(last, last + n1, -1);
}

void addEdge(int u, int v) {
    head[edges] = v;
    prev[edges] = last[u];
    last[u] = edges++;
}

void bfs() {
    fill(dist, dist + n1, -1);
    int sizeQ = 0;
    for (int u = 0; u < n1; ++u) {
        if (!used[u]) {
            Q[sizeQ++] = u;
            dist[u] = 0;
        }
    }
    for (int i = 0; i < sizeQ; i++) {
        int u1 = Q[i];
        for (int e = last[u1]; e >= 0; e = prev[e]) {
            int u2 = matching[head[e]];
            if (u2 >= 0 && dist[u2] < 0) {
                dist[u2] = dist[u1] + 1;
                Q[sizeQ++] = u2;
            }
        }
    }
}

bool dfs(int u1) {
    vis[u1] = true;
    for (int e = last[u1]; e >= 0; e = prev[e]) {
        int v = head[e];
        int u2 = matching[v];
        if (u2 < 0 || !vis[u2] && dist[u2] == dist[u1] +
1 && dfs(u2)) {
            matching[v] = u1;
            used[u1] = true;
            return true;
        }
    }
    return false;
}

int maxMatching() {
    fill(used, used + n1, false);
    fill(matching, matching + n2, -1);
    for (int res = 0;;) {
        bfs();
        fill(vis, vis + n1, false);
        int f = 0;
        for (int u = 0; u < n1; ++u)
            if (!used[u] && dfs(u))
                ++f;
        if (!f)
            return res;
        res += f;
    }
}

int main() {
    init(2, 2);

    addEdge(0, 0);
    addEdge(0, 1);
```

```
    addEdge(1, 1);

    cout << (2 == maxMatching()) << endl;
}
```

4.5 Minimum Cost Flow (Successive Shortest Path)

Source : Self

```
typedef int F;
typedef int C;
#define F_INF 1e+9
#define C_INF 1e+9
#define NUM 10005

int V;

vector<F> cap;
vector<C> cost;
vector<int> to,prev;

C dist[NUM];
int last[NUM],path[NUM];


struct mincostflow{
    mincostflow(int n){
        cap.clear();
        cost.clear();
        to.clear();
        prev.clear();
        V = n;
        FOR(a,1,V)
        {
            last[a] = -1;
        }
    }

    void addedge(int x, int y, F w, C c){
        cap.pb(w); cost.pb(c);  to.pb(y);
prev.pb(last[x]); last[x] = SIZE(cap)-1;
        cap.pb(0); cost.pb(-c); to.pb(x);
prev.pb(last[y]); last[y] = SIZE(cap)-1;
    }
    pair<F,C> SPFA(int s, int t){
        F ansf=0;
        C ansc=0;
        FOR(a,1,V) dist[a] = C_INF;
        FOR(a,1,V) path[a] = -1;
        deque <pair <C,int> > pq;
        dist[s] = 0;
        path[s] = -1;
        pq.push_front(mp(0,s));
        while(!pq.empty())
        {
            C d = pq.front().fi;
            int p = pq.front().se;
            pq.pop_front();
            if (dist[p] == d)
            {
                int e = last[p];
                while(e != -1)
                {
                    if (cap[e] > 0)
                    {
                        C nd = dist[p] + cost[e];
                        if (nd < dist[to[e]])
                        {
                            dist[to[e]] = nd;
                                path[to[e]] = e;
                            if (cost[e] <= 0)
                            {


pq.push_front(mp(nd,to[e]));
                            }
```

```
                            else
pq.push_back(mp(nd,to[e]));
                        }
                    }
                    e = prev[e];
                }
            }
        }
        if(path[t] != -1)
        {
            ansf = F_INF;
            int e = path[t];
            while(e != -1)
            {
                MIN(ansf,cap[e]);
                e = path[to[e^1]];
            }
            e = path[t];
            while(e != -1)
            {
                ansc += cost[e] * ansf;
                cap[e^1] += ansf;
                cap[e] -= ansf;
                e = path[to[e^1]];
            }
        }

        return mp(ansf,ansc);
    }
    pair <F,C> calc(int s, int t){
        F ansf=0;
        C ansc=0;
        while(1)
        {
            pair <F, C> p = SPFA(s,t);
            if(path[t] == -1) break;
            ansf += p.fi; ansc += p.se;
        }
        return mp(ansf,ansc);
    }
};
```

5 String Processing

5.1 Suffix Array

```
const int MAX_N = 100005;
char str [MAX_N];
int N, m, SA [MAX_N], LCP [MAX_N];
int x [MAX_N], y [MAX_N], w [MAX_N], c [MAX_N];

inline bool cmp (const int a, const int b, const int l) {
return (y [a] == y [b] && y [a + l] == y [b + l]); }

void Sort () {
    for (int i = 0; i < m; ++i) w [i] = 0;
    for (int i = 0; i < N; ++i) ++w [x [y [i]]];
    for (int i = 0; i < m - 1; ++i) w [i + 1] += w [i];
    for (int i = N - 1; i >= 0; --i) SA [--w [x [y [i]]]]
= y [i];
}

void DA () {
    ++N;
    for (int i = 0; i < N; ++i) x [i] = str [i], y[i] =
i;
    Sort ();
    for (int i, j = 1, p = 1; p < N; j <<= 1, m = p) {
        for (p = 0, i = N - j; i < N; i++) y [p++] = i;
        for (int k = 0; k < N; ++k) if (SA [k] >= j) y
[p++] = SA [k] - j;
        Sort ();
        FOR(a,0,MAX_N-1)
        {
            swap(x[a],y[a]);
        }
```

```
        for (p = 1, x [SA [0]] = 0, i = 1; i < N; ++i) x
[SA [i]] = cmp (SA [i - 1], SA [i], j) ? p - 1 : p++;
    }
    for (int i = 1; i < N; ++i) SA [i - 1] = SA [i]; --N;
}

void kasaiLCP () {
    for (int i = 0; i < N; ++i) c [SA [i]] = i;
    LCP [0] = 0;
    for (int i = 0, h = 0; i < N; ++i) if (c[i] > 0) {
            int j = SA [c [i] - 1];
            while (i + h < N && j + h < N && str [i + h]
== str [j + h]) ++h;
            LCP [c [i]] = h;
            if (h > 0) --h;
        }
}

void suffixArray () {
    m = 256;
    N = strlen (str);
    DA ();
    kasaiLCP ();
}
```

5.2 String Automaton (Aho-Corasick)

```
struct Node
{
     int id;
     vector<int> end;
     Node *fail;
     Node *next[26];
};

struct AC
{
     int num;
     void create(Node* &cur)
     {
          cur = new Node;
          cur->id = ++num;
          cur->end.clear();
          cur->fail = NULL;
          FOR(a,0,25)
               cur->next[a] = NULL;
     }

     Node *root;

     void reset(Node* &cur)
     {
          if (cur == NULL) return;
          FOR(a,0,25)
               reset(cur->next[a]);
          delete cur;
     }

     void reset()
     {
          reset(root);
     }

     AC()
     {
          root = NULL;
          num = 0;
          create(root);
     }

     void insert(int id,char s[],int pos,int len,Node*
&cur)
     {
          printf("%d %d %d\n",id,pos,cur->id);
          if (pos == len)
          {
```

```
                cur->end.pb(id);
                return;
        }
        int c = s[pos]-'A';
        if (cur->next[c] == NULL) create(cur-
>next[c]);
        insert(id,s,pos+1,len,cur->next[c]);
    }

    void insert(int id,char s[])
    {
        insert(id,s,0,strlen(s),root);
    }

    void build(Node* cur,Node* par,int c)
    {
        if (cur == NULL) return;

        if (cur == root);
        else if (par == root) cur->fail = par;
        else
        {
            cur->fail = par->fail;
            while(cur->fail->next[c] == NULL)
            {
                if (cur->fail == root) break;
                cur->fail = cur->fail->fail;
            }
            if (cur->fail->next[c] == NULL);
            else cur->fail = cur->fail->next[c];
        }
        FOR(a,0,25)
        {
            build(cur->next[a],cur,a);
        }
    }

    void build()
    {
        build(root,NULL,-1);
    }

    void print(Node *cur,int c)
    {
        if (cur == NULL) return;
        if (cur != root) printf("%d %d %c\n",cur-
>id,cur->fail->id,char('A'+c));
        FOR(a,0,25)
        {
            print(cur->next[a],a);
        }
    }

    void print()
    {
        cout << "PRINT" << endl;
        print(root,-1);
    }
};


char s[100005];

int main()
{
    int n;
    scanf("%d",&n);

    AC lol;
    FOR(a,1,n)
    {
        scanf("%s",&s);
        lol.insert(a,s);
    }
    lol.build();
}
```

5.3 KMP

```
int p[1000001];
char s1[1000001];
char s2[1000001];

int main()
{
	gets(s2);
	gets(s1);
	int n = strlen(s1);
	int m = strlen(s2);

	//KMP INIT
	int j = -1;
	p[0] = -1;

	// Build Table
	FOR(i,0,n-1)
	{
		while(j >= 0 && s1[i] != s1[j]) j = p[j];
		j++;
		p[i+1] = j;
	}

	j = 0;
	int found = 0;
	FOR(i,0,m-1)
	{
		while(j >= 0 && s2[i] != s1[j]) j = p[j];
		j++;
		if (j == n)
		{
			found++;
			j = p[j];
		}
	}
	printf("%d\n",found);
}
```

6 MATH

6.1 EXTENDED EUCLID

```
void extendedEuclid(int a,int b)
{
	if (b == 0)
	{
		x = 1;
		y = 0;
		d = a;
		return;
	}
	int x1 = y;
	int y1 = x - (a/b)*y;
	x = x1;
	y = y1;
}
```

6.2 MATRICES

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

typedef vector<int> vi;
typedef vector<vi> vvi;

const int mod = 1234567891;

vvi matrixUnit(int n) {
    vvi res(n, vi(n));
    for (int i = 0; i < n; i++)
        res[i][i] = 1;
    return res;
}

vvi matrixAdd(const vvi &a, const vvi &b) {
```

```
    int n = a.size();
    int m = a[0].size();
    vvi res(n, vi(m));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            res[i][j] = (a[i][j] + b[i][j]) % mod;
    return res;
}

vvi matrixMul(const vvi &a, const vvi &b) {
    int n = a.size();
    int m = a[0].size();
    int k = b[0].size();
    vvi res(n, vi(k));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < k; j++)
            for (int p = 0; p < m; p++)
                res[i][j] = (res[i][j] + (long long) a[i]
[p] * b[p][j]) % mod;
    return res;
}

vvi matrixPow(const vvi &a, int p) {
    if (p == 0)
        return matrixUnit(a.size());
    if (p & 1)
        return matrixMul(a, matrixPow(a, p - 1));
    return matrixPow(matrixMul(a, a), p / 2);
}

vvi matrixPowSum(const vvi &a, int p) {
    int n = a.size();
    if (p == 0)
        return vvi(n, vi(n));
    if (p % 2 == 0)
        return matrixMul(matrixPowSum(a, p / 2), matrixAd
d(matrixUnit(n), matrixPow(a, p / 2)));
    return matrixAdd(a, matrixMul(matrixPowSum(a, p - 1),
 a));
}

int main() {
    vvi a(2, vi(2));
    a[0][0] = 1;
    a[0][1] = 1;
    a[1][0] = 1;
    vvi b = matrixPow(a, 10);
}
```

6.3 Matrix Exponential

```
#define D 7
typedef long long I;
I mod;

I llmult(I a,I b) {  // O(1) for (a*b)%m
   long double res = a;
   res *= b;
   I c = I(res / mod);
   a *= b;
   a -= c * mod;
   a %= mod;
   if (a < 0) a += mod;
   return a;
}

struct matrixexp
{
    I init[D][D];
    I num[D];
    I mod;
    int N;
    matrixexp(int s,I m,I i[][D],I j[])
    {
        N = s;
        mod = m;
        FOR(a,0,N-1)
```

```
            FOR(b,0,N-1) init[a][b] =
((i[a][b])%mod+mod)%mod;
        FOR(a,0,N-1) num[a] = j[a];
    }

    inline void mult(I m1[][D],I m2[][D],I res[][D])
    {
        FOR(a,0,N-1)
        {
            FOR(b,0,N-1)
            {
                res[a][b] = 0;
                FOR(c,0,N-1)
                {
                    I tmp = llmult(m1[a][c],m2[c][b]);
                    res[a][b] += tmp;
                    if (res[a][b] >= mod) res[a][b] -=
mod;
                }
            }
        }
    }
    inline void exp(I k,I res[][D])
    {
        if (k==0)
        {
            FOR(a,0,N-1)
                FOR(b,0,N-1) res[a][b] = (a==b);
            return;
        }
        if ((k&1)==0)
        {
            exp(k/2,res);
            I ret[D][D];
            mult(res,res,ret);
            FOR(a,0,N-1)
                FOR(b,0,N-1) res[a][b] = ret[a][b];
        }
        else
        {
            exp(k-1,res);
            I ret[D][D];
            mult(res,init,ret);
            FOR(a,0,N-1)
                FOR(b,0,N-1) res[a][b] = ret[a][b];
        }
    }
    inline I calc(I k,int p)
    {
        I ret[D][D];
        exp(k,ret);
        I ans = 0;
        FOR(a,0,N-1)
        {
            I tmp = llmult(num[a],ret[a][p]);
            ans += tmp;
            if (ans >= mod) ans -= mod;
        }
        return ans;
    }
};
```

6.4 Gauss Jordan Elimination

```
int P[105][105];
int T[105][105];
double mat[105][105];

int main()
{
     int tc;
     scanf("%d",&tc);
     while(tc--)
     {
           int n;
```

```
		scanf("%d",&n);
		FOR(a,1,n)
		{
			FOR(b,1,n)
			{
				scanf("%d",&P[a][b]);
			}
		}
		FOR(a,1,n)
		{
			FOR(b,1,n)
			{
				scanf("%d",&T[a][b]);
				mat[a][b] = 0;
			}
		}
		if (n == 1)
		{
			printf("%.9lf\n",(double)0);
			continue;
		}
		FOR(a,1,n-1)
		{
			FOR(b,1,n-1)
			{
				mat[a][n] += P[a][b]*T[a][b];
				mat[a][b] = -P[a][b];
			}
			mat[a][n] += P[a][n]*T[a][n];
			mat[a][a] += 100;
		}
		FORD(a,n-1,2)
		{
			int pvt = a;
			FORD(b,a-1,1)
			{
				if (fabs(mat[pvt][a]) <
fabs(mat[b][a]))
				{
					pvt = b;
				}
			}
			FOR(b,1,n)
			{
				swap(mat[pvt][b],mat[a][b]);
			}
			FORD(b,a-1,1)
			{
				double rat =
(double)mat[b][a]/(double)mat[a][a];
				FOR(c,1,n)
				{
					mat[b][c] -=
mat[a][c]*rat;
				}
			}
		}

		printf("%.9lf\n",(double)mat[1][n]/(double)mat[1][1]
);
		}
}
```

7 GEOMETRY

7.1 GEOMETRY TEMPLATE

```
double degToRad(double d) {
	return d * PI / 180.0;
}
double radToDeg(double r) {
	return r * 180.0 / PI;
}

struct point {
	double x, y;
	point() {
```

```
		x = INF, y = INF;
	}
	point(double _x, double _y) {
		x = _x, y = _y;
	}
	bool operator <(point other) const {
		if (fabs(x - other.x) > EPS)
			return x < other.x;
		return y < other.y;
	}
};

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

struct line {
	double a, b, c;
};

void pointsToLine(point p1, point p2, line &l) {
	if (fabs(p1.x - p2.x) < EPS) // vertical line
		l.a = 1.0, l.b = 0.0, l.c = -p1.x;
	else {
		l.a = -(double) (p1.y - p2.y) / (p1.x -
p2.x);
		l.b = 1.0;
		l.c = -(double) (l.a * p1.x) - p1.y;
	}
}

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

vec toVector(point p1, point p2) {
	return vec(p2.x - p1.x, p2.y - p1.y);
}

vec scaleVector(vec v, double s) {
	return vec(v.x * s, v.y * s);
}

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

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

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

double distToLine(point p, point A, point B, point &c) {
	vec Ap = toVector(A, p), AB = toVector(A, B);
	double u = dot(Ap, AB) / norm_sq(AB);
	c = translate(A, scaleVector(AB, u));
	return dist(p, c);
}

double distToLineSegment(point p, point A, point B, point
&c) {
	vec Ap = toVector(A, p), AB = toVector(A, B);
	double u = dot(Ap, AB) / norm_sq(AB);
	if (u < 0.0) {
		c = point(A.x, A.y);
		return dist(p, A);
	}
	if (u > 1.0) {
		c = point(B.x, B.y);
		return dist(p, B);
```

```
     }
     return distToLine(p, A, B, c);
}
```

7.2 Graham Scan

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

typedef pair<double, double> point;

bool cw(const point &a, const point &b, const point &c) {
    return (b.first - a.first) * (c.second - a.second) -
(b.second - a.second) * (c.first - a.first) < 0;
}

vector<point> convexHull(vector<point> p) {
    int n = p.size();
    if (n <= 1)
        return p;
    int k = 0;
    sort(p.begin(), p.end());
    vector<point> q(n * 2);
    for (int i = 0; i < n; q[k++] = p[i++])
        for (; k >= 2 && !cw(q[k - 2], q[k - 1], p[i]); --k)
            ;
    for (int i = n - 2, t = k; i >= 0; q[k++] = p[i--])
        for (; k > t && !cw(q[k - 2], q[k - 1], p[i]); --k)
            ;
    q.resize(k - 1 - (q[0] == q[1]));
    return q;
}

int main() {
    vector<point> points(4);
    points[0] = point(0, 0);
    points[1] = point(3, 0);
    points[2] = point(0, 3);
    points[3] = point(1, 1);
    vector<point> hull = convexHull(points);
    cout << (3 == hull.size()) << endl;
}
```

7.3 Line Sweep Convex Hull

```
pll p[300001];
pll lo[300001];
pll hi[300001];

LL cross(pll O, pll A,pll B)
{
     return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) *
(B.x - O.x);
}

int main()
{
     LL n;
     scanf("%lld",&n);
     FOR(a,1,n)
     {
          scanf("%lld %lld",&p[a].x,&p[a].y);
     }
     sort(p+1,p+n+1);
     //lower
     LL clo = 0;
     FOR(a,1,n)
     {
          while (clo >=2 && cross(lo[clo-1], lo[clo],
p[a]) <= 0) clo--;
          lo[++clo] = p[a];
     }
     //upper
     LL chi = 0;
```

```
	FORD(a,n,1)
	{
		while (chi >=2 && cross(hi[chi-1], hi[chi],
p[a]) <= 0) chi--;
		hi[++chi] = p[a];
	}
	lo[++clo] = lo[1];
	hi[++chi] = hi[1];
	LL ar=0;
	FOR(a,1,clo-1)
	{
		ar += lo[a].x*lo[a+1].y;
		ar -= lo[a].y*lo[a+1].x;
	}
	FOR(a,1,chi-1)
	{
		ar += hi[a].x*hi[a+1].y;
		ar -= hi[a].y*hi[a+1].x;
	}
	if (ar < 0) ar *= -1;
	if (ar%2==0) printf("%lld.0\n",ar/2);
	else printf("%lld.5\n",ar/2);
}
```

8 EXTRA

8.1 JAVA BIGINTEGER

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


public class coba
{
	public static void main(String[] args)
	{
		Scanner in = new Scanner(System.in);
		BigInteger A = in.nextBigInteger();
		BigInteger B = in.nextBigInteger();
		A = A.add(B);
		System.out.println(A);
	}
}
```

9 SCRIPT

9.1 LINUX

```
#!/bin/bash

NAME=$1

/usr/bin/g++ -Wno-deprecated-declarations -Wno-unused-
result -DTEST -O2 -o $NAME $NAME.cpp
```

9.2 WINDOWS

```
@echo off
g++ -static -fno-optimize-sibling-calls -fno-strict-
aliasing -lm -s -x c++ -Wl,--stack=268435456 -O2 -o %1
%1.cpp
```

9.3 WINDOWS (C++ 11)

```
@echo off
g++ -static -fno-optimize-sibling-calls -fno-strict-
aliasing -lm -s -x c++ -Wl,--stack=268435456 -O2 -
std=c++11 -o %1 %1.cpp
```