

● string.cpp

```
int* z_function(char s[],int n)
{
    int* z = new int[n];
    memset(z, 0, sizeof(z));
    z[0] = n;
    int L = 0, R = 1;
    for (int i=1; i<n; ++i)
        if (R <= i || z[i-L] >= R-i){
            int x = (R <= i ? i : R);
            while (x < n && s[x] == s[x-i]) x++;
            z[i] = x-i;
            if (i < x) {L = i; R = x;}
        }
        else
            z[i] = z[i-L];
    return z;
}
/*banana*/
void IBWT(){
    vector<int> index[256];
    for(int i=0; i<N; i++) index[t[i]].push_back(i);

    for(int i=0, n=0; i<256; i++)
        for(int j=0; j<index[i].size(); j++)
            next[n++] = index[i][j];
}
/*kmp*/
for(int i=0, j=-1; i<t.size(); i++){
    while(j>=0 && p[j+1]!=t[i]) j = f[j];
    if(p[j+1]==t[i]) j++;
    if(j==p.size()-1){
        int ans = i - p.size();
        j = f[j];
    }
}
```

● 最小字典序表示法

```
void solve(){
    scanf("%s",t);
    s[0] = '\0';
    strcat(s, t);
    strcat(s+n, t);

    int j = 1, i = 0;
    while(i<n && j<n){
        if(s[j]<s[i]) i = j, j = i+1;
        else if(s[j]>s[i]) j++;
        else{
            int k = 0;
            while(k<n){
                if(s[i+k]==s[j+k]) k++;
                else if(s[i+k]<s[j+k]){
                    j = j+k + 1;
                    break;
                }
            }
            else{
                i = j;
                j = i + 1;
                break;
            }
        }
        if(k==n) break;
    }
    printf("%d\n", i);
}
```

● sa.cpp

```
int d2[maxn], d[maxn];
int ra[maxn], he[maxn], sa[maxn], c[maxn];
void build_sa(int n,int m){
    int *x = ra, *y = he;

    for(int i=0; i<m; i++) c[i] = 0;
    for(int i=0; i<n; i++) c[x[i]=d[i]]++;
    for(int i=1; i<m; i++) c[i] += c[i-1];
    for(int i=n-1; i>=0; i--) sa[--c[x[i]]] = i;

    for(int k=1; k<=n; k<=1){
        int p = 0;
        for(int i=n-k; i<n; i++) y[p++] = i;
        for(int i=0; i<n; i++) if(sa[i]>=k) y[p++] = sa[i]-k;

        for(int i=0; i<m; i++) c[i] = 0;
        for(int i=0; i<n; i++) c[x[y[i]]]++;
        for(int i=1; i<m; i++) c[i] += c[i-1];
        for(int i=n-1; i>=0; i--) sa[--c[x[y[i]]]] = y[i];

        swap(x, y);
        p = 0;
        x[sa[0]] = p++;
        for(int i=1; i<n; i++)
            x[sa[i]] = y[sa[i]]==y[sa[i-1]]&&sa[i]+k<n&&sa[i-1]+k<n&&y[sa[i]+k]==y[sa[i-1]+k]?
            p-1:p++;
        if(p>=n) break;
        m = p;
    }
}
void build_he(int n){
    for(int i=0; i<n; i++) ra[sa[i]] = i;
    // def he[i] = lcp(sa[i], sa[i-1])
    // --> he[ra[i]]>=he[ra[i-1]]-1
    he[0] = 0;
    for(int i=0,k=0; i<n; i++){
        if(ra[i]){
            if(k) k--;
            int j = sa[ra[i]-1];
            while(d[i+k]==d[j+k] && i+k < n && j+k < n) k++;
            he[ra[i]] = k;
        }
    }
}
```

● AC.cpp

```
const int maxn = 100;
const int maxkind = 26;
const int maxlen = 100;
const int maxsize = maxn*maxlen + 10;
struct AC{
    int ch[maxsize][maxkind], f[maxsize],
    last[maxsize], val[maxsize];
    int root, memid;
    AC(){ clear(); }
    void newNode(){
        memset(ch[memid], 0, sizeof(ch[memid]));
        f[memid] = last[memid] = val[memid] = 0;
        return memid++;
    }
    void clear(){
        memid = 0;
        root = newNode();
    }
    void insert(const char* s,int v){
        int tmp = root;
        for(int i=0; s[i]; i++){
            int id(ID[s[i]]);
            if(!ch[tmp][id]) ch[tmp][id] = newNode();
            tmp = ch[tmp][id];
        }
    }
}
```

```

    val[tmp] = v;
}
void getfail(){
    queue<int> Q;
    f[root] = 0;
    for(int i=0; i<maxkind; i++) if(ch[root][i]){
        int u = ch[root][i];
        f[u] = last[u] = 0;
        Q.push(u);
    }
    while(!Q.empty()){
        int x = Q.front(); Q.pop();
        for(int i=0; i<maxkind; i++) if(ch[x][i]){
            int tmp = f[x], u = ch[x][i];
            while(tmp && !ch[tmp][i]) tmp = f[tmp];
            f[u] = ch[tmp][i];
            last[u] = val[f[u]]? f[u]:last[f[u]];
            Q.push(u);
        }
    }
}
void find(const char *s){
    int tmp = root;
    for(int i=0; s[i]; i++){
        int id = ID(s[i]);
        while(tmp && !ch[tmp][id]) tmp = f[tmp];
        tmp = ch[tmp][id];
        if(val[id])// find
        if(last[id]) //find
    }
}
};

```

● Hash.cpp

```

#define MAXN 1000000
#define prime_mod 1073676287
typedef long long T;
char s[MAXN+5];
T h[MAXN+5];
T h_base[MAXN+5];
inline void hash_init(int len,T prime=0xdefaced){
    h_base[0]=1;
    for(int i=1;i<=len;++i){
        h[i]=(h[i-1]*prime+s[i-1])%prime_mod;
        h_base[i]=(h_base[i-1]*prime)%prime_mod;
    }
}
inline T get_hash(int l,int r){
    return (h[r+1]-(h[l]*h_base[r-
l+1])%prime_mod+prime_mod)%prime_mod;
}

```

● Treap.cpp

```

#include <iostream>
#include <algorithm>
#include <cstdio>
using namespace std;
const int INF = 9e9;
struct Node{
    int val, pri, size, mi, tag;
    bool rev;
    Node *l, *r;
    Node(){
        Node(int
v):val(v),pri(rand()),size(1),rev(0),mi(v),tag(0){ l =
r = NULL;}
    void down();
    void up();
}*root;

int Size(Node *o){ return o? o->size:0;}
int Min(Node *o){ return o? o->mi:INF;}
int Val(Node *o){ return o? o->val:-1;}

```

```

void Node::down(){
    if(tag){
        val += tag;
        mi += tag;
        if(l) l->tag += tag;
        if(r) r->tag += tag;
        tag = 0;
    }
    if(rev){
        swap(l,r);
        if(l) l->rev ^= 1;
        if(r) r->rev ^= 1;
        rev = 0;
    }
}

void Node::up(){
    if(l) l->down();
    if(r) r->down();
    size = 1 + Size(l) + Size(r);
    mi = std::min( min(Min(l), Min(r)),val );
}

void print(Node *o){
    if(o){
        print(o->l);
        printf("%d ", o->val);
        print(o->r);
    }
}

Node* merge(Node* a,Node *b){
    if(!a || !b) return a? a:b;
    if(a->pri < b->pri){
        a->down();
        a->r = merge(a->r, b);
        a->up();
        return a;
    }else{
        b->down();
        b->l = merge(a, b->l);
        b->up();
        return b;
    }
}

void spilt(Node *o, Node *&a, Node *&b, int k){
    if(!o) a = b = NULL;
    else{
        o->down();
        if(Size(o->l)>=k){
            b = o;
            spilt(o->l, a, b->l, k);
        }
        else{
            a = o;
            spilt(o->r, a->r, b, k-Size(o->l)-1);
        }
        o->up();
    }
}

void Insert(Node *&o, int k,int v){
    if(!o) o = new Node(v);
    else{
        Node* tmp = new Node(v);
        Node *a, *b;
        spilt(o, a, b, k);
        o = merge(merge(a,tmp), b);
    }
}

void Del(Node *&o, int k){
    if(!o) return;
}

```

```

else{
    Node *a, *b, *c;
    spilt(o, a, b, k);
    spilt(a, a, c, k-1);
    o = merge(a, b);
}
}

int Min(Node *&o, int x, int y){
    if(!o) return 0;
    else{
        Node *a, *b, *c;
        spilt(o, a, b, y);
        spilt(a, a, c, x-1);
        if(c==0) return 0;
        c->up();
        int ans = c->mi;
        o = merge(merge(a,c), b);
        return ans;
    }
}

void Add(Node *&o, int x, int y, int v){
    if(!o) return;
    Node *a, *b, *c;
    spilt(o, a, b, y);
    spilt(a, a, c, x-1);
    if(c) c->tag += v;
    o = merge(merge(a,c), b);
}

```

```

void Reverse(Node *&o, int x, int y){
    if(!o) return;
    Node *a, *b, *c;
    spilt(o, a, b, y); // a b c
    spilt(a, a, c, x-1);
    if(c) c->rev ^= 1;
    o = merge(merge(a,c), b);
}

```

```

void Rotate(Node *&o, int x, int y, int t){
    if(!o) return;
    Node *a, *b, *c;
    spilt(o, a, b, y);
    spilt(a, a, c, x-1);

    Node *d, *e;
    t %= (y-x+1);
    if(t<0) t = y-x+1+t;
    spilt(c, d, e, Size(c)-t);
    c = merge(e, d);
    o = merge(merge(a,c), b);
}

```

```

/*
ADD x y D: Add D to each number in sub-sequence
{Ax ... Ay}. For example, performing "ADD 2 4 1" on
{1, 2, 3, 4, 5} results in {1, 3, 4, 5, 5}
REVERSE x y: reverse the sub-sequence {Ax ... Ay}. For
example, performing "REVERSE 2 4" on {1, 2, 3, 4, 5}
results in {1, 4, 3, 2, 5}
REVOLVE x y T: rotate sub-sequence {Ax ... Ay} T
times. For example, performing "REVOLVE 2 4 2" on {1,
2, 3, 4, 5} results in {1, 3, 4, 2, 5}
INSERT x P: insert P after Ax. For example, performing
"INSERT 2 4" on {1, 2, 3, 4, 5} results in {1, 2, 4,
3, 4, 5}
DELETE x: delete Ax. For example, performing "DELETE
2" on {1, 2, 3, 4, 5} results in {1, 3, 4, 5}
MIN x y: query the participant what is the minimum
number in sub-sequence {Ax ... Ay}. For example, the
correct answer to "MIN 2 4" on {1, 2, 3, 4, 5} is 2
*/
int main()
{

```

```

int n;
while(scanf("%d",&n)==1){
    root = NULL;
    for(int i=0; i<n; i++){
        scanf("%d",&a);
        root = merge(root, new Node(a));
    }
    int m, x, y, c;
    char s[20];
    scanf("%d", &m);
    for(int i=0; i<m; i++){
        scanf("%s", s);
        if(s[0]=='A'){
            scanf("%d%d",&x,&y,&c);
            Add(root, x, y, c);
        }
        else if(s[0]=='R' && s[3]=='E'){
            scanf("%d",&x,&y);
            Reverse(root, x, y);
        }
        else if(s[0]=='R'){
            scanf("%d%d",&x,&y,&c);
            Rotate(root, x, y, c);
        }
        else if(s[0]=='I'){
            scanf("%d",&x,&y);
            n++;
            Insert(root, x, y);
        }
        else if(s[0]=='D'){
            scanf("%d",&x);
            Del(root, x);
            n--;
        }
        else{
            scanf("%d",&x,&y);
            printf("%d\n", Min(root, x, y));
        }
    }
}
return 0;
}

```

● LCA

```

// adj[u] : adjacency list of u
// par[u][i] : (2^i)-th parent pf u
int LOG = 20;
int time = 0;
void dfs(int u, int p) {
    par[u][0] = p;
    timer_in[u] = ++timer;
    for (int v : adj[u]) if (v!=p) dfs(v, u);
    time_out[u] = ++timer;
}
bool anc(int x, int y) {
    return time_in[x] <= time_in[y]
        && time_out[y] <= time_out[x];
}
int lca(int x, int y) {
    if (anc(y, x)) return y;
    for (int j = LOG; j >= 0; j--) {
        if (!anc(par[y][j], x)) y = par[y][j];
    }
    return par[y][0];
}
int main() {
    int root = 1; // set root node
    dfs(root, root);

    for (int j = 1; j <= LOG; j++)
        for (int i = 1; i <= n; i++)
            pair[i][j] = par[par[i][j-1]][j-1];
    return 0;
}

```

● 樹鍊剖分

```
vector<int> G[maxn];
int pa[maxn], maxson[maxn], son[maxn];
int dep[maxn];
int link[maxn], linkpa[maxn];
int linkcnt = 0;
void dfs(int x, int p){
    pa[x] = p;
    dep[x] = dep[p]+1;
    son[x] = 1, maxson[x] = -1;
    for(int i=0; i<G[x].size(); i++){
        if(G[x][i]!=p){
            dfs(G[x][i], x);
            son[x] += son[G[x][i]];
            if(maxson[x]==-1 ||
            son[G[x][i]]>son[maxson[x]]) maxson[x] = G[x][i];
        }
    }
}
void build_link(int x, int plink){
    link[x] = ++linkcnt;
    linkpa[x] = plink;

    if(maxson[x]!=-1) build_link(maxson[x], plink);
    for(int i=0; i<G[x].size(); i++){
        int u = G[x][i];
        if(u==maxson[x] || u==pa[x]) continue;
        build_link(u, u);
    }
}
ll cal(int a, int b, int type){
    ll ans = 0;
    int ta = linkpa[a], tb = linkpa[b];
    while(linkpa[a]!=linkpa[b]){
        int A, B;

        if(dep[ ta ] <= dep[ tb ]){
            swap(a, b);
            swap(ta, tb);
        }
        A = link[ta];
        B = link[a];

        // if(type==1) T.add(1, n, 1, A, B);
        // else ans += T.query(1, n, 1, A, B);
        a = pa[linkpa[a]];
        ta = linkpa[a];
    }

    if(a==b) return ans;
    if(dep[a] > dep[b]) swap(a, b);

    int A = link[a] + 1, B = link[b];
    // if(type==1) T.add(1, n, 1, link[a]+1,
    link[b]);
    // else ans += T.query(1, n, 1, link[a]+1,
    link[b]);
    if(type==0) return ans;
}

```

● SCC.cpp

```
struct Kosaraju {
    // Vertex i belong to which SCC, call findSCC to
    build.
    int SCCof[MAXV+5], V, cnt;
    bool vis[MAXV+5];
    vector<int> *G, *Grev;
    stack<int> stk;
    void dfs(vector<int> *Gcur, int v) {
        for (auto u : Gcur[v]) {
            if (!vis[u]) {

```

```
                vis[u]=true;
                dfs(Gcur,u);
            }
        }
        if (Gcur==G) stk.push(v);
        else SCCof[v]=cnt;
    }
    int findSCC(int _V, vector<int> *_G, vector<int>
    *_Grev) {
        // G: Adjacency List of graph. Grev: Reverse
        graph of G.
        // No need for init, return # of SCC, 1-based
        V=_V; G=_G; Grev=_Grev;
        for (int i=1;i<=V;i++) vis[i]=0;
        for (int i=1;i<=V;i++) {
            if (!vis[i]) {
                vis[i]=true;
                dfs(G,i);
            }
        }
        cnt=0;
        for (int i=1;i<=V;i++) vis[i]=0;
        while (!stk.empty()) {
            int v=stk.top();
            stk.pop();
            if (!vis[v]) {
                cnt++;
                vis[v]=true;
                dfs(Grev,v);
            }
        }
        return cnt;
    }
    void compress(vector<int> *Gtar) {
        // Pack SCC into one vertex, store into Gtar
        // Call findSCC before this, 1-based
        for (int i=1;i<=V;i++)
            for (auto j : G[i])
                if (SCCof[i]!=SCCof[j])
                    Gtar[SCCof[i]].push_back(SCCof[j]);
    }
};

```

● Dinic flow

```
template<class T>
struct Dinic{
    struct Edge{
        int f,to;
        T c;
        Edge(int _f,int _to,T _c):f(_f),to(_to),c(_c){}
    };

    // IMPORANT
    // maxn is the number of vertices in the graph
    // Not the N in the problem statement!!
    vector<int> G[maxn];
    vector<Edge> es;
    int level[maxn],st, end, n;
    int cur[maxn];

    void init(int _n){
        n = _n;
        es.clear();
        for(int i=0; i<=n; i++) G[i].clear();
    }

    void addEdge(int f,int t,T c, bool
    directed=false){
        es.push_back(Edge(f,t,c));
        G[f].push_back(es.size()-1);
        es.push_back(Edge(t,f,directed?0:c));
    }
};

```

```

G[t].push_back(es.size()-1);
}

bool BFS(int s,int t){
    queue<int> Q;
    for(int i=0; i<=n; i++) level[i] = 0;
    level[s] = 1;
    Q.push(s);
    while(!Q.empty()){
        int x = Q.front(); Q.pop();
        for(int i=0; i<G[x].size(); i++){
            Edge e = es[ G[x][i] ];
            if(e.c==0 || level[e.to]) continue;
            level[e.to] = level[x] + 1;
            Q.push(e.to);
        }
    }
    return level[t]!=0;
}

T DFS(int s,int cur_flow){ // can't exceed c
    if(s==end) return cur_flow;

    T ans = 0, temp, total = 0;

    for(int& i=cur[s]; i<G[s].size(); i++){
        Edge &e = es[ G[s][i] ];
        if(e.c==0 || level[e.to]!=level[s]+1)
            continue;
        temp = DFS(e.to, min(e.c, cur_flow));
        if(temp!=0){
            e.c -= temp;
            es[G[s][i]^1].c += temp;
            cur_flow -= temp;
            total += temp;
            if(cur_flow==0) break;
        }
    }
    return total;
}

T maxFlow(int s,int t){
    T ans = 0;
    st = s, end = t;
    while(BFS(s,t)){
        while(true){
            memset(cur, 0, sizeof(cur));
            T temp = DFS(s,INF);
            if(temp==0) break;
            ans += temp;
        }
    }
    return ans;
};

```

● 最小費用流

```

template<class T>
struct Min_cost_flow {
    // 0-based
    struct Edge {
        int fr,to;
        T cap,cost;
    };
    // IMPORANT
    // MAXV is the number of vertices in the graph
    // Not the N in the problem statement!!
    int V,E;
    vector<Edge> es;
    vector<int> G[MAXV+5];
};

```

```

void init(int _V) {
    V=_V;
    E=0;
    for (int i=0;i<=V;i++) G[i].clear();
    es.clear();
}

int add_edge(int fr, int to, T cap, T cost) {
    es.pb({fr,to,cap,cost});
    es.pb({to,fr,0,-cost});
    G[fr].push_back(E);
    G[to].push_back(E^1);
    E+=2;
    return E-2;
}

bool SPFA(int s, int t, T &ans_flow, T &ans_cost)
{
    queue<int> que;
    int pre[MAXV+5];
    T dist[MAXV+5],flow[MAXV+5];
    bool inque[MAXV+5];
    for (int i=0;i<=V;i++) {
        dist[i]=INF;
        inque[i]=false;
    }
    dist[s]=0;
    flow[s]=INF;
    inque[s]=true;
    que.push(s);
    while (!que.empty()) {
        int v=que.front(); que.pop();
        inque[v]=false;
        for (int idx : G[v]) {
            Edge &e=es[idx];
            if (e.cap>0 &&
                dist[e.fr]+e.cost<dist[e.to]) {
                flow[e.to]=min(flow[e.fr],e.cap);
                dist[e.to]=dist[e.fr]+e.cost;
                pre[e.to]=idx;
                if (!inque[e.to])
                    que.push(e.to),inque[e.to]=true;
            }
        }
    }
    if (dist[t]==INF) return false;
    //if (dist[t]>=0) return false;
    // Add above line -> min cost > max flow
    (priority)
    // Without -> max flow > min cost

    int v=t;
    ans_flow+=flow[t];
    ans_cost+=(dist[t]*flow[t]);
    while (v!=s) {
        es[pre[v]].cap-=flow[t];
        es[pre[v]^1].cap+=flow[t];
        v=es[pre[v]].fr;
    }
    return true;
}

pair<T,T> min_cost_flow(int s, int t) {
    T ans_flow=0, ans_cost=0;
    while (SPFA(s,t,ans_flow,ans_cost));
    return make_pair(ans_flow,ans_cost);
};

```

● Blossom matching

```

struct Blossom {
    #define MAXN 505 // Max solvable problem, DON'T
    CHANGE
    // 1-based, IMPORTANT
    vector<int> g[MAXN];
};

```

```

int parent[MAXN], match[MAXN], belong[MAXN],
state[MAXN];
int n;
int lca(int u, int v) {
    static int cases = 0, used[MAXN] = {};
    for (++cases; ; swap(u, v)) {
        if (u == 0)
            continue;
        if (used[u] == cases)
            return u;
        used[u] = cases;
        u = belong[parent[match[u]]];
    }
}
void flower(int u, int v, int l, queue<int> &q) {
    while (belong[u] != l) {
        parent[u] = v, v = match[u];
        if (state[v] == 1)
            q.push(v), state[v] = 0;
        belong[u] = belong[v] = l, u =
parent[v];
    }
}
bool bfs(int u) {
    for (int i = 0; i <= n; i++)
        belong[i] = i;
    memset(state, -1, sizeof(state[0])*(n+1));
    queue<int> q;
    q.push(u), state[u] = 0;
    while (!q.empty()) {
        u = q.front(), q.pop();
        for (int i = 0; i < g[u].size(); i++) {
            int v = g[u][i];
            if (state[v] == -1) {
                parent[v] = u, state[v] = 1;
                if (match[v] == 0) {
                    for (int prev; u; v =
prev, u = parent[v]) {
                        prev = match[u];
                        match[u] = v;
                        match[v] = u;
                    }
                    return 1;
                }
                q.push(match[v]),
state[match[v]] = 0;
            } else if (state[v] == 0 &&
belong[v] != belong[u]) {
                int l = lca(u, v);
                flower(v, u, l, q);
                flower(u, v, l, q);
            }
        }
    }
    return 0;
}
int blossom() {
    memset(parent, 0, sizeof(parent[0])*(n+1));
    memset(match, 0, sizeof(match[0])*(n+1));
    int ret = 0;
    for (int i = 1; i <= n; i++) {
        if (match[i] == 0 && bfs(i))
            ret++;
    }
    return ret;
}
void addEdge(int x, int y) {
    g[x].push_back(y), g[y].push_back(x);
}
void init(int _n) {
    n = _n;
    for (int i = 0; i <= n; i++)
        g[i].clear();
}
} algo;

```

● 穩定婚姻

```

const int maxn = 1100;
int manWant[maxn][maxn], nextW[maxn];
int women[maxn][maxn], order[maxn][maxn];
int wife[maxn], husband[maxn];
queue<int> singleDog;

void engage(int m, int w){
    if(husband[w]!=0){
        wife[ husband[w] ] = 0;
        singleDog.push( husband[w] );
        husband[w] = 0;
    }
    husband[w] = m;
    wife[m] = w;
    // cout << m << " --> " << w << endl;
}

int main()
{
    int Time, n, cas = 0;
    scanf("%d",&Time);

    while(Time-- && scanf("%d",&n)==1){
        for(int i=1; i<=n; i++){
            for(int j=1; j<=n; j++)
                scanf("%d",&manWant[i][j]);
            nextW[i] = 1;
            wife[i] = 0;
            singleDog.push(i);
        }

        for(int i=1; i<=n; i++){
            for(int j=1; j<=n; j++){
                scanf("%d",&women[i][j]);
                order[i][ women[i][j] ] = j;
            }
            husband[i] = 0;
        }

        while(!singleDog.empty()){
            int x = singleDog.front(); singleDog.pop();
            // cout << x << endl;
            int to = manWant[x][nextW[x]++];

            if(husband[to]==0) engage(x, to);
            else if(order[to][husband[to]] >
order[to][x]) engage(x, to);
            else singleDog.push(x);
        }
        if(cas++) printf("\n");
        for(int i=1; i<=n; i++) printf("%d\n",
wife[i]);
    }
    return 0;
}

```

● KM

```

template<class T>
struct KM_n_3
{
    T G[maxn][maxn];
    T lx[maxn], ly[maxn], y_slack[maxn];
    int x_match[maxn], y_match[maxn];
    int px[maxn], py[maxn];
    int n;

    void toggle(int y){
        x_match[py[y]] = y;
    }
}

```

```

    y_match[y] = py[y];
    if(px[y_match[y]]!=-2) toggle(px[y_match[y]]);
}

/* n = |L| = |R|, id of nodes start with 1*/
int init(int _n){
    n = _n;
    for(int i=0; i<=n; i++){
        for(int j=0; j<=n; j++){
            G[i][j] = 0;
        }
    }
}

int add_edge(int a, int b, T c){ G[a][b] = c; }

int dfs(int x){
    for(int y=1; y<=n; y++){
        if(py[y]) continue;
        T slack = lx[x] + ly[y] - G[x][y];
        if(slack==0){
            py[y] = x;
            if(y_match[y]==0){
                toggle(y);
                return true;
            }
            else{
                if(px[y_match[y]]) continue;
                px[ y_match[y] ] = y;
                if(dfs(y_match[y])) return 1;
            }
        }
        else if(y_slack[y] > slack){
            y_slack[y] = slack;
            y_par[y] = x;
        }
    }
    return false;
}

void update(vector<int>& Y){
    Y.clear();
    T d = INF;
    for(int i=1; i<=n; i++) if(!py[i]) d = min(d,
y_slack[i]);

    for(int i=1; i<=n; i++){
        if(px[i]) lx[i] -= d;
        if(py[i]) ly[i] += d;
        else{
            y_slack[i] -= d;
            if(y_slack[i]==0) Y.push_back(i);
        }
    }
}

T km(){
    for(int i=0; i<=n; i++) x_match[i] = y_match[i]
= 0;
    for(int i=1; i<=n; i++){
        lx[i] = G[i][1], ly[i] = 0;
        for(int j=1; j<=n; j++){
            lx[i] = max(lx[i], G[i][j]);
        }
    }
    for(int i=1; i<=n; i++){
        for(int j=0; j<=n; j++) y_slack[j] = INF;
        for(int j=0; j<=n; j++) px[j] = py[j] = 0;
        px[i] = -2;
        if(dfs(i)) continue;

        // adjust labeling until finding an
augmenting path
        bool find = false;
        while(!find){
            vector<int> Y;

```

```

        update(Y);
        for(auto y:Y){
            if(find) break;
            py[y] = y_par[y];
            if(y_match[y]==0){
                toggle(y);
                find = true;
            }
            else{
                px[ y_match[y] ] = y;
                if(dfs(y_match[y])) find = true;
            }
        }
    }

    T ans = 0;
    for(int x=1; x<=n; x++) ans +=
G[x][x_match[x]];
    return ans;
}

void dump(vector<pair<int,int>>& ans){
    for(int i=1; i<=n; i++)
if(G[i][x_match[i]]!=0){
        ans.push_back({i, x_match[i]});
    }
}
}

```

● EXT_GCD

```

typedef pair < LL, LL> ii;
ii exd_gcd( LL a, LL b) {
    if (a % b == 0) return ii(0, 1);
    ii T = exd_gcd(b, a % b);
    return ii( T.second, T.first - a / b * T.second);
}

LL mod_inv(LL x) {
    // P is mod number, gcd(x,P) must be 1
    return (exd_gcd(x,P).first%P+P)%P;
}

```

● LUCAS

```

struct Lucas {
    // P is mod number, must be prime
    LL fac[MAXP+5], P;
    void init(LL _P) {
        P = _P;
        LL i;
        fac[0] = 1;
        for(i = 1; i <= MAXP; i++) {
            fac[i] = fac[i-1]*i % P;
        }
    }
    LL qpow(LL a, LL p) {
        if (p==0) return 1;
        LL tmp=qpow(a,p/2);
        if (p&1) return tmp*tmp%P*a%P;
        return tmp*tmp%P;
    }
    LL C(LL n, LL k) {
        if(k > n) return 0;
        return fac[n]*qpow(fac[k]*fac[n-k]%P, P-2) %
P;
    }
    LL lucas(LL n, LL k) {
        if(k ==0) return 1;
        else return (C(n%P, k%P)*lucas(n/P, k/P))%P;
    }
};

```


● Miller Rabin

```
LL mod_mul(LL a, LL b, LL mod) {
    // return (__int128)a*b%mod;
    /* In case __int128 doesn't work(32* multi to
    avoid ovf) */
    LL x=0,y=a%mod;
    while(b > 0){
        if (b&1) x = (x+y)%mod;
        y = (y*2)%mod;
        b >>= 1;
    }
    return x%mod;
}

LL qpow(LL a, LL p, LL mod) {
    if (p<=0) return 1;
    LL temp = qpow(a,p/2,mod);
    temp = mod_mul(temp,temp,mod);
    if (p&1) return mod_mul(temp,a,mod);
    return temp;
}

bool MRtest(LL a, LL d, LL n) {
    LL x = qpow(a,d,n);
    if (x==1 || x==n-1) return true;
    while (d != n-1) {
        x = mod_mul(x,x,n);
        d *= 2;
        if (x==n-1) return true;
        if (x==1) return false;
    }
    return false;
}

bool is_prime(LL n) {
    if (n==2) return true;
    if (n<2 || n%2==0) return false;
    LL table[7] = {2, 325, 9375, 28178, 450775,
    9780504, 1795265022}, d=n-1;
    while (d%2 != 0) d>>=1; // n-1 = d * 2^r, d is
    odd.
    for (int i=0; i<7; i++) {
        LL a = table[i] % n;
        if (a==0 || a==1 || a==n-1) continue;
        if (!MRtest(a,d,n)) {
            return false;
        }
    }
    return true;
}
```

● Computational Geometry

```
const double PI=acos(-1);
struct Point {
    // Also a vector
    double x,y;
    Point operator+(const Point &p) const {
        return {x+p.x,y+p.y};
    }
    Point operator-(const Point &p) const {
        return {x-p.x,y-p.y};
    }
    Point operator*(double mul) const {
        return {x*mul,y*mul};
    }
    Point operator/(double mul) const {
        return {x/mul,y/mul};
    }
    bool operator==(const Point &p) const {
        return x==p.x&&y==p.y;
    }
    double cross(const Point &v) const {
        return x*v.y-y*v.x;
    }
    double dot(const Point &v) const {
```

```
        return x*v.x+y*v.y;
    }
    Point normal() { // Normal vector
        return {-y,x};
    }
    double len() const { // Length
        return sqrt(x*x+y*y);
    }
    double angle(const Point &v) const {
        // Angle from *this to v in [-pi,pi].
        return atan2(cross(v),dot(v));
    }
    Point rotate_about(double theta, const Point &p)
    const {
        // Rotate this point counterclockwise by
        theta about p
        double nx=x-p.x,ny=y-p.y;
        return {nx*cos(theta)-
        ny*sin(theta)+p.x,nx*sin(theta)+ny*cos(theta)+p.y};
    }
};

struct Line {
    // Also a segment
    Point p1,p2;
    double a,b,c; // ax+by+c=0
    Line(){}
    Line(const Point &p1, const Point &p2) {
        p1=_p1; p2=_p2;
        pton();
    }
    void pton() {
        // IMPORTANT if you don't use constructor.
        a=p1.y-p2.y;
        b=p2.x-p1.x;
        c=-a*p1.x-b*p1.y;
    }
    int relation(const Point &p) {
        // For line, 0 if point on line
        // -1 if left, 1 if right
        Point dir=p2-p1;
        double crs=dir.cross(p-p1);
        return crs==0?0:crs<0?-1:1;
    }
    Point normal() {
        Point dir=p2-p1;
        return {-dir.y,dir.x};
    }
    bool on_segment(const Point &p) {
        // Point on segment
        return relation(p)==0&&(p2-p).dot(p1-p)<=0;
    }
    bool parallel(const Line &l) {
        // Two line parallel
        return (p2-p1).cross(l.p2-l.p1)==0;
    }
    bool equal(const Line &l) {
        // Two line equal
        return relation(l.p1)==0&&relation(l.p2)==0;
    }
    bool cross_seg(const Line &seg) {
        // Line intersect segment
        Point dir=p2-p1;
        return dir.cross(seg.p1-
        p1)*dir.cross(seg.p2-p1)<=0;
    }
    int seg_intersect(const Line &s) const{
        // Two segment intersect
        // 0 -> no, 1 -> one point, -1 -> infinity
        Point dir=p2-p1, dir2=s.p2-s.p1;
        double c1=dir.cross(s.p2-p1);
        double c2=dir.cross(s.p1-p1);
        double c3=dir2.cross(p2-s.p1);
        double c4=dir2.cross(p1-s.p1);
        if (c1==0&&c2==0) {
            if ((s.p2-p1).dot(s.p1-p1)>0&&(s.p2-
            p2).dot(s.p1-p2)>0&&
            (p1-s.p1).dot(p2-s.p1)>0&&(p1-
            s.p2).dot(p2-s.p2)>0) return 0;
            if (p1==s.p1&&(p2-p1).dot(s.p2-
            p1)<=0) return 1;
            if (p1==s.p2&&(p2-p1).dot(s.p1-
            p1)<=0) return 1;
            if (p2==s.p1&&(p1-p2).dot(s.p2-
```



```

p2)<=0)return 1;
if(p2==s.p2&&(p1-p2).dot(s.p1-
p2)<=0)return 1;
return -1;
}else if(c1*c2<=0&&c3*c4<=0)return 1;
return 0;
}
Point line_intersection(const Line &l) const{
// Intersection of lines
// pton(); L.pton();
double deno=a*1.b-l.a*b;
if (deno!=0) {
return { (1.c*b-c*1.b)/deno, (1.a*c-
a*1.c)/deno};
}
// Reaches here means no intersection.
(parallel)
return {1234,4321};
}
Point seg_intersection(Line &s) const {
Point dir=p2-p1, dir2=s.p2-s.p1;
// pton(); L.pton();
double c1=dir.cross(s.p2-p1);
double c2=dir.cross(s.p1-p1);
double c3=dir2.cross(p2-s.p1);
double c4=dir2.cross(p1-s.p1);
if (c1==0&&c2==0) {
if(p1==s.p1&&(p2-p1).dot(s.p2-
p1)<=0)return p1;
if(p1==s.p2&&(p2-p1).dot(s.p1-
p1)<=0)return p1;
if(p2==s.p1&&(p1-p2).dot(s.p2-
p2)<=0)return p2;
if(p2==s.p2&&(p1-p2).dot(s.p1-
p2)<=0)return p2;
}else if(c1*c2<=0&&c3*c4<=0)return
line_intersection(s);
// Reaches here means either INF or NOT ANY
// Use seg_intersect to check Ouo
return {1234,4321};
}
double dist(const Point &p, bool is_segment)
const {
// Point to Line/segment
Point dir=p2-p1,v=p-p1;
if (is_segment) {
if (dir.dot(v)<0) return v.len();
if ((p1-p2).dot(p-p2)<0) return (p-
p2).len();
}
double d=abs(dir.cross(v))/dir.len();
return d;
}
};

struct Polygon {
vector<Point> V; // Counterclockwise
double area() const {
double res=0;
for (int i=1;i+1<V.size();i++) {
res+=(V[i]-V[0]).cross(V[i+1]-V[0]);
}
return abs(res/2.0);
}
bool contain(const Point &p) {
// Point can't on side
int i, j, k = 0;
for(i = 0, j = V.size()-1; i < V.size(); j =
i++) {
if(V[i].y > p.y != V[j].y > p.y &&
p.x < (V[j].x-V[i].x)*(p.y-
V[i].y)/(V[j].y-V[i].y)+V[i].x)
k++;
}
return k&1;
}
};

```

● Convex Hull

```

void convex_hull(vector<Point> &ps, vector<Point>
&hull) {
// Find convex hull of ps, store in hull

```

```

vector<Point> &stk=hull;
stk.resize(ps.size()+1);
sort(ps.begin(),ps.end()); // Using x to cmp, y
secondary.
int t=-1; // top
for (int i=0;i<ps.size();i++) {
// cross<-EPS -> count collinear, cross<EPS
-> not
while (t>=1&&(stk[t]-stk[t-1]).cross(ps[i]-
stk[t])<EPS) t--;
stk[++t]=ps[i];
}
int low=t;
for (int i=ps.size()-2;i>=0;i--) {
// cross<-EPS -> count collinear, cross<EPS
-> not
while (t>low&&(stk[t]-stk[t-1]).cross(ps[i]-
stk[t])<EPS) t--;
stk[++t]=ps[i];
}
stk.resize(t);
}

```

● EPS

```

const double EPS=1e-9;
struct Double{
double d;
Double(double d=0):d(d){}
bool operator <(const Double &b)const{return d-
b.d<-EPS;}
bool operator >(const Double &b)const{return d-
b.d>EPS;}
bool operator ==(const Double &b)const{return
abs(d-b.d)<=EPS;}
bool operator !=(const Double &b)const{return
abs(d-b.d)>EPS;}
bool operator <=(const Double &b)const{return d-
b.d<=EPS;}
bool operator >=(const Double &b)const{return d-
b.d>=-EPS;}
operator double()const{return d;}
};

```

● Smallest Circle

```

struct Circle{
Point x;
double r;
bool incircle(const Point &c)const{return (x-
c).len()<=r+EPS;}
bool stincircle(const Point &c)const{return (x-
c).len()<=r-EPS;}
};

Circle TwoPointCircle(const Point &a, const Point &b)
{
Point m=(a+b)/2;
return (Circle){m,(a-m).len()};
}

Circle outcircle(Point a, Point b, Point c) {
if(TwoPointCircle(a,b).incircle(c)) return
TwoPointCircle(a,b);
if(TwoPointCircle(b,c).incircle(a)) return
TwoPointCircle(b,c);
if(TwoPointCircle(c,a).incircle(b)) return
TwoPointCircle(c,a);
Point ret;
double a1=b.x-a.x, b1=b.y-a.y, c1=(a1*a1+b1*b1)/2;
double a2=c.x-a.x, b2=c.y-a.y, c2=(a2*a2+b2*b2)/2;
double d = a1*b2 - a2*b1;
ret.x=a.x+(c1*b2-c2*b1)/d;
ret.y=a.y+(a1*c2-a2*c1)/d;
return (Circle){ret,(ret-a).len()};
}

```

```

}
//rand required
Circle SmallestCircle(vector<Point> &p){
    int n=p.size();
    if(n==0) return {{INF,INF},0};
    if(n==1) return (Circle){p[0],0.0};
    if(n==2) return TwoPointCircle(p[0],p[1]);
    random_shuffle(p.begin(),p.end());
    Circle c = {p[0],0.0};
    for(int i=0;i<n;++i){
        if(c.incircle(p[i])) continue;
        c=Circle{p[i],0.0};
        for(int j=0;j<i;++j){
            if(c.incircle(p[j])) continue;
            c=TwoPointCircle(p[i],p[j]);
            for(int k=0;k<j;++k){
                if(c.incircle(p[k])) continue;
                c=outcircle(p[i],p[j],p[k]);
            }
        }
    }
    return c;
}

```

● Gaussian elimination

```

typedef double Matrix[maxn][maxn];
void gauss_elimination(Matrix A, int n){
    int r;
    for(int i=0; i<n; i++){
        r = i;
        for(int j=i+1; j<n; j++){
            if(fabs(A[j][i]) > fabs(A[r][i])) r = j;
            if(r!=i) for(int j=0; j<n; j++) swap(A[r][j],
A[i][j]);

            for(int k=i+1; k<n; k++){
                double f = A[k][i]/A[i][i];
                for(int j=i; j<n; j++) A[k][j] -=
f*A[i][j];
            }
        }

        for(int i=n-1; i>=0; i--){
            for(int j=i+1; j<n; j++)
                A[i][n] -= A[j][n] * A[i][j];
            A[i][n] /= A[i][i];
        }
    }
}

```

● LL multiplication

```

long long mul(long long a, long long b) {
    long long ans = 0, step = a % MOD;
    while (b) {
        if (b & 1L) ans += step;
        if (ans >= MOD) ans %= MOD;
        step <<= 1L;
        if (step >= MOD) step %= MOD;
        b >>= 1L;
    }
    return ans % MOD;
}

```

● Formulas

滿足 $\text{ceil}(n/i)=k$ 之最大 i :

INF, if $k=1$
 $n/(k-1)-1$, else if $k-1$ 整除 n
 $x/(k-1)$, else

滿足 $\text{floor}(n/i)=k$ 之最大 i : $\text{floor}(n/k)$

尤拉函數: $\phi(n)=n$ 乘上所有 $(1-1/p)$, 對 n 之所有質因數 p

費馬小定理: $a * a^{(p-2)} = 1 \pmod{p}$, a, p 互質

次方同餘定理: $a^k \pmod{p} = (a \pmod{p})^{(k \pmod{p-1})} \pmod{p}$
 是質數

枚舉擴展歐幾里得之解:

若 x_0, y_0 為 $a*x+b*y = k$ 之一組解, 則

$x=x_0+t*b/\text{gcd}(a,b)$, $y=y_0+t*a/\text{gcd}(a,b)$ 亦為解, t

為整數

最大獨立集: 點的集合, 其內點不相鄰

最小點覆蓋: 點的集合, 所有邊都被覆蓋

最大匹配: 邊的集合, 其內邊不共用點

最小邊覆蓋: 邊的集合, 所有點都被覆蓋

最大獨立集+最小點覆蓋= V (數值)

最大匹配+最小邊覆蓋= V (數值)

最大匹配=最大流(二分圖)

最大匹配=最小點覆蓋(二分圖)

最小點覆蓋+最小邊覆蓋= V (數值, 二分圖)

一矩陣 A 所有eigen value之合=對角線合

一矩陣 A 所有eigen value之積= $\det(A)$

三角形 ABC , 對邊長 abc :

$\text{area}=\sqrt{s(s-a)(s-b)(s-c)}$, $s=\text{周長}/2$

$a/\sin A = b/\sin B = c/\sin C = 2R$, R 為外接圓半徑

內接圓半徑= $2*\text{area}/(a+b+c)$

外接圓半徑= $abc/4*\text{area}$

某些質數:

54018521, 370248451, 6643838879, 119218851371,

5600748293801

39916801, 479001599, 87178291199, 8589935681,

433494437, 2971215073