**------------QUEUE------------**

#define MAX\_SIZE 500

typedef int ElementType;

typedef struct Queue {

ElementType data[MAX\_SIZE];

int front, rear;

} Queue;

void make\_null\_queue(Queue \*pQ) {

pQ->front = 0;

pQ->rear = -1;

}

void enqueue(Queue \*pQ, ElementType u) {

pQ->rear++;

pQ->data[pQ->rear] = u;

}

ElementType front(Queue \*pQ) {

return pQ->data[pQ->front];

}

void dequeue(Queue \*pQ) {

pQ->front++;

}

int empty(Queue \*pQ) {

return pQ->front > pQ->rear;

}

**------------LIST-------------**

#define MAX\_N 100

typedef struct List{

ElementType data[100];

int size;

}List;

void make\_null(List \*pL){

pL->size = 0;

}

void push\_back(List \*pL, ElementType x){

pL->data[pL->size] = x;

pL->size++;

}

void copy\_list(List \*pL1, List \*pL2){

make\_null(pL1);

for(int i = 0; i < pL2->size; ++i){

push\_back(pL1, pL2->data[i]);

}

}

int emptyList(List \*pL){

return pL->size == 0;

}

**-------------STACK-----------**

#define MAX\_SIZE 100

typedef int ElementType;

typedef struct {

ElementType data[MAX\_SIZE];

int top\_idx;

} Stack;

void make\_null\_stack(Stack \*pS) {

pS->top\_idx = -1;

}

void push(Stack \*pS, ElementType u) {

pS->top\_idx++;

pS->data[pS->top\_idx] = u;

}

ElementType top(Stack \*pS) {

return pS->data[pS->top\_idx];

}

void pop(Stack \*pS) {

pS->top\_idx--;

}

int empty(Stack \*pS) {

return pS->top\_idx == -1;

}

**----------BFS Tree-----------**

int mark[MAXN];

int parent[MAXN];

void BFS(Graph \*pG, int s){

Queue Q;

makenull(&Q);

enqueue(&Q, s);

while(!empty(&Q)){

int u = front(&Q);

dequeue(&Q);

if(mark[u]){

continue;

}

mark[u] = 1;

for(int v = 1; v<=pG->n;++v){

if(adjacent(pG, u, v)

&& !mark[v]){

if(parent[v] == 1){

parent[v]=u;

enqueue(&Q, v);

}

}

}

}

}

**----------DFS Tree-----------**

int mark[MAX\_VERTICES];

int parent[MAX\_VERTICES];

void dfs(Graph \*pG, int a){

Stack S;

make\_null\_stack(&S);

push(&S, a);

while(!empty(&S)){

int u = top(&S);

pop(&S);

if(mark[u]==1){

continue;

}

mark[u]=1;

for(int i=1;i<=pG->n;++i){

if(adjacent(pG, u, i)

&& !mark[i]){

parent[i] = u;

push(&S, i);

}

}

}

}

**----------DFS Tree-----------**

int mark[MAX\_VERTICES];

int parent[MAX\_VERTICES];

void dfs(Graph \*pG, int u){

if(mark[u]==1){

return;

}

mark[u]=1;

for(int i=1;i<=pG->n;++i){

if(adjacent(pG, u, i)

&& !mark[i]){

parent[i] = u;

dfs(pG, i);

}

}

}

**---------SỐ BPLT MAX---------**

int mark[MAX\_VERTICES];

int nb\_u;

void dfs(Graph \*pG, int u){

if(mark[u]){

return;

}

//printf("%d\n", u);

mark[u]=1;

nb\_u++;

for(int i=1;i<=pG->n;++i){

if(adjacent(pG, u, i)){

dfs(pG, i);

}

}

}

int main(){  
 ...

int max\_cnt = 0;

nb\_u = 0;

for(int i=1;i<=G.n;++i){

if(mark[i]==0){

dfs(&G, i);

if(max\_cnt < nb\_u){

max\_cnt = nb\_u;

}

nb\_u = 0;

}

}

...

}

**-----KIỂM TRA ĐT PHÂN ĐÔI----**

#define WHITE 0

#define BLUE 1

#define RED 2

int colors[MAXN];

int conflict;

void colorize(Graph \*pG, int u, int c){

colors[u] = c;

for(int v = 1; v <= pG->n;++v){

if(adjacent(pG, u, v)){

if(colors[v] == c){

conflict = 1;

return;

}else if(colors[v]==0){

colorize(pG,v,3-c);

}

}

}

}

**------KIỂM TRA CHU TRÌNH-----**

#define WHITE 0

#define GRAY 1

#define BLACK 2

int color[MAX\_VERTICES];

int has\_circle;

void dfs(Graph \*pG, int u){

if(color[u] == BLACK){

return;

}

color[u] = GRAY;

for(int i=1;i<=pG->n;++i){

if(adjacent(pG, u, i)){

if(color[i]==WHITE){

dfs(pG, i);

}else if(color==GRAY){

has\_circle = 1;

}

}

}

color[u] = BLACK;

}

**--------ĐẾM BPLT MẠNH--------**

int num[MAXN],min\_num[MAXN];

Stack S;

int onStack[MAXN];

int k,cnt;

#define min(a,b) (a<b)?a:b

void SCC(Graph \*pG, int s){

num[s] = min\_num[s] = k++;

push(&S, s);

onStack[s] = 1;

for(int v = 1; v <=pG->n;++v){

if(adjacent(pG, s, v)){

if(num[v] < 0){

SCC(pG, v);

min\_num[s] = min(min\_num[v], min\_num[s]);

}else if(onStack[v]){

min\_num[s] = min(min\_num[s], num[v]);

}

}

}

if(num[s] == min\_num[s]){

int w;

do{

w = top(&S);

pop(&S);

onStack[w] = 0;

}while(w!=s);

cnt++;

}

}

Int main(){  
 ...

for(int u = 1; u <= n;++u){

num[u] = -1;

min\_num[u] = 9999;

onStack[u] = 0;

}

k = 1;

makeNull(&S);

cnt = 0;

for(int u = 1; u <=n;++u){

if(num[u] == -1){

SCC(&G, u);

}

}

...

}

**--------FLOYD WARSHALL-------**

int pi[MAXN][MAXN];

int next[MAXN][MAXN];

void Floyd(Graph \*pG){

for(int u = 1; u <= pG->n;++u){

for(int v = 1; v <= pG->n;++v){

pi[u][v] = oo;

next[u][v] = -1;

}

}

for(int u = 1; u <= pG->n;++u){

pi[u][u] = 0;

next[u][u] = u;

}

for(int u = 1; u <= pG->n;++u){

for(int v = 1; v <= pG->n;++v){

if(adjacent(pG, u, v)){

pi[u][v]=getWeight(pG,u,v);

next[u][v] = v;

}

}

}

for(int k = 1; k <= pG->n;++k){

for(int u = 1; u <= pG->n;++u){

for(int v = 1; v <= pG->n;++v){

if(pi[u][k] != oo && pi[k][v] != oo){

if(pi[u][k] + pi[k][v] < pi[u][v]){

pi[u][v] = pi[u][k] + pi[k][v];

next[u][v] = next[u][k];

}

}

}

}

}

}

**----SỐ ĐƯỜNG ĐI NGẮN NHẤT----**

int mark[MAX\_N];

int pi[MAX\_N], p[MAX\_N];

**int cnt[MAX\_N];**

void Dijkstra(Graph \*pG, int s){

for(int u = 1; u<=pG->n;++u){

mark[u] = 0;

pi[u] = INT\_MAX;

**cnt[u] = 0;**

}

pi[s] = 0;

p[s] = -1;

**cnt[s] = 1;**

for(int it = 1; it<=pG->n-1;++it){

int min\_pi = INT\_MAX;

int idx;

for(int u = 1; u<=pG->n;++u){

if(pi[u] < min\_pi && !mark[u]){

min\_pi = pi[u];

idx = u;

}

}

mark[idx] = 1;

for(int v = 1; v<=pG->n;++v){

if(adjacent(pG, idx, v)){

if(!mark[v] && pi[idx] + pG->A[idx][v] < pi[v]){

pi[v] = pi[idx] + pG->A[idx][v];

p[v]=idx;

**cnt[v]=cnt[idx];**

}else if(pi[idx]+pG->A[idx][v] == pi[v]){

**cnt[v] += cnt[idx];**

}

}

}

}

}

**---------------------------MÊ CUNG SỐ--------------------------**

int mark[MAXN];

int pi[MAXN];

int p[MAXN];

#define oo 999999

void SolvingMaze(Maze \*pM, int s){

int N = pM->n;

int M = pM->m;

int numV = pM->m\*pM->n;

for(int u = 1; u <= numV;++u){

mark[u] = 0;

pi[u] = oo;

p[u] = 0;

}

pi[s] = 0;

p[s] = -1;

int di[] = {-1, 1, 0, 0};

int dj[] = {0, 0, -1, 1};

for(int it = 1; it <= numV - 1; ++it){

int idx, min = oo;

for(int u = 1; u <= numV;++u){

if(!mark[u] && pi[u] < min){

min = pi[u];

idx = u;

}

}

mark[idx] = 1;

int idx\_i = (idx - 1)/M;

int idx\_j = (idx - 1)%M;

for(int k = 0; k < 4;++k){

int ii = idx\_i + di[k];

int jj = idx\_j + dj[k];

if(ii >= 0 && ii <= N-1 && jj >=0 && jj <= M-1){

int v = ii\*M + jj + 1;

if(!mark[v] && pi[idx] + pM->A[ii][jj] < pi[v]){

pi[v] = pi[idx] + pM->A[ii][jj];

p[v] = idx;

}

}

}

}

}

**--------QUẢN LÍ DỰ ÁN--------**

int d[MAXN];

void TopoSort(Graph \*pG, List \*pL){

List S1,S2;

makeNull(&S1);

int deg[MAXN];

for(int u = 1; u <= pG->n;++u){

deg[u] = 0;

for(int x = 1; x<=pG->n;++x){

if(adjacent(pG, x, u)){

deg[u]++;

}

}

if(deg[u] == 0){

pushBack(&S1, u);

}

}

while(S1.size > 0){

makeNull(&S2);

for(int i = 0; i < S1.size;++i){

int u = S1.data[i];

pushBack(pL, u);

for(int v = 1; v <= pG->n;++v){

if(adjacent(pG, u, v)){

deg[v]--;

if(deg[v] == 0){

pushBack(&S2, v);

}

}

}

}

copyList(&S1, &S2);

}

}

#define oo 999999

#define max(a,b) (a>b) ? a : b

#define min(a,b) (a<b) ? a : b

int main(){

Graph G;

//freopen("dt.txt", "r", stdin);

int n;

scanf("%d", &n);

init\_graph(&G, n+2);

int alpha = n+1, beta = n+2;

d[alpha] = 0;

for(int u = 1; u <= n;++u){

scanf("%d", &d[u]);

int x;

do{

scanf("%d", &x);

if(x > 0){

add\_edge(&G, x, u);

}

}while(x>0);

}

for(int u = 1; u <=n;++u){

int deg\_neg = 0;

int deg\_pos = 0;

for(int x = 1; x <= n;++x){

if(adjacent(&G, x, u)){

deg\_neg++;

}

if(adjacent(&G, u, x)){

deg\_pos++;

}

}

if(deg\_neg == 0){

add\_edge(&G, alpha, u);

}

if(deg\_pos == 0){

add\_edge(&G, u, beta);

}

}

List L;

makeNull(&L);

TopoSort(&G, &L);

int t[MAXN];

t[alpha] = 0;

for(int i = 1; i <= L.size;++i){

int u = L.data[i];

t[u] = -oo;

for(int x = 1; x <= G.n;++x){

if(adjacent(&G, x, u)){

t[u] = max(t[u],t[x]+d[x]);

}

}

}

int T[MAXN];

T[beta] = t[beta];

for(int j = L.size - 2; j >= 0; --j){

int u = L.data[j];

T[u] = +oo;

for(int v = 1; v <= G.n;++v){

if(adjacent(&G, u, v)){

T[u] = min(T[u],T[v]-d[u]);

}

}

}

}

**----------CHU LIU------------**

void buildH(Graph \*pG, int root, Tree \*pH){

init\_tree(pH, pG->n);

for(int e = 0; e < pG->m; ++e){

int u = pG->edges[e].u;

int v = pG->edges[e].v;

int w = pG->edges[e].w;

int link = pG->edges[e].link;

if(w < pH->weight[v]){

pH->parent[v] = u;

pH->weight[v] = w;

pH->link[v] = link;

}

}

pH->parent[root] = -1;

pH->parent[root] = 0;

}

int id[MAXN];

int find\_cycle(Tree \*pH, int root){

int color[MAXN];

for(int i = 1; i <=pH->n;++i){

id[i] = -1;

color[i] = -1;

}

int no = 0;

for(int i = 1; i <= pH->n; ++i){

int u = i;

while(u != root && id[u] == -1 && color[u]!=i){

color[u] = i;

u = pH->parent[u];

}

if(color[u] == i){

no++;

int v = pH->parent[u];

while(v != u){

id[v] = no;

v = pH->parent[v];

}

id[u] = no;

}

}

return no;

}

void contract(Graph \*pG, Tree \*pH, int no, Graph \*pG1){

init\_graph(pG1, no);

for(int e = 0; e < pG->m;++e){

int u = pG->edges[e].u;

int v = pG->edges[e].v;

int w = pG->edges[e].w;

if(id[u] != id[v]){

add\_edge(pG1, id[u], id[v], w - pH->weight[v],e);

}

}

}

void expand(Tree \*pH, Graph \*pG1, Tree \* pH1){

for(int i = 1; i <= pH->n; ++i){

if(pH->parent[i] != -1){

Edge pe = pG1->edges[pH->link[i]];

pH1->parent[pe.v] = pe.u;

pH1->weight[pe.v] += pH->weight[i];

pH1->link[pe.v] = pe.link;

}

}

}

#define MAXIT 10

void ChuLiu(Graph \*pG0, int s, Tree \*pT){

Graph G[MAXIT];

Tree H[MAXIT];

int t = 0;

int root = s;

G[0] = \*pG0;

while(1){

buildH(&G[t], root, &H[t]);

int no = find\_cycle(&H[t], root);

if(no == 0){

break;

}

for(int i = 1; i <= H[t].n;++i){

if(id[i] == -1){

id[i] = ++no;

}

}

contract(&G[t],&H[t], no, &G[t+1]);

root = id[root];

t++;

}

for(int k = t; k > 0; --k){

expand(&H[k], &G[k-1], &H[k-1]);

}

\*pT = H[0];

}

**---------------------------FORD FULKERSON----------------------**

int FordFulkerson(Graph \*pG, int s, int t){

init\_flow(pG);

int max\_flow = 0;

Queue Q;

do{

for(int i = 1; i <= pG->n; ++i){

labels[i].dir = 0;

}

labels[s].dir = +1;

labels[s].parent = s;

labels[s].sigma = oo;

make\_null\_queue(&Q);

enqueue(&Q, s);

int found = 0;

while(!empty(&Q)){

int u = front(&Q);

dequeue(&Q);

for(int v = 1; v <= pG->n; ++v){

if(pG->C[u][v]!=NO\_EDGE && labels[v].dir==0 && pG->F[u][v] < pG->C[u][v]){

labels[v].dir = +1;

labels[v].parent = u;

labels[v].sigma = min(labels[u].sigma, pG->C[u][v] - pG->F[u][v]);

enqueue(&Q, v);

}

}

for(int p = 1; p <= pG->n; ++p){

if(pG->C[p][u] != NO\_EDGE && labels[p].dir == 0 && pG->F[p][u] > 0){

labels[p].dir = -1;

labels[p].parent = u;

labels[p].sigma = min(labels[u].sigma, pG->F[p][u

enqueue(&Q, p);

}

}

if(labels[t].dir != 0){

found = 1;

break;

}

}

if(found){

int sigma = labels[t].sigma;

int u = t;

while(u != s){

int p = labels[u].parent;

if(labels[u].dir > 0){

pG->F[p][u] += sigma;

}else{

pG->F[p][u] -= sigma;

}

u = p;

}

max\_flow += sigma;

}else{

break;

}

}while(1);

return max\_flow;

}