# CTDL

**\*List:**

typedef int ElementType;

typedef struct

{

ElementType data[MAXLENGTH];

int size;

} List;

void makenullList(List \*L)

{

L->size = 0;

}

void pushback(List \*L, ElementType x)

{

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

L->size++;

}

ElementType element\_at(List L, int i)

{

return L.data[i - 1];

}

**\*QUEUE**

typedef struct

{

int data[MAX\_ELEMENTS];

int front, rear;

} Queue;

void make\_null\_queue(Queue \*Q)

{

Q->front = 0;

Q->rear = -1;

}

void push(Queue \*Q, int x)

{ Q->rear++;

int countlist(List L)

{

return L.size;

}

void copylist(List \*L1, List \*L2)

{

List temp = \*L2;

\*L2 = \*L1;

\*L1 = temp;

}

Q->data[Q->rear] = x;

}

int top(Queue \*Q)

{

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

}

void pop(Queue \*Q)

{

Q->front++;

}

int empty(Queue \*Q)

{

return Q->front > Q->rear;

}

**\*Stack**

typedef int ElementType;

typedef struct

{

int data[MAXLENGTH];

int size;

} Stack;

void makenullStack(Stack \*S)

{

S->size = 0;

}

ElementType top(Stack S)

{

return S.data[S.size - 1];

}

**\*Ma trận D-D(kề)**

typedef struct

{

int n;

int A[MAXN][MAXN];

} Graph;

void init\_graph(Graph \*G, int n)

{

int i, j;

G->n = n;

for (i = 1; i <= n; i++)

for (j = 1; j <= n; j++)

G->A[i][j] = 0;

}

void add\_edge(Graph \*G, int x, int y)

{

G->A[x][y] = 1;

G->A[y][x] = 1;

}

int adjacent(Graph G, int x, int y)

{

void pop(Stack \*S)

{

S->size--;

}

int empty(Stack S)

{

return S.size == 0;

}

void push(Stack \*S, ElementType x)

{

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

S->size++;

}

if (G.A[x][y] == 1)

return 1;

return 0;

}

List neighbors(Graph \*G, int x)

{

List L;

makenullList(&L);

int i;

for (i = 1; i <= G->n; i++)

if (adjacent(\*G, x, i)== 1)

pushback(&L, i);

return L;

}

int degree(Graph G, int x)

{

int deg = 0;

int i;

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

if (G.A[x][i] != 0)

deg += G.A[x][i];

return deg;

}

**\*DSCung**

typedef struct

{

int x, y;

int w;

} Edge;

typedef struct

{

int n, m;

Edge edges[MAX\_EDGES];

} Graph;

**\*Duyệt rộng**

void bfs(Graph \*G)

{

Queue q;

make\_null\_queue(&q);

int mark[MAXN];

int i;

for (i = 1; i <= G->n; i++)

mark[i] = 0;

for (i = 1; i <= G->n; i++)

if (mark[i] == 0)

{

mark[i]=1;

push(&q,i);

printf("%d\n",i);

while (!empty(&q))

{

int i;

int u = top(&q);

pop(&q);

List list = neighbors(G, u);

for (i = 1; i <= list.size; i++)

void init\_graph(Graph \*G, int n)

{

G->n = n;

G->m = 0;

}

void add\_edge(Graph \*G, int x, int y, int w)

{

G->edges[G->m].x = x;

G->edges[G->m].y = y;

G->edges[G->m].w = w;

G->m++;

}

if (mark[element\_at(&list, i)] == 0)

{

printf("%d\n",

element\_at(&list, i)); mark[element\_at(&list, i)]=1;

push(&q, element\_at(&list, i));

}

}

}

}

}

**\*Duyệt sâu (Stack)**

void dfs(Graph \*G)

{

int mark[MAXELEMENT];

int i, j;

Stack S;

makenullStack(&S);

for (i = 0; i <= G->n; i++)

mark[i] = 0;

for (i = 1; i <= G->n; i++)

if (mark[i] == 0)

{

push(&S, i);

while (!emptys(&S))

{

int u = top(&S);

pop(&S);

if (mark[u] == 1)

continue;

**\*Duyệt sâu (đệ quy)**

int mark[MAXELEMENT];

void visit(Graph \*G, int u)

{

if (mark[u] == 1)

return;

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

mark[u] = 1;

List list = neighbors(G, u);

int i;

for (i = 1; i <= list.idx; i++)

visit(G, element\_at(&list, i));

}

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

mark[u] = 1;

List list;

makenullList(&list);

list = neighbors(G, u);

for (j = 1; j <= list.size; j++)

push(&S, element\_at(&list, j));

}

}

}

void dfs\_dequy(Graph \*G)

{

int i;

for (i = 0; i < G->n; i++)

mark[i] = 0;

for (i = 1; i <= G->n; i++)

if (mark[i] == 0)

visit(G, i);

}

- Tìm cha thêm int parent[MAXN];

- Riêng Stack thì thay đổi cấu trúc

typedef struct{ typedef struct{

int u; int top\_idx;

int parent; ELEMENT\_TYPE A[MAXELEMENT];

} ELEMENT\_TYPE; } Stack;

**\*KTra Chu trình (DSC)**

int parent[MAXELEMENT];

int findRoot(int u)

{

while (u != parent[u])

u = parent[u];

return u;

}

int chutrinh(Graph \*G)

{

int i;

for (i = 1; i <= G->n; i++)

parent[i] = i;

for (i = 0; i < G->m; i++)

{

int r\_u = findRoot(G->edges[i].x);

int r\_v = findRoot(G->edges[i].y);

if (r\_u != r\_v)

parent[r\_v] = r\_u;

else

return 0;

}

return 1;

}//0 thì có chu trình

**\*Tìm BPLT**

int min(int x, int y)

{

return (x < y) ? x : y;

}

Stack S;

int on\_stack[MAXN];

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

int count = 0;//Lưu số BPLT

void strongconnect(Graph \*G, int x)

{

int i;

num[x] = min\_num[x] = idx;

idx++;

push(&S, x);

on\_stack[x] = 1;

List list = neighbors(G, x);

int j;

for (j = 1; j <= list.size; j++)

{

int y = element\_at(&list, j);

if (num[y] == -1)

{

strongconnect(G, y);

min\_num[x] = min(min\_num[x], min\_num[y]);

}

else if (on\_stack[y])

min\_num[x] = min(min\_num[x], num[y]);

}

if (num[x] == min\_num[x])

{

count++;

int w;

do

{

w = top(&S);

pop(&S);

on\_stack[w] = 0;

} while (w != x);

}

}

**\*Dijkstra**

int mark[MAXN];

int pi[MAXN];

int p[MAXN];

void Dijkstra(Graph \*G, int s)

{

int i, j, it;

for (i = 1; i <= G->n; i++)

{

pi[i] = INFINITY;

mark[i] = 0;

}

pi[s] = 0;

p[s] = -1;

for (it = 1; it < G->n; it++)

{

int min\_pi = INFINITY;

for (j = 1; j <= G->n; j++)

if (mark[j] == 0 && pi[j] < min\_pi)

{

min\_pi = pi[j];

i = j;

}

**\*BellmanFord**

int pi[MAXN];

int p[MAXN];

void BellmaFord(Graph \*G, int s)

{

int j, it;

int i;

for (i = 1; i <= G->n; i++)

pi[i] = INFINITY;

pi[s] = 0;

p[s] = -1;

for (it = 1; it < G->n; it++)

for (j = 0; j < G->m; j++){

mark[i] = 1;

for (j = 1; j <= G->n; j++)

if (G->w[i][j] != NOEDGE && mark[j] == 0)

{

if (pi[i] + G->w[i][j] < pi[j])

{

pi[j] = pi[i] + G->w[i][j];

p[j] = i;

}

}

}

}

//Từ s tới t

// printf("%d", pi[t]+pi[s]);

int x = G->edges[j].x;

int y = G->edges[j].y;

int w = G->edges[j].w;

if (pi[x] + w < pi[y])

{

pi[y] = pi[x] + w;

p[y] = x;

}

}

}

**\*Chu trình âm**

void checkChuTrinhAm(Graph \*G, int s)

{

BellmaFord(G, s);

int j;

for (j = 0; j < G->m; j++)

{

int x = G->edges[j].x;

int y = G->edges[j].y;

int w = G->edges[j].w;

if (pi[x] + w < pi[y])

{

printf("YES");

return;

}

}

printf("NO");

}

**\*Ranking**

int rank[MAXVERTICLES];

int k = 0;

void ranking(Graph \*G)

{

int d[MAXVERTICLES];

int x, u;

for (u = 1; u <= G->n; u++)

{

d[u] = 0;

rank[u] = 0;

}

for (x = 1; x <= G->n; x++)

for (u = 1; u <= G->n; u++)

if (G->A[x][u] != 0)

d[u]++;//d[v]++

List S1, S2;

makenullList(&S1);

for (u = 1; u <= G->n; u++)

if (d[u] == 0)

pushback(&S1, u);

**\*Ranking Sort**

void topo\_sort(Graph \*G, List \*L)

{

ranking(G);

makenullList(L);

int i, j;

for (i = 0; i <= k; i++)

for (j = 1; j <= G->n; j++)

if (rank[j] == i)

pushback(L, j);

}

int i;

while (S1.size > 0)

{

makenullList(&S2);

for (i = 1; i <= S1.size; i++){

//Ngược: (i = S1.size; i >= 1; i--)

int u = element\_at(&S1, i);

rank[u] = k;

int v;

for (v = 1; v <= G->n; v++)

if (G->A[u][v] != 0){

d[v]--;

if (d[v] == 0)

pushback(&S2, v);

}

}

copylist(&S1, &S2);

k++;

}

}

**\*Thi công**

void main(){

//Đọc đồ thị

Graph G;

int n, u, x, v, j;

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

scanf("%d", &n);

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

//Đỉnh alpha = 0

d[n + 1] = 0;

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

{

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

do

{

scanf("%d", &x);

if (x > 0)

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

} while (x > 0);

}

//Thêm cung từ alpha tới đỉnh bậc 0

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

{

int deg\_neg = 0;

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

if (G.A[x][u] > 0)

deg\_neg++;

if (deg\_neg == 0)

add\_edge(&G, n + 1, u);

}

//Thêm cung từ đỉnh cuối tới beta

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

{

int deg\_pos = 0;

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

if (G.A[u][x] > 0)

deg\_pos++;

if (deg\_pos == 0)

add\_edge(&G, u, n + 2);

}

List L;

int i;

//Sắp sếp theo rank lưu vào List

topo\_sort(&G, &L);

//Tính t[u]

int t[MAXVERTICLES];

t[n + 1] = 0;

for (i = 2; i <= L.size; i++)

{

int u = element\_at(&L, i);

t[u] = -1;

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

if (G.A[x][u] > 0)

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

}

//Tính T[u]

int T[MAXVERTICLES];

T[n + 2] = t[n + 2];

for (i = L.size - 1; i >= 1; i--)

{

int u = element\_at(&L, i);

T[u] = INFINITY;

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

if (G.A[u][x] > 0)

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

}

}

//Tìm đường Gantt

//for (i = 1; i <= n+2; i++)

// if (t[i] == T[i])

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

**\*Kruskal**

int parent[MAXN];

int findRoot(int u)

{

return (parent[u] == u) ? u : findRoot(parent[u]);

}

void swap(Edge \*e1, Edge \*e2)

{

Edge temp = \*e1;

\*e1 = \*e2;

\*e2 = temp;

}

int Kruskal(Graph \*G, Graph \*T)

{

//Edges Sort

int i, j;

for (i = 0; i <= G->m - 1; i++)

for (j = i + 1; j <= G->m - 1; j++)

if (G->edges[i].w > G->edges[j].w)

swap(&(G->edges[i]), &(G->edges[j]));

//Kruskal

init\_graph(T, G->n);

int u;

for (u = 1; u <= G->n; u++)

parent[u] = u;

int sum = 0;

int e;

for (e = 0; e <= (G->m - 1); e++)

{

int u = G->edges[e].x;

int v = G->edges[e].y;

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

int root\_u = findRoot(u);

int root\_v = findRoot(v);

if (root\_u != root\_v)

{

add\_edge(T, u, v, w);

parent[root\_v] = root\_u;

sum += w;

}

}

return sum;

}

**\*PRIM**

int mark[MAXN];

int pi[MAXN];

int p[MAXN];

int Prim(Graph \*G, Graph \*T){

init\_graph(T, G->n);

int i, u, v;

for (i = 1; i <= G->n; i++)

{

pi[i] = INFINITY;

mark[i] = 0;

}

pi[1] = 0;

mark[1] = 1;

for (v = 1; v <= G->n; v++)

if (G->w[1][v] != NOEDGE)

{

pi[v] = G->w[1][v];

p[v] = 1;

}

int sum = 0;

for (i = 1; i < G->n; i++)

{

int min\_dist = INFINITY, min\_u;

for (u = 1; u <= G->n; u++)

if (mark[u] == 0)

if (min\_dist > pi[u]){

min\_dist = pi[u];

min\_u = u;

}

u = min\_u;

mark[u] = 1;

add\_edge(T, p[min\_u], min\_u, min\_dist);

sum += min\_dist;

for (v = 1; v <= G->n; v++)

if (G->A[u][v] != NOEDGE && mark[v] == 0)

if (pi[v] > G->w[u][v])

{

pi[v] = G->w[u][v];

p[v] = u;

}

}

return sum;

}

**\*Luồng**

//Thiết lập cấu trúc

typedef struct{

int C[MAXN][MAXN];

int F[MAXN][MAXN];

int n;

} Graph;

typedef struct{

int dir;

int pre;

int sigma;

} Label;

Label labels[MAXN];

void init\_graph(Graph \*G, int n){

G->n = n;

}

void init\_flow(Graph \*G){

int u, v;

for (u = 1; u <= G->n; u++)

for (v = 1; v <= G->n; v++)

G->F[u][v] = 0;

}

void add\_edge(Graph \*G, int x, int y, int w){

G->C[x][y] = w;

}

int min(int x, int y){

return (x < y) ? x : y;

}

int FordFullkerson(Graph \*G, int s, int t){

init\_flow(G);

Queue Q;

int sum\_flow = 0;

do{

//Xoá nhãn rồi gán nhãn cho s

int u;

for (u = 1; u <= G->n; u++)

labels[u].dir = 0;

labels[s].dir = +1;

labels[s].pre = s;

labels[s].sigma = INF;

make\_null\_queue(&Q);

push(&Q, s);

//Lặp gán nhãn cho các đỉnh

int found = 0;

while (!empty(&Q)){

//Lấy 1 đỉnh trong Queue

int x = top(&Q);

pop(&Q);

int v;

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

//Xét đỉnh kề x, cung thuận

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

{

labels[v].dir = +1;

labels[v].pre = x;

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

push(&Q, v);

}

//Xét đỉnh kề x, cung nghịch

if (labels[v].dir == 0 && G->C[v][x] != NO\_EDGE && G->F[v][x] > 0)

{

labels[v].dir = -1;

labels[v].pre = x;

labels[v].sigma = min(labels[x].sigma, G->F[x][v]);

push(&Q, v);

}

}

//t được gán nhãn thì thoát while 2 để tăng luồng

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

found = 1;

break;

}

}

//Tăng luồng

if (found == 1){

int x = t;

int sigma = labels[t].sigma;

sum\_flow += sigma;

while (x != s){

int u = labels[x].pre;

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

//Tăng

G->F[u][x] += sigma;

}else{

/Giảm

G->F[x][u] -= sigma;

}

x = u;

}

}else{

break; //Thoát while 1

}

} while (1);

return sum\_flow; //return LCD

}

//In 2 lát cắt

// printf("X0:");

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

// if (labels[u].dir != 0)

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

// }

// printf("\nY0:");

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

// if (labels[u].dir == 0)

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

// }

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |

# IDEAS

1. **Ma trận đỉnh - đỉnh**

Bảng 1..n, có cung thì thêm G->A[u][v]=1 or G->A[u][v]=w với trọng số

1. **Danh sách cung**

Dùng cho Bellman-Ford, Kruskal.

1. **Duyệt rộng**

Dùng Queue:

- Khởi tạo mark[u]=0

- Đưa 1 vào frontier

- Loop đếm empty queue:

+ Lấy phần tử đầu tiên frontier ra duyệt

+ Lấy neighbor

+ Duyệt đỉnh kề

1. **Duyệt sâu**
   1. **Ngăn xếp:** Như duyệt rông mà dùng *Stack frontier;*
   2. **Đệ quy:**

- Khởi tao mark[u]=0

- Nếu mark[u]==0 => Vist đỉnh u (1..n):

+ Duyệt u

+ Lấy neighbor

+ Duyệt đỉnh kề = visit()

1. **Kiểm tra chu trình**

parent[u] lưu đỉnh cha của u

Khởi tạo parent[u]=u => mỗi đỉnh là 1 BPLT

Find root từng cung trong DSC, nếu 2 root trùng nhau sẽ tạo ra chu trình => return 0;

1. **Dijkstra**

Khởi tạo:

+ p[i] = +oo (1..n);

+ pi[s]=0; p[s]=-1;

+ mark[i]=0 (1..n);

Lặp n-1 lần:

+ Chọn đỉnh chưa đánh dấu (mark[i]==0) có pi[i] nhỏ nhất;

+ mark[i]=1;

+ for(1->n){

Nếu pi[i] + G->A[i][j] < pi[j];

pi[j]=pi[i]+G->A[i][j];

p[j]=i;

}

1. **Bellman – Ford**

Khởi tạo:

+ pi[i] = + oo;

+ pi[s]= 0;p[s]=-1;(s: đỉnh bắt đầu duyệt vd:1);

Lặp n-1 lần:

+ for(tất cả các cung)

if( pi[u] + w < pi[v] )

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

p[v]=u;

1. **Kiểm tra chu trình âm**

Duyệt qua các cung 1 lần nữa:

If( pi[u] + w < pi[v] ) //có chu trình âm

1. **Ranking**

Tính d[u] (G->A[u][v] ==1 : d[v]++);

Đưa đỉnh I có d[i] == 0 vào S(List);

K= rank ban đầu mình muốn (0 or 1);

Lặp đến S rỗng (while)

For(các đỉnh u trong S1) //

Rank[u]=k;

For(các đỉnh kề của u) //v: các đỉnh kề của u;ĐK: G->A[u][v]!=0

D[v]--;

Nếu d[v] ==0 thì thêm v vào S2

K++;

Copy S2 vào S1;

- Ngược: Giống như ranking. Nhưng tính d[u] thì đổi d[v]++ thành d[u]++

For(các đỉnh kề của u) thành For(các đỉnh cha của u) //ĐK: G->A[v][u] !=0

1. **Ranking Sort**

Ranking(Graph\* G,List\* L);

Khởi tạo L;

For(i:L->k) //k là bậc lớn nhất ; L là tùy vào lúc đầu khởi tạo k bằng mấy

For(j:1->n)

Nếu rank[j]==k thì push vào List

1. **Thi công**
2. **Kruskal**

Sắp xếp các cung từ nhỏ đến lớn.

Tạo cây T mới vs T->n =G->n;

For(i:1->n) parent[i]=i;

Sum\_w=0;

For(e:1->m)//m là số cung của đồ thị G

Int u=edge[e].u;

Tương tự vs v,w;

Root\_u=findroot(u); root\_v=findroot(v);

Nếu root\_u != root\_v thì thêm cung (u,v) vào cây T và cập nhật lại cha của root\_v = root\_u; sum\_w+=w;

1. **Prim**

Khởi tạo pi[i]= oo; mark[i]=0; //i:1->n;

Gán pi[1]=0; mark[1]=1;

For(v:1->n) nếu có cung (1,v) thì pi[v]=G->A[1][v]; p[v]=1;

Sum=0;

For(i:1->n) //bé hơn(n-1 lần)

Khởi tạo min\_dist=INF, min\_u;

For(u:1->n)

Nếu mark[u] ==0 và min\_dist > pi[u] thì min\_dist=pi[u]; min\_u=u;

U=min\_u;

Mark[u]=1;

Thêm cung (p[u],u) vào cây T;

Sum+= min\_dist;

For(v:1->n)

Nếu có cung(u,v) và mark[v]==0 và pi[v]>G->A[u][v] thì pi[v]=G->A[u][v] ; p[v]=u;