# **CẤU TRÚC LƯU TRỮ DỮ LIỆU**

## LIST/STACK

- Định nghĩa List/Stack

**typedef struct{**

**int data[max];**

**int size;**

**} list;**

- Các hàm hỗ trợ code:

**void makeNull(list \*l){**

**l->size = 0;**

**}**

**int empty(list \*l){**

**return l->size == 0;**

**}**

**void pop(list \*l){** //Stack

**l->size--;**

**}**

**void push(list \*l, int x){** //Stack

**l->data[l->size] = x;**

**l->size++;**

**}**

**int top(list \*l){** //Stack

**return l->data[l->size -1];**

**}**

**int elementAt(list \*l, int p){**

**return l->data[p -1];**

**}**

* *List và Stack được khai báo và sử dụng chung!*

Note:

## QUEUE

- Định nghĩa Queue

**typedef struct {**

**int data[max];**

// Front: Dau hang | Rear: Cuoi hang

**int front, rear;**

**} queue;**

- Các hàm hỗ trợ code:

**void makenullQueue(queue \*q){**

**q->front = 0;**

**q->rear = -1;**

**}**

**int emptyQueue(queue \*q){**

**return (q->front > q->rear);**

**}**

**int topQueue(queue \*q){**

**return q->data[q->front];**

**}**

**void popQueue(queue \*q){**

**q->front++;**

**}**

**void pushQueue(queue \*q, int x){**

**q->rear++;**

**q->data[q->rear] = x;**

**}**

* *Queue được cải tiến từ hàng đợi tịnh tiến*

Note:

# **CẤU TRÚC LƯU TRỮ ĐỒ THỊ**

## ĐỒ THỊ ĐỈNH ĐỈNH

- Định nghĩa cấu trúc

**#define max 100**

**#define inf 9999**

**typedef struct{**

**int a[max][max];**

**int n, m;**

**} graph;**

**-** Các hàm hỗ trợ code -

**void init(graph \*g , int n, int m){**

**int i, j;**

**g->n = n;** // số đỉnh đầu vào

**g->m = m;** // số cung đầu vào

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

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

**g->a[i][j] = 0;**

**}**

**}**

**}**

- Đồ thị vô hướng -

**void add(graph \*g , int x, int y){**

**g->a[x][y] = g->a[y][x] = 1;**

**}**

**int adjacent(graph \*g , int x, int y){**

**return (g->a[x][y] == 1);**

**}**

**int degree(graph \*g , int x){**

**int i, deg = 0;**

**for (i = 1; i <= g->n; i++){**

**if (g->a[i][x] == 1)**

**deg++;**

**}**

**return deg;**

**}**

- Khai báo list ở đây! -

**list neightbors(graph \*g, int x){**

**list l;**

**makeNull(&l);**

**int i;**

**for (i = 1; i <= g->n; i++){**

**if (g->a[i][x] == 1){**

**push(&l, i);**

**}**

**}**

**return l;**

**}**

- Mẫu hàm main -

**int main(){**

**graph g;**

**int n, m, i, u, v;**

**freopen("graph\_data.txt", "r", stdin);**

// Nộp trên else bỏ dòng này!

**scanf("%d%d", &n, &m);**

**init(&g, n, m);**

**for (i=1; i<=m; i++){**

**scanf("%d%d", &u, &v);**

**add(&g, u, v);**

**}**

**}**

Note:

## DANH SÁCH CUNG

- Định nghĩa cấu trúc –

#define max 100

#define inf 9999

typedef **struct{**

**int u, v, w;** // Do thi vo huong trong so w

**} edge;**

**typedef struct{**

**edge edges[max];** // Danh sách cung

**int n;** // Số đỉnh

**int m;** // Số cung

**} graph;**

- Các hàm bổ trợ -

**void init(graph \*g, int n){**

**g->n = n;** // Đồ thị có n đỉnh

**g->m = 0;** // Chưa thêm cung nào

**}**

**void add(graph \*g, int x, int y, int w){**

**g->edges[g->m].u = x;**

// gán 1 đầu

**g->edges[g->m].v = y;**

**g->edges[g->m].w = w;**

**g->m++;** // like size in list

**}**

**int deg(graph \*g, int x){**

**int i, deg = 0;**

**for (i=0; i<g->m; i++){**

**if (g->edges[i].u==x ||**

**g->edges[i].v==x){**

**deg++;**

**}**

**}**

**return deg;**

**}**

**int adjacent(graph \*g, int x, int y){**

**int i;**

**for (i=0; i<g->m; i++){**

**edge tmp = g->edges[i];**

// Lưu biến

// TH1: u = x và v = y

// TH2: u = y và v = x

**if ((tmp.u==x &&**

**tmp.v==y) ||**

**(tmp.u==y &&**

**tmp.v==x)){**

**return 1;**

**}**

**}**

**return 0;**

**}**

- mẫu hàm main -

**int main(){**

**graph g;**

**int n, m, i, u, v, w;**

**freopen("graph\_data.txt", "r", stdin);**

// Nộp trên else thì xóa

**scanf("%d%d", &n, &m);**

**init(&g, n);**

**for (i=1; i<=m; i++){**

**scanf("%d%d%d", &u, &v, &w);**

**add(&g, u, v, w);**

**}**

**return 0;**

**}**

Note:

# **CÁC THUẬT TOÁN DUYỆT ĐỒ THỊ**

## Duyệt đồ thị theo chiều sâu dùng Stack

- Các biến toàn cục -

**#define max 100**

**int mark[max], parent[max];**

* *Hàm gồm x cho đồ thị không liên thông,*
* *Mảng parent[] để vẽ cây duyệt đồ thị.*
* *Tùy theo yêu cầu đề bài mà tinh chỉnh lại hàm!*

**void depthFirstSearch(graph \*g, int x, int parent[]){**

**list s; makeNull(&s);**

**push(&s, x);**

**parent[x] = 0;**

**int connect[max], i;**

**for (i = 1; i <= g->n; i++){**

// Đánh dấu đỉnh trong bộ phận liên thông đã được duyệt

**connect[i] = 0;**

**}**

**while (!empty(&s)){**

**int u = top(&s);**

**pop(&s);**

**if (connect[u]) continue;**

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

**connect[u] = 1**

**mark[u] = 1;**

**list l = neightbors(g, u);**

**for (i=0; i<l.size; i++){**

**int v = l.data[i];**

**if (!connect[v]) {**

**push(&s, v);**

**parent[v] = u;** // Cập nhập đỉnh cha cho v

**}**

**}**

**}**

**}**

## Duyệt đồ thị theo chiều sâu đệ quy

- Các biến toàn cục -

**#define max 100**

**int mark[max], parent[max];**

* *Hàm gồm x cho đồ thị không liên thông,*
* *Mảng parent[] để vẽ cây duyệt đồ thị.*
* *Tùy theo yêu cầu đề bài mà tinh chỉnh lại hàm!*

**void depthFirstSearchRecursive(graph \*g, int u, int p){**

**if (mark[u]) return;**

**printf("Duyet: %d\n", u); // in thu tu duyet cac dinh**

**parent[u] = p;**

**mark[u] = 1;**

**list l = neightbors(g, u);**

**int i, v;**

**for (i=0; i<l.size; i++){**

**v = l.data[i];**

**depthFirstSearchRecursive(g, v, u);**

**}**

**}**

- Muốn kết quả duyệt giống như dùng stack thì đổi hàm -

// Lấy list các đỉnh là hàng xóm của x:

**list neightbors(graph \*g, int x){**

**list l; makeNull(&l);**

**int i;**

**for (i = g->n; i >= 1; i++){**

**if (g->a[i][x] == 1){**  //Đơn đồ thị

**push(&l, i);**

**}**

**}**

**return l;**

**}**

Note:

## Duyệt đồ thị theo chiều rộng dùng Queue

- Các biến toàn cục -

**int mark[max];**

**int parent[max];**

* *Hàm gồm x cho đồ thị không liên thông,*
* *Mảng parent[] để vẽ cây duyệt đồ thị.*
* *Tùy theo yêu cầu đề bài mà tinh chỉnh lại hàm!*

**void breatFirstSearch(graph \*g, int x, int parent[]) {**

**queue q; makenullQueue(&q);**

**pushQueue(&q, x);**

**parent[x] = 0;**

**int connect[max], i;**

**for (i = 1; i <= g->n; i++){**

**connect[i] = 0;**

**}**

**printf("Duyet %d\n", x);**

**connect[x] = 1;**

**while (!emptyQueue(&q)){**

**int u = topQueue(&q); popQueue(&q);**

**list l = neightbors(g, u);**

**for (i=0; i<l.size; i++){**

**int v = l.data[i];**

**if (!connect[v]) {**

**printf("Duyet %d\n", v);**

**pushQueue(&q, v);**

**connect[v] = 1;**

**mark[v] = 1;**

**parent[v] = u;**

**}**

**}**

**}**

**}**

Note:

- Ở hàm main bổ sung thêm các vòng lặp để sử dụng kết quả duyệt -

//Đọc dữ liệu, khởi tạo, …

**for (i=1; i<=n; i++){** // Init các mảng toàn cục

**mark[i] = 0;** // Chưa duyệt

**parent[i] = -1;** // Chưa có đỉnh cha

**}**

**for (i = 1; i <= n; i++){** // Duyệt đồ thị (liên thông hoặc không)

**if (!mark[i]){**

**breatFirstSearch(&g, i, parent);** // Duyệt sâu Stack

**/\*hoặc\*/ depthFirstSearchRecursive(&g, i, parent);** // Duyệt sâu đệ quy

**/\*hoặc\*/ breatFirstSearch(&g, i, parent);** // Duyệt rộng Queue

**}**

**}**

**for (i=1; i<=n; i++){** // In các parent các đỉnh để vẽ cây duyệt

**printf("%d %d\n", i, parent[i]);**

**}**

Note:

# **THUẬT TOÁN TÌM BỘ PHẬN LIÊN THÔNG MẠNH**

1. - Các biến toàn cục –

**int num[max], min\_num[max], on\_stack[max], idx = 1;**

**list s;**

**int smaller(int a, int b){**

**return (a<=b) ? a: b;**

**}**

**int initAlm(int n, int m){**

**int i;**

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

**num[i] = min\_num[i] = on\_stack[i] = -1;**

**}**

**makeNull(&s);**

**}**

**void tarjan(graph \*g, int x) {**

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

**push(&s, x);**

**on\_stack[x] = 1;**

**list l = neightbors(g, x);**

**int j;**

**for (j = 0; j < l.size; j++) {**

**int y = l.data[j];**

**if (num[y] < 0){**

**tarjan(g, y);**

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

**}**

**else if (on\_stack[y]){**

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

**}**

**}**

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

**printf("%d la dinh Khop\nBo phan lien thong manh gom: ", x);**

**do {**

**int w = top(&s); pop(&s);**

**printf("%d ", w);**

**on\_stack[w] = 0;**

**} while (w != x);**

**}**

**}**

# **THUẬT TOÁN TÌM ĐƯỜNG ĐI NGẮN NHẤT(Đồ thị vô hướng trọng số)**

## MooreDijksta(Đỉnh bất kì - Trọng số dương)

**int mark[max], pi[max], p[max];**

**void moore(graph \*g, int s){**

**int u, v, i;**

**for (u=1; u<=g->n; u++){ // Kh?i t?o các m?ng**

**pi[u] = inf;**

**mark[u] = 0;**

**}**

**pi[s] = 0; // di tu dinh s, ve den dinh s -> = 0**

**p[s] = -1; // dinh bat dau khong co parent -> = -1**

**for (i=1; i<g->n; i++){ // Duy?t n - 1 lan**

**int j, min\_pi = inf;**

**for (j=1; j<=g->n; j++){**

**// Tìm ??nh ch?a duy?t có giá tr? min\_pi**

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

**min\_pi = pi[j];**

**u = j;**

**}**

**}**

**mark[u] = 1; // ?ánh d?u ?ã duy?t xong ??nh ?ó**

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

**if (g->a[u][v] && !mark[v]){**

**if (pi[u] + g->a[u][v] < pi[v]){**

**// cap nhap lai pi va p cho tung dinh**

**pi[v] = pi[u] + g->a[u][v];**

**p[v] = u;**

**}**

**}**

**}**

**}**

**}**

**void answer(graph \*g){**

**int i;**

**for (i = 1; i<=g->n; i++){**

**printf("Dinh: %d - parent: %d - Chi phi: %d\n", i, p[i], pi[i]);**

**}**

**}**

**Đề thi thử**: *Tính độ dài đường đi ngắn nhất từ đỉnh start đến đỉnh end, in lộ trình đi qua các đỉnh trên quãng đường di chuyển*

* Gọi hàm moore(&g, start, end) => Tạo ra mảng pi[] là danh sách đường đi ngắn nhất từ start đến các đỉnh còn lại
* Khởi tạo thêm mảng k[max], t = 0, hàm parent;

void parent(Graph \*G, int x){

t++;

if (p[x] == -1){

k[t] = x;

return;

}

else{

k[t] = x;

parent(G, p[x]);

}

}

Cụ thể ở hàm main:

moore(&g, start);

int check = 1;

for (e = 1; e<=n; e++){

if (pi[e] == inf) { // Kiem tra lien thong

check = 0;

break;

}

}

if (check){

int i;

printf("%d \n", pi[end]);

parent(&g, end);

for (i = t; i >= 1; i--) printf("%d ", k[i]);

}

else printf("-1"); // khong lien thong

## BellmanFord(Đỉnh bất kì - Trọng số âm)

**int pi[max], p[max];**

**void bellmanFord(graph \*g, int s, int e){**

**int u, v, w, it, k;**

**for (u=1; u<=g->n; u++) pi[u] = inf;**

**pi[s] = 0; // di tu dinh s, ve den dinh s -> = 0**

**p[s] = -1; // dinh bat dau khong co parent -> = -1**

**for (it=1; it<g->n; it++){ // duyet n -1 lan**

**for (k = 0; k<g->m; k++){ // k = 0 do cau truc graph-List-edge**

**u = g->edges[k].u;**

**v = g->edges[k].v;**

**w = g->edges[k].w;**

**if (pi[u] + w < pi[v]){ // cap nhap lai pi, p cho các dinh**

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

**p[v] = u;**

**}**

**if (pi[v] + w < pi[u]){ // cap nhap lai pi, p cho các ??nh**

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

**p[u] = v;**

**}**

**}**

**}**

**printf("%d\n", pi[e]);**

**int t = e, j = 0, i, tree[max];**

**while(t != s){**

**tree[j] = t;**

**t = p[t];**

**j++;**

**}**

**tree[j] = s;**

**for (i = j; i >= 0; i--){**

**printf("%d ", tree[i]);** // In hành trình duyet

**}**

**}**

**int checkCycleNegative(graph \*g){**  // Kiếm tra chu trình âm

**int k, u, v, w;**

**for (k = 0; k<g->m; k++){**

**u = g->edges[k].u;**

**v = g->edges[k].v;**

**w = g->edges[k].w;**

**if (pi[u] != inf && pi[u] + w < pi[v]){ //**Phát hiện chu trình âm

**return 1;**

**}**

**}**

**return 0;**

**}**

**void answer(graph \*g){**

**int i;**

**for (i = 1; i<= g->n; i++){**

**printf("pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);**

**}**

**}**

# **THUẬT TOÁN PHÂN ĐÔI ĐỒ THỊ**

- Các biến toàn cục -

**#define white -1**

**#define blue 0**

**#define red 1**

**int color[MAX], conflict;**

## đồ thị vô hướng

**void colorize(graph \*g, int u, int c) {**

**color[u] = c;**

**int i;**

**list l = neightbors(&g, u);**

**for (i = 0; i < l.size; i++){**

**int v = l.data[i];**

**if (color[v] == white) colorize(g, v, !c);**

**else if (color[v] == c) {**

**conflict = 1;** //Dung do: 2 dinh ke to cung mau

**return;**

**}**

**}**

**}**

**int checkBipartite(graph \*g){**

**int i;**

**for (i=1; i<=g->n; i++){**

**color[i] = white;** // khoi tao mang chua duyet

**}**

**conflict = 0;** // Không đụng độ

**colorize(g, 1, blue);**

**return conflict;**

**}**

**Đề thi thử:** *Kiểm tra phân chia đồ thị, in các đỉnh trong từng nhóm*

* Xử lý ở hàm main:

int can = checkBipartite(&g);

if (!can){

int a = 0, b =0;

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

if (color[i] == 0) a++; // màu blue

else if (color[i] == 1) b++; // màu red

}

printf(“%d %d, a, b);

// mặc khác nếu đề kêu liệt kê các đỉnh trong từng nhóm:

1. Tạo 2 list a, list b;

2. Tại mỗi if thì push từng đỉnh vào từng list phù hợp => break khỏi for sau đó in từng list ra

}

else{

printf(“Khong the phan chia”);

}

Note:

# **THUẬT TOÁN TÔ MÀU ĐỒ THỊ**

- Các biến toàn cục -

**#define white 0** // dinh chua duoc duyet

**#define black 1** // dinh da duyet (gom dinh do va cac hang xom cua no)

**#define gray 2** // dinh dang duoc duyet

**int color[max], cycle;**

## Đồ thị vô hướng

**void coloring(graph \*g, int x, int parent){** // Có thêm biến parent

**color[x] = gray;**

**int i;**

**list l = neightbors(&g, x);**

**for (i = 0; i < l.size; i++){**

**int y = l.data[i];**

**if (y == parent) continue;** // Điểm khác so với vô hướng

**if (color[y]==gray){**

**cycle = 1;**

**return;**

**}**

**if (color[y] == white) coloring(g, y, x);** // Gọi đệ quy

**}**

**color[x] = black;**

**}**

**int checkCycle(graph \*g){**

**int i;**

**for(i=1; i<=g->n; i++){**

**color[i] = white;** // Khởi tạo mảng chưa duyệt

**}**

**cycle = 0;** // không tồn tại chu trình

**for(i=1; i<=g->n; i++){**

**if (color[i] == white) coloring(g, i, 0);**

**}**

**return cycle;**

**}**

## Đồ thị có hướng

**void coloring(graph \*g, int u){**

**color[u] = gray;**

**int i;**

**list l = neightbors(&g, u);**

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

**int v = elementAt(&l, i);**

**if (color[v]==gray){**

**cycle = 1;**

**return;**

**}**

**if (color[v] == white){**

**coloring(g, v);**

**}**

**}**

**color[u] = black;**

**}**

**int checkCycle(graph \*g){**

**int i;**

**for(i=1; i<=g->n; i++){**

**color[i] = white;**

**}**

**cycle = 0;**

**coloring(g, 1);**

**return cycle;**

**}**

* Như vậy có hướng và vô hướng khác nhau ở chỗ có parent và cách gọi hàm

Note:

# **THUẬT TOÁN XẾP HẠNG ĐỒ THỊ**

## RANKING(Đồ thị có hướng trọng số)

**int rank[max];**

**void copyList(list \*s1, list \*s2){** // makenull(s1), copy s2 cho vao s1

**makenull(s1);**

**int i;**

**for (i=0; i<s2->size; i++){**

**push(s1, s2.data[i]);** // copy s2 vao s1

**}**

**}**

**void ranking(graph \*g){**

**int d[max], i, u, v;** // d[] luu tru bac vao các đỉnh trong graph

**for (u=1; u<=g->n; u++) d[u] = 0;**

**for (i = 1; i<=g->n; i++){**

**for (u = 1; u<= g->n; u++){**

**if (g->a[u][i]){**

**d[i]++;** // Tính bậc vào của từng đỉnh

**}**

**}**

**}**

**list s1, s2; makeNull(&s1);**

**for (u = 1; u<= g->n; u++){**

**if (!d[u]) push(&s1, u);** //Thêm dinh root ban đầu vào s1

**}**

**int k = 1;**

**while (s1.size){**

**makenull(&s2);**

**for (i = 0; i<s1.size; i++){**

**u = s1.data[i];**

**rank[u] = k; // xep hang cho cong viec**

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

**if (g->a[u][v]){**

**d[v]--;** // Giảm bậc vào của các neightbors của u

**if (!d[v]) push(&s2, v);** // push nó vào s2

**}**

**}**

**}**

**copyList(&s1, &s2);** // s1 = s2(copy de chay tiep while)

**k++;** // tang rank cho lan xep hang tiep theo

**}**

**}**

**void Answer\_Ranking(graph \*g){** // print kết quả

**int i;**

**for (i = 1; i<= g->n; i++){**

**printf("Dinh %d - Rank %d\n", i, rank[i]);**

**}**

**}**

## BÀI TOÁN TỔ CHỨC THI CÔNG(Ranking + đồ thị có hướng trọng số)

- Các biến toàn cục –

**int t[max];** // Thời gian bắt đầu công việc sớm nhất

**int T[max];** // Thời gian bắt đầu công việc muộn nhất

**void add(graph \*g, int u, int v, int w){**

**g->a[u][v] = w;**

**}**

**int degreeRa(graph \*g, int x){**

**int deg = 0, i;**

**for (i = 1; i <= g->n; i++){**

**if (g->a[x][i]) deg++;**

**}**

**return deg;**

**}**

**int degreeVao(graph \*g, int x){**

**int deg = 0, i;**

**for (i = 1; i <= g->n; i++){**

**if (g->a[i][x]) deg++;**

**}**

**return deg;**

**}**

**void topoSort(graph \*g, list \*topo) {**

**int rank[max], i;**

**makeNull(topo);**

**for (i = 1; i <= g->n; i++){**

**rank[i] = degreeVao(g, i);**

**if (!rank[i]) push(topo, i);**

**}**

**int t = 0;**

**while(t != g->n){**

**int x = topo->data[t];**

**list l = neighbor(g, x);**

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

**int y = l.data[i];**

**rank[y]--;**

**if (!rank[y])**

**push\_back(topo, y);**

**}**

**t++;** // push hết các đỉnh vào topo

**}**

**}**

**int greater(int a, int b) {**

**return (a>=b) ? a : b;**

**}**

**int smaller(int a, int b) {**

**return (a<=b) ? a : b;**

**}**

**void solve(graph \*g, list topo) {**

**int i, x, n = g->n;**

**t[n + 1] = 0;**

**for (i = 0; i < topo.size; i++) {**

**int u = topo.data[i];**

**t[u] = 0;**

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

**if (g->a[x][u] > 0) {**

**t[u] = greater(t[u], t[x] + g->a[x][u]);**

**}**

**}**

**}**

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

**for (i = topo.size - 2; i >= 0; i--) {**

**int u = topo.data[i];**

**T[u] = inf;**

**for (x = 1; x <= n + 2; ++x) {**

**if (g->a[u][x] > 0) {**

**T[u] = smaller(T[u], T[x] – g->a[u][x]);**

**}**

**}**

**}**

**printf("%d\n", t[n + 2]);**

**for (i = 1; i <=n +2; ++i) {**

**printf("%d-%d\n", t[i], T[i]);**

**}**

**}**

Note:

**int main() {**

**graph g;**

**int n, v, i, a[max];**

// scanf(n), init(g, n)

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

**scanf("%d %d", &a[i], &v);**

**while (v != 0) {**

**add(&g, v, i, a[v]);**

**scanf("%d ", &v);**

**}**

**}**

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

**if (degreeVao(&g, i) == 0) add(&g, n + 1, i, 0);**

**if (degreeRa(&g, i) == 0) add(&g, i, n + 2, a[i]);**

**}**

**list topo;**

**ranking(&g, &topo);**

**push(&topo, n + 2);**

**solve(&g, topo);**

**return 0;**

**}**

Note:

# **THUẬT TOÁN TÌM CÂY KHUNG CÓ TRỌNG LƯỢNG NHỎ NHẤT**

## PRIM(Cây khung có trọng số nhỏ nhất – Đồ thị vô hướng trọng số - Đỉnh Đỉnh)

- Các biến toàn cục -

**int pi[max], p[max], mark[max];**

**void printT(graph \*t){** // xu ly in danh sach cung ra (graph Node-Node)

**int i, j;**

**for (i = 1; i <= t->n ; i++){**

**for (j =1; j <= t->n; j++){**

**if (j >= i && t->a[i][j]){**

**printf("%d %d %d\n", i, j, t->a[i][j]);**

**}**

**}**

**}**

**}**

**int prim(graph \*g, graph \*t){**

**init(t, g->n, g->m);** // khoi tao cay t tree rong

**int i, u, v, sumw = 0;**

**for (u=1; u<= g->n; u++){**

**pi[u] = inf;**

**mark[u] = 0;**

**if(g->a[1][u]){**

**pi[u] = g->a[1][u];** // gan pi[v] = trong so cung (1, v)

**p[u] = 1;** //Dinh trong s gan voi v la dinh 1

**}**

**}**

**pi[1]=0;**

**mark[1] = 1; // Chon dinh 1 va danh dau no**

**for (i = 1; i< g->n; i++){ // Lặp n -1 lần**

**int min\_dist = inf, min\_u;**

**for (u = 1; u <= g->n; u++){**

**if (!mark[u] && min\_dist > pi[u]){**

**min\_dist = pi[u];**

**min\_u = u;**

**}**

**}**

**u = min\_u;** // danh dau u co pi[u] nho nhat

**mark[min\_u] = 1;**

**add(t, min\_u, p[min\_u], min\_dist);**

**sumw += min\_dist;**

// Cap nhap lai mpi va p cua cac dinh ke voi u

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

**if (g->a[u][v] && !mark[v]){**

**if (pi[v] > g->a[u][v]){**

**pi[v] = g->a[u][v];**

**p[v] = u;**

**}**

**}**

**}**

**}**

**return sumw;**

**}**

Note:

## KRUSKAL(cây khung có w min – Đồ thị vô huớng trọng số - Danh Sách Cung)

- Các biến hàm bổ trợ -

**int parent[max];**

**void swap(edge \*a, edge \*b){**

**edge temp = \*a; //** giải thuật dùng danh sách cung

**\*a = \*b;**

**\*b = temp;**

**}**

**void bubbleSort(graph \*g){**

**int i, j;**

**for (i=0; i <g->m -1; i++){**

**for(j=g->m -1; j>=i+1; j--){**

**if (g->edges[j].w < g->edges[j-1].w){** // Sort min->max

**swap(&g->edges[j], &g->edges[j-1]);**

**}**

**}**

**}**

**}**

**int findRoot(int u){**

**while(parent[u] != u) u = parent[u];**

**return u;**

**}**

**int kruskal(graph \*g, graph \*t){**

**bubbleSort(g);**

**int u, e, sumw = 0;**

**init(t, g->n);**

**for (u=1; u<=g->n; u++){**

**parent[u] = u;**

**}**

**for (e=0; e<g->m; e++){**

**int u = g->edges[e].u, v = g->edges[e].v, = g->edges[e].w;**

**int root\_u = findRoot(u), root\_v = findRoot(v);**

**if (root\_u != root\_v){**

**add(t, u, v, w);**

**parent[root\_v] = root\_u;**

**sumw+=w;**

**}**

**}**

**return sumw;**

**}**

Note:

# **THUẬT TOÁN TÌM LUỒNG CỰC ĐẠI**

- Các biến toàn cục và các hàm bổ trợ code -

[Code khai báo queue]

**typedef struct {**

**int c[max][max];**

**int f[max][max];**

**int n, m;**

**} graph;** //Cấu trúc graph mới

**void init(graph\* g, int n) {**

**g->n = n;**

**g->m = 0;**

**}**

**typedef struct {**

**int dir;**

**int pre;**

**int sigma;**

**}label;** //Cấu trúc label

**label labels[max];**

**int marks[max];**

**void initFlow(graph\* g){**

**int i, j;**

**for (i=1; i<=g->n; i++){**

**for (j=1; j<=g->n; j++){**

**g->f[i][j] = 0;**

**}**

**}**

**}**

**int smaller(int a, int b) {**

**return (a<=b) ? a : b;**

**}**

- Thuật toán Ford–Fulkerson-

**int ford(graph \*g, int s, int t) {**

**initFlow(g);**

**queue q;**

**int u, v, sumFlow = 0;**

**while(1) {**

**for (u = 1; u <= g->n; u++) labels[u].dir = 0;**

// Khoi tao nhanh cho lat cat

**labels[s].dir = 1;**

**labels[s].pre = s;**

**labels[s].sigma = inf;**

**makenullQueue(&q);**

**pushQueue(&q, s);**

**int found = 0;**

**while (!emptyQueue(&q)) {**

**u = topQueue(&q);**

**popQueue(&q);**

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

**if (!labels[v].dir && g->c[u][v]**

**&& g->f[u][v] < g->c[u][v]) {**

**labels[v].dir = 1;**

**labels[v].pre = u;**

**labels[v].sigma = smaller(labels[u].sigma,**

**g->c[u][v] - g->f[u][v]);**

**pushQueue(&q, v);**

**}**

**if (!labels[v].dir && g->c[v][u] && g->f[v][u] > 0) {**

**labels[v].dir = -1;**

**labels[v].pre = u;**

**labels[v].sigma = smaller(labels[u].sigma,**

**g->f[u][v]);**

**pushQueue(&q, v);**

**}**

**}**

**if (labels[t].dir) {**

**found = 1;**

**break;**

**}**

**}**

**if (found) {**

**int x = t;**

**int sigma = labels[t].sigma;**

**sumFlow += sigma;**

**while (x != s) {**

**u = labels[x].pre;**

**if (labels[x].dir > 0) g->f[u][x] += sigma;**

**else g->f[x][u] -= sigma;**

**x = u;**

**}**

**} else break;**

**}**

**return sumFlow;**

**}**

note:

- Sử dụng thuật toán tại hàm main-

**int main() {**

**graph g;**

**int n, m, u, v, i, c;**

**freopen("data.txt", "r", stdin);** // Nop bai tren else bo dong nay

**scanf("%d %d", &n, &m);**

**init(&g, n);**

**for (i = 1; i <= m; i++) {**

**scanf("%d%d%d", &u, &v, &c);**

**g.c[u][v] = c;**

**}**

**int maxFlow = ford(&g, 1, n);**

**printf("Max Flow: %d\n", maxFlow);**

**printf("X0: ");** //Tap S lat cat

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

**if (labels[i].dir) printf("%d ", i);**

**}**

**printf("\nY0: ");** //Tap t lat cat

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

**if (!labels[i].dir) printf("%d ", i);**

**}**

**return 0;**

**}**

Note: