**I - Đồ thị**

- Đồ thị vô hướng :

+ Tìm Bậc của các đỉnh :

\* Nếu là danh sách kề : deg(u) = số các đỉnh kề của đỉnh đó

\* -------danh sách cạnh : deg(u) = số lần xuất hiện của đỉnh u trong danh sách cạnh ( cả cột đỉnh đầu và đỉnh cuối)

\*\* Bán bậc vào (deg-(u)) là tổng số lần xuất hiện của u

trên danh sách đỉnh cuối.

\*\* Bán bậc ra (deg+(u)) là tổng số lần xuất hiện của u

trên danh sách đỉnh đầu

\* -------ma trận kề : deg(u) = tông số 1 ở cả hàng và cột của đỉnh u

- Đồ thị có hướng :

+ Tìm bậc của các đỉnh :

\* Nếu là danh sách kề : deg(u) = tổng số các đỉnh kề với đỉnh đó trong ds kề

\*------- danh sách cạnh : deg(u) = số lần xuất hiện của đỉnh đó trong ds cạnh

\*\* deg(-(u)) : là tổng số lần xuất hiện ở ds đỉnh cuối

\*\* deg(+(u)) : -------------------------- ds đỉnh đầu

\*------- ma trận kề : deg(u) = tông số 1 ở cả hàng và cột của đỉnh u

**II - Thuật toán**

**1. DFS**

- Thuật toán :

Dfs(u){

Thăm(u);

vs[u] = 1;

for v thuoc ke(u) do

if (vs[v] = 0) {

e[v]= u; Dfs(v);

}

}

Hàm C / C++ :

* Cài đặt 1 : đệ qui

*// G cho bởi ma trận kề a[i][j]*

int a[100][100], n, u, vs[100], e[100];

void DfsDequy(int u)

{

    int v;

    cout << u << ” ”;

    vs[u] = 1;

    for (v = 1; v <= n; v++)

        if (vs[v] == 0 && a[u][v] == 1)

        {

            e[v] = u;

            DfsDequy(v);

        }

}

* Cài đặt 2 : stack

…

**2. BFS**

- Thuật toán :

Bfs(u)

{

    q = rỗng;

    <Đưa u vào q>;

    vs[u] = 1;

    while q ≠ rỗng

    {

        <Lấy v ra khỏi q>;

        Thăm(v);

    for i thuộc ke(v) do

        if (vs[i] = 0)

        {

            <Đưa i vào q>;

            vs[i] = 1;

            e[i] = v;

        }

    }

}

* Cài đặt hàm = C / C++:

*// G cho bởi ma trận kề a[i][j]*

int a[100][100], n, u, vs[100], e[100], q[100];

void Bfs(int u)

{

    int v, dq = 1, cq = 0;

    cq++;

    q[cq] = u;

    vs[u] = 1;

    while (dq <= cq)

    {

        v = q[dq];

        dq++;

        cout << v << ” ”;

        for (int i = 1; i <= n; i++){

            if (vs[i] == 0 && a[v][i] == 1)

            {

                cq++;

                q[cq] = i;

                vs[i] = 1;

                e[i] = v;

            }

        }

    }

}

3. Duyệt tất cả các đỉnh của đồ thị

- Sử dụng DFS:

void DuyetDfs()

{

    int v;

    for (v = 1; v <= n; v++)

    {

        vs[v] = 0;

        e[v] = 0;

    }

    for (v = 1; v <= n; v++){

        if (vs[v] == 0)

        DfsDequy(v);

    }

}

Sử dụng BFS :

void DuyetBfs()

{

    int v;

    for (v = 1; v <= n; v++)

    {

        vs[v] = 0;

        e[v] = 0;

    }

    for (v = 1; v <= n; v++){

        if (vs[v] == 0)

        Bfs(v);

    }

}

4. Tìm đường đi từ u -> v:

Thuật toán : Tìm đường đi từ đỉnh u đến đỉnh v;

Bước khởi tạo : Tất cả x ∈ G chưa được thăm(vs[x] = 0) và

                chưa xác định cạnh(e[x] = 0);

Bước 1 : Thực hiện DFS(u) / BFS(u);

Bước 2 : (Trả lại kết quả)

             Nếu vs[v] = 0 ⇒ không có đường đi từ u đến v;

Nếu vs[v] = 1 ⇒ xuất đường đi từ u đến v bằng cách sử dụng

    e[];

- Use DFS:

int a[100][100], n, u, v, vs[100], e[100];

void PathDFS(){

    for(int x = 1; x <= n; x++){

        vs[x] = 0; e[x] = 0;

    }

    dfs(u);

    if(vs[v] == 1){

        int t = v;

        while(t > 0){

            cout << t << " <- ";

            t = e[t];

        }

    }

    else{

        cout << "Khong cos duong di"

    }

}

-Use BFS:

int a[100][100], n, u, v, vs[100], e[100];

void PathDFS(){

    for(int x = 1; x <= n; x++){

        vs[x] = 0; e[x] = 0;

    }

    bfs(u);

    if(vs[v] == 1){

        int t = v;

        while(t > 0){

            cout << t << " <- ";

            t = e[t];

        }

    }

    else{

        cout << "Khong cos duong di"

    }

}

5. Kiểm tra tính liên thông của đồ thị

- Vô hướng :

Thuật toán: Kiểm tra tính liên thông của đồ

thị vô hướng

Bước 1: Thực hiện Dfs(1)/Bfs(1)

Bước 2: Tính số lượng k các đỉnh được duyệt;

Bước 3: Nếu k= n xuất 1; nếu k < n xuất 0

-Cài đặt DFS:

int a[100][100], vs[100], n;

int ltDfs()

{

    int v;

    for (v = 1; v <= n; v++){

        vs[v] = 0;

    }

    DfsDequy(1);

    for (v = 1; v <= n; v++){

        if (vs[v] == 0)

            return (0);

    }

    return (1);

}

-Cài đặt BFS:

int a[100][100], vs[100], n;

int ltBfs()

{

    int v;

    for (v = 1; v <= n; v++){

        vs[v] = 0;

    }

    BfsDequy(1);

    for (v = 1; v <= n; v++){

        if (vs[v] == 0) return (0);

    }

    return (1);

}

6. Tìm số thành phần liên thông

- thuật toán:

Bước khởi tạo: k= 0; lt[u]= 0 với mọi đỉnh u;

Bước 1: Nếu mọi đỉnh u đều có lt[u] > 0 thì chuyển

bước 3, ngược lại chọn đỉnh u có lt[u] = 0;

Bước 2: k= k + 1, thực hiện Dfs(u)/Bfs(u) và gán cho

các đỉnh i được duyệt tới lt[i]= k (thay vs[] bởi lt[]); quay

lại bước 1;

Bước 3: Xuất k và lt[u] với mọi đỉnh u;

-Cài đặt DFS:

*// G cho bởi ma trận kề a[i][j]*

int a[100][100], lt[100], n;

int tpltDfs()

{

    int u, int k = 0;

    for (u = 1; u <= n; u++)

        lt[u] = 0;

    for (u = 1; u <= n; u++)

    if (lt[u] == 0)

    {

        k++;

        DfsDequy(u);

    }

    return (k);

}

-Cài đặt BFS:

*// G cho bởi ma trận kề a[i][j]*

int a[100][100], lt[100], n;

int tpltBfs()

{

    int u, int k = 0;

    for (u = 1; u <= n; u++)

        lt[u] = 0;

    for (u = 1; u <= n; u++)

    if (lt[u] == 0)

    {

        k++;

        BfsDequy(u);

    }

    return (k);

}

7. Đỉnh trụ

- Thuật toán :

Bước 1: Tìm số k thành phần liên thông của G;

Bước 2: Xét mọi đỉnh u ∈ G:

    2.1:Bỏ u và các cạnh liên thuộc u và tính số

        thành phần liên thông l của đồ thị G\{u};

    2.2: Nếu l > k thì ghi nhận u là đỉnh trụ;

    2.3: Trả lại u và các cạnh liên thuộc u;

Bước 3: Xuất danh sách các đỉnh trụ

-Cài đặt:

8. Cạnh cầu:

- Thuật toán :

Bước 1: Tìm số k thành phần liên thông của G;

Bước 2: Xét mọi cạnh e ∈ G:

    2.1: Bỏ cạnh e = (u,v) (các đỉnh u và v vẫn giữ

        lại) và tính số thành phần liên thông l của đồ thị G\{e};

    2.2: Nếu l > k thì ghi nhận e là cạnh cầu;

    2.3: Trả lại e ;

Bước 3: Xuất danh sách các cạnh cầu

-Cài đặt:

9. TÍnh liên thông của đồ thị có hướng :

- Thuật toán :

Bước khởi tạo: i = 1;

Bước 1: Thực hiện Dfs(i)/Bfs(i) và Tính số k các đỉnh đã duyệt;

Bước 2: Nếu k < n chuyển bước 4; nếu k = n thì chuyển bước 3;

Bước 3: Nếu i=n xuất 1, nếu i< n thì i= i+1 và quay lại bước 1;

Bước 4: Thực hiện Dfs(1)/Bfs(1) khi coi các cạnh của đồ thị là

vô hướng và Tính số k các đỉnh đã duyệt;

Bước 5: Nếu k < n xuất 0; nếu k = n thì xuất 2;

10. Tìm các thành phần liên thông mạnh của đồ thị có hướng:

- Thuật toán:

Bước khởi tạo: k= 0; lt[u]= 0 với mọi đỉnh u;

Bước 1: Thực hiện DFS(u)/BFS(u) với mọi u ∈ G và đánh dấu

vs[u][v]=1 với mỗi v được duyệt đến;

Bước 2: Nếu mọi đỉnh u đều có lt[u] > 0 thì chuyển bước 4,

ngược lại chọn đỉnh u có lt[u] = 0;

Bước 3: k= k + 1, lt[u]= k và xét tất cả các đỉnh v có vs[u][v] = 1,

vs[v][u]= 1 và lt[v]= 0 thì lt[v]= k; quay lại bước 2;

Bước 4: Xuất k và lt[u] với mọi đỉnh u;

-Cài đặt : Kosaraju

11. Chu trình và đường đi Euler

- Vô hướng :

+ Chu trình đơn chứa tất cả các cạnh của G

được gọi là chu trình Euler.

 => G là đồ thị Euler <=> G chứa chu trình Euler.

+ Đường đi Euler trong G là đường đi đơn chứa

tất cả các cạnh của G.

 => G là đồ thị nửa Euler <=> G chứa đường đi

Euler.

+ Tính chất :

\* Đồ thị vô hướng G là đồ thị Euler <=> G liên thông và

all đỉnh có bậc chẵn

\* Đồ thị vô hướng G là đồ thị nửa Euler <=> G liên thông

và [deg(u) % 2 == 1] <= 2.

+ Thuật toán Tìm đường đi / chu trình Euler :

Bước 1(Kiểm tra điều kiện): Nếu G không thỏa mãn điều kiện

thì kt= 0, nếu có chu trình Euler thì kt= 1; nếu có đường đi Euler

thì kt= 2;

Bước 2:

(2.1) Nếu kt= 0 => thông báo đồ thị không có chu trình/đường đi

Euler và dừng;

(2.2) Nếu kt= 1 => chọn u là đỉnh cho trước và

chuyển sang bước 3;

(2.3) Nếu kt= 2 => chọn u là đỉnh bậc lẻ có số hiệu nhỏ nhất và

chuyển sang bước 3;

Bước 3 (Xây dựng chu trình/đường đi Euler bắt đầu từ

đỉnh u):

    (3.1) Tạo mảng CE để ghi chu trình/ đường đi Euler và Stack

          để xếp các đỉnh sẽ xét. Xếp đỉnh u vào Stack;

    (3.2) Xét đỉnh v nằm trên cùng của Stack và thực hiện:

            - Nếu v là đỉnh cô lập thì lấy v ra khỏi Stack và đưa vào CE.

            - Nếu v có đỉnh kề là x thì đưa x vào Stack sau đó xóa

              cạnh nối v với x;

    (3.3) Quay lại (3.2) cho tới khi stack rỗng;

Bước 4: Xuất chu trình/đường đi Euler chứa trong CE

theo thứ tự ngược lại.

-Cài đặt thuật toán :

int a[100][100], n, u, vs[100], e[100]. **s**[10000], ce[10000];

int kt(int a[][], int n) {

    int v;

    for (v= 1; v<= n; v++) {

        vs[v]= 0; e[v]= 0;

    }

    DfsDequy(1); int ok= 1;

    for (v= 1; v<= n; v++){

        if (vs[v] == 0){

            ok= 0; break;

        }

    }

    if (ok == 0) return(0);

    int bl= 0;

    for (v= 1; v<= n; v++) {

        int deg= 0;

        for (int i= 1; i<= n; i++) {

            if (a[v][i] == 1) deg++;

            if (deg%2 == 1) {

                bl++;

            }

            if (bl> 2) return(0);

            if (bl == 1){

                u= v; ok= 2;

            }

            return(ok);

        }

    }

}

void ceu(int a[][], int n, int u) {

    int top=0, v;

    top++; s[top]= u; k= 0;

    while (top > 0) {

        int v= s[top];

        int ok= 1;

        for (int x= 1; x<= n; x++){

            if (a[v][x] ==1){

                top++; s[top]= x;

                ok= 0;

                a[v][x]= 0;

                a[x][v]= 0;

                break;

            }

        }

        if (ok== 1) {

            k++, ce[k]= v; top--;

        }

    }

    for (v= k; v> 0; v--)

        cout << ce[v] << “ “;

}

- Có hướng :

+ Tính chất

- Đồ thị có hướng G là đồ thị Euler <=> G liên thông yếu

và mọi đỉnh v thuộc V có deg-(v) = deg+(v).

- Đồ thị có hướng G là đồ thị nửa Euler <=> G liên thông

yếu và số đỉnh v thuộc V có bán bậc-vào và bán bậc-ra

chênh lệch nhau 1 đon vị <= 2.

+ Thuật toán :

Bước 1 (Kiểm tra điều kiện):

- Nếu G không thỏa mãn điều kiện thì kt= 0,

- nếu có chu trình Euler thì kt= 1;

- nếu có đường đi Euler thì kt= 2;

Bước 2:

    (2.1) Nếu kt= 0 ⇒ thông báo đồ thị không có chu trình/đường đi

        Euler và dừng;

    (2.2) Nếu kt= 1 ⇒ chọn u là đỉnh cho trước và chuyển sang

        bước 3;

    (2.3) Nếu kt= 2 ⇒ chọn u là đỉnh có bán bậc ra nhiều hơn bán

    bậc vào và chuyển sang bước 3

Bước 3 (Xây dựng chu trình/đường đi Euler bắt đầu từ đỉnh u):

    (3.1) Tạo mảng CE để ghi chu trình/ đường đi Euler và Stack

        để xếp các đỉnh sẽ xét. Xếp đỉnh u vào Stack;

    (3.2) Xét đỉnh v nằm trên cùng của Stack và thực hiện:

        - Nếu v là đỉnh cô lập thì lấy v ra khỏi Stack và đưa vào CE.

        - Nếu v có đỉnh kề là x thì đưa x vào Stack sau đó xóa

        cạnh nối v với x;

    (3.3) Quay lại (3.2) cho tới khi stack rỗng;

Bước 4: Xuất chu trình/đường đi Euler chứa trong CE

theo thứ tự ngược lại.

-Cài đặt :

int a[100][100], n, u, vs[100], s[10000], ce[10000];

int kt(int a[][], int n) {

    int v, top= 0, dem= 0;

    top++; s[top]= 1; dem++;

    while (top> 0){

        int v= s[top], ok= 1;;

        for (int i= 1; i<= n; i++){

            if ((a[v][i] == 1 || a[i][v]== 1) && vs[i]== 0){

                top++; s[top]= i;

                vs[i] =1;

                dem++;

                ok= 0;

                break;

            }

            if (ok == 1) top--;

        }

    }

    if (dem< n) return(0);

    int x= 0, y= 0;

    for (v= 1; v<= n; v++) {

        int d1= 0, d2= 0;

        for (int i= 1; i<= n; i++){

            if (a[i][v] == 1) d1++;

            if (a[v][i] == 1) d2++;

        }

        if (d1!= d2){

            if (abs(d1-d2) > 1) {

                return(0);

            }

            else{

                if (x> 0 && y> 0)

                return(0);

            }

        }

        else {

            if (d2> d1) x= v;

            else y= v;

        }

        if (x== 0 && y== 0) {

            return(1);

        }

        else{

            u= x; return(2);

        }

    }

}

void ceu(int a[][], int n, int u) {

    int top=0, v;

    top++;

    s[top]= u; k= 0;

    while (top > 0) {

        int v= s[top];

        int ok= 1;

        for (int x= 1; x<= n; x++){

            if (a[v][x] ==1){

                top++;

                s[top]= x;

                ok= 1;

                a[v][x]= 0; break;

            }

        }

        if (ok== 1) {

            k++;

            ce[k]= v;

            top--;

        }

    }

    for (v= k; v> 0; v--) cout << ce[v] << “ “;

}

12. Chu trình và đường đi hamilton

- Chu trình đơn đi qua tất cả các đỉnh của G, mỗi

đỉnh 1 lần gọi là chu trình Hamilton

 => G là đồ thị Hamilton <=> G chứa chu trình Hamilton

- Đường đi Hamilton trong G là đường đi đi qua tất

cả các đỉnh của G, mỗi đỉnh 1 lần

 => G là đồ thị nửa Hamilton <=> G chứa đường đi Hamilton

- Thuật toán sử dụng đệ quy:

Hamilton(k){

    for v thuộc Ke(x[k-1]){

        if (k= n+1) {

            if (v= x[1])

                GhiNhan(x[1], …, x[n]);

        }

        else {

            if (vs[v]= 0) {

                x[k]= v; vs[v]= 1;

                Hamilton(k+1);

                vs[v]= 0;

            }

        }

    }

}

- Liệt kê các chu trình hamilton bắt đầu tại u:

LietKeHamilton(u){

    for (v thuộc G)

        vs[v]= 0;

    x[1]= u;

    vs[u]= 1;

    Hamilton(2);

}

13. Tìm đường đi ngắn nhất

- DIJKSTRA

+ Thuật toán :

Khởi tạo: d[v]= a[s][v]; e[v]= s; vs[v]= 0;

(1) Bắt đầu tìm kiếm từ s: d[s]= 0; e[s]= 0; vs[s]= 1;

(2) Tìm đỉnh u sao cho d[u]= min{d[i] | vs[i] = 0};

Nếu không tìm được thì chuyển sang (5). Nếu tìm được thì

sang (3).

(3) Đặt vs[u]= 1;

(4) Đối với tất cả v thuộc G thỏa mãn

(vs[v]= 0) & (d[v]> d[u] + a[u][v]) thì thay thế:

e[v]= u; d[v]= d[u] + a[u][v];

và quay lại (2).

(5) Xuất d[v] và đường đi từ s đến v

- Cài đặt :

int n, s, a[100][100], d[100], e[100], vs[100];

void Dijkstra(int s){

    int u, v;

    for (v=1; v<= n; v++){

        d[v]= a[s][v];

        e[v]=s;

    }

    d[s]= 0; e[s] = 0; vs[s]= 1;

    while (1){

        int u= 0, min= 32767;

        for (v=1; v<= n; v++) {

            if (vs[v]==0 && d[v]< min){

                u= v; min= d[v];

            }

        }

        if (u== 0) return;

        vs[u]= 1;

        for (v=1; v<=n; v++){

            if (vs[v]== 0 && d[v]> d[u]+a[u][v]) {

                d[v]= d[u] + a[u][v];

                e[v] = u;

            }

        }

    }

}

- BellmanFord:

+ Thuật toán :

Khởi tạo: d[v]= a[s][v]; e[v]= s; vs[v]= 0;

(1) Bắt đầu tìm kiếm từ s: d[s]= 0; e[s]= 0; vs[s]= 1;

(2) Tìm đỉnh u sao cho d[u]= min{d[v] | vs[v] = 0}. Nếu không tìm

được thì chuyển sang (5). Nếu tìm được thì sang (3).

(3) Đặt vs[u]= 1. Nếu u= t thì chuyển sang (5); ngược lại chuyển

sang (4);

(4) Đối với tất cả v ∈ G thỏa mãn

(vs[v]= 0) & (d[v]> d[u] + a[u][v]) thì thay thế:

e[v]= u; d[v]= d[u] + a[u][v];

và quay lại (2);

(5) Nếu d[t] < max thì xuất d[t] và đường đi từ s đến t; nếu

ngược lại xuất không có đường đi từ s đến t.

+ Cài đặt :

int n, s, a[100][100], d[100], e[100];

int BellmanFord(int s){

    int dem, u, v;

    for (v=1; v<= n; v++){

        d[v]= a[s][v]; e[v]=s;

    }

    d[s]= 0; e[s] = 0; int ok= 0;

    for (dem=1; dem<= n-1; dem++){

        int ok= 1;

        for (v=1; v<=n; v++){

            for (u=1; u<=n; u++){

                if (d[v] > d[u] + a[u][v]) {

                    d[v] = d[u] + a[u][v];

                    e[v] = u;

                    ok= 0;

                }

            }

        }

        if (ok== 1) return(1);

    }

    return(0);

}

- Floyd :

+ Thuật toán :

- Khởi tạo: d[i][j]= a[i][j]; e[i][j]= i;

- Với mọi k thuộc G, i thuộc G, j thuộc G sao cho

    Nếu (d[i][j]> d[i][k] + d[k][j]) thì thay thế:

        e[i][j]= k; d[i][j]= d[i][k] + d[k][j];

- Xuất kết quả:

+ Nếu có đỉnh u mà d[u][u] < 0 thì xuất thông báo

G chứa chu trình âm;

+ Ngược lại xuất d[i][j] và e[i][j].

+ Cài đặt :

int n, a[100][100], d[100][100], e[100][100];

int Floyd(){

    int i, j, k;

    for (i=1;i<= n; i++){

        for (j=1; j<= n; j++){

            d[i][j]= a[i][j];

            e[i][j]= i;

        }

    }

    for (k=1; k<= n; k++){

        for (i=1; i<=n; i++){

            for (j=1; j<=n; j++){

                if (d[i][j] > d[i][k] + d[k][j]) {

                    d[i][j] = d[i][k] + d[k][j];

                    e[i][j] = k;

                }

            }

        }

    }

    for (i= 1; i<= n; i++) {

        if (d[i][i] < 0) return(0);

    }

    return(1);

}

14. Cây khung , cây khung cực tiểu

+ Thuật toán Theo DFS:

◼ Thuật toán TreeDfs(u):

Bước 1: Thực hiện DFS(u);

Bước 2 (Xuất kết quả):

- Nếu số đỉnh được duyệt bằng n thì xuất kết quả T;

- Nếu số đỉnh được duyệt nhỏ hơn n thì xuất thông

báo: “Không có cây khung”;

+ Cài đặt :

*// G cho bởi ma trận kề a[i][j]*

int a[100][100], n, u, vs[100], e[100];

void DfsDequy(int u) {

    int v;

    vs[u]= 1;

    for (v= 1; v<=n; v++){

        if (vs[v]==0 && a[u][v]==1){

            e[v]= u;

            DfsDequy(v);

        }

    }

}

void TreeDfs(int u) {

    int v;

    for (v= 1; v<=n; v++) vs[v] = 0;

    DfsDequy(u);

    int dem= 0;

    for (v= 1; v<=n; v++) {

        if (vs[v] ==1) dem++;

    }

    if (dem == n) {

        for (v= 1; v<=n; v++) {

            if (e[v] != 0) cout << v << ” ” << e[v] << "\n";

        }

    }

    else{

        cout << ”Khong co Cay khung”;

    }

}

+ Thuật toán theo BFS ;

◼ Thuật toán TreeBfs(u)

Bước 1: Thực hiện BFS(u);

Bước 2 (Xuất kết quả):

- Nếu số đỉnh được duyệt bằng n thì xuất kết quả T;

- Nếu số đỉnh được duyệt nhỏ hơn n thì xuất thông

báo: “Không có cây khung”;

+ Cài đặt :

*// G cho bởi ma trận kề a[i][j]*

int a[100][100], n, u, vs[100], e[100], q[100];

void Bfs(int u) {

    int v, dq= 1, cq= 0;

    vs[u]= 1; e[u]= 0; cq++; q[cq]= u;

    while (dq <= cq){

        int i = q[dq]; dq++;

        for (v= 1; v<=n; v++){

            if (vs[v]==0 && a[i][v]==1){

                e[v]= u; vs[v]= 1; cq++; q[cq]= v;

            }

        }

    }

}

void TreeBfs(int u) {

    int v;

    for (v= 1; v<=n; v++) vs[v] = 0;

    Bfs(u);

    int dem= 0;

    for (v= 1; v<=n; v++) if (vs[v] ==1) dem++;

    if (dem == n) {

        for (v= 1; v<=n; v++) {

            if (e[v] != 0) cout << v << ” ” << e[v] << endl;

        }

    }

    else{

        cout << ”Khong co Cay khung”;

    }

}

- Cây Khung cực tiểu:

+ Thuật toán :

Khởi tạo: d[v] = a[s][v]; e[v] = s; vs[s] = 1; e[s] = 0;

➢ Tại mỗi bước tìm u sao cho d[u] = Min{d[v] | vs[v] = 0}

➢ Nếu tìm được u cần cập nhật d[v]:

nếu vs[v] = 0 và d[v] > a[u][v] thì thay thế:

d[v] = a[u][v]; e[v] = u;

➢ Nếu không tìm được u thì thông báo “Khong co cay khung”.

+ Cài đặt :

int n, a[100][100];

int vs[100], d[100], e[100];

void Prim(int s) {

    for (int v= 1; v<= n, v++) {

        vs[v]= 0; d[v]= a[s][v]; e[v]= s;

    }

    vs[s]= 1; d[s]= 0; e[s]= 0;

    int wt= 0, dem = 1;

    while (dem < n) {

        int u = 0;

        int min = 30000;

        for (v = 1; v<= n; v++)

        if (vs[v]==0 && d[v] < min) {

            min= d[v]; u= v;

        }

    }

    if (u==0) {

        cout << “Khong co cay khung”;

        return;

    }

    vs[u]= 1; wt= wt + a[u][e[u]];

    for (v= 1; v<= n; v++){

        if (vs[v]==0 && d[v] > a[u][v])

        {

            d[v]= a[u][v];

            e[v]= u;

        }

    }

    cout << wt << "\n";

    for (v= 1; v<= n; v++){

        if (e[v] !=0)

        cout << v << “ “ <<e[v] << "\n";

    }

    return;

}

- Kruskal:

+ Thuật toán :

Khởi tạo :

Sắp xếp các cạnh theo thứ tự tăng của trọng số e1, ..., em;

    T = rỗng, WT = 0, k = 0;

    for(int i = 1; i <= m; i++){

        if(T v {ei} không chứa chu trình){

            T = T v {ei};

            WT = WT + trọng số của ei;

            k++;

            if(k == n - 1) return {T và WT};

        }

    }

    return (G không có cây khung);

+ Cài đặt:

…

15. Bài toán luồng cực đại trong mạng

- Ford-furkerson:

Thuật toán Max\_Flow {

    for u ∈V :

        for v thuộc V :

            f(u, v) = 0;

    Stop = 0;

    while (!Stop) {

        <Xác định đồ thị tăng luồng Gf >;

        if (Tìm được đường tăng luồng P bằng Bfs(s)) {

            <Tìm δ là trọng số nhỏ nhất trên P>;

            <Tăng luồng f theo P>;

        }

        else Stop = 1;

    }

    return (f, val(f));

}

+ Cài đặt:

int Stop = 0;

int q[100];

int d[100];

int vs[100];

int e[100];

int fl[100][100];

void FindPath()

{

    int cq, dq, u, v;

    for (u = 1; u <= n; u++)

        vs[u] = 0;

    cq = 1;

    dq = 1;

    q[cq] = s;

    vs[s] = 1;

    e[s] = 0;

    d[s] = 10000;

    while (dq <= cq)

    {

        u = q[dq];

        dq++;

        for (v = 1; v <= n; v++)

        if (vs[v] == 0)

        {

            if (c[u][v] > 0 && fl[u][v] < c[u][v])

            {

                e[v] = u;

                d[v] = (d[u] < c[u][v] - fl[u][v]) ? d[u] : c[u][v] - fl[u][v];

                cq++;

                q[cq] = v;

                vs[v] = 1;

                if (v == t)

                    return;

            }

            if (c[v][u] > 0 && fl[v][u] > 0)

            {

                e[v] = -u;

                d[v] = (d[u] < fl[v][u]) ? d[u] : fl[v][u];

                cq++;

                q[cq] = v;

                vs[v] = 1;

                if (v == t)

                    return;

            }

        }

    }

    Stop = 1;

}