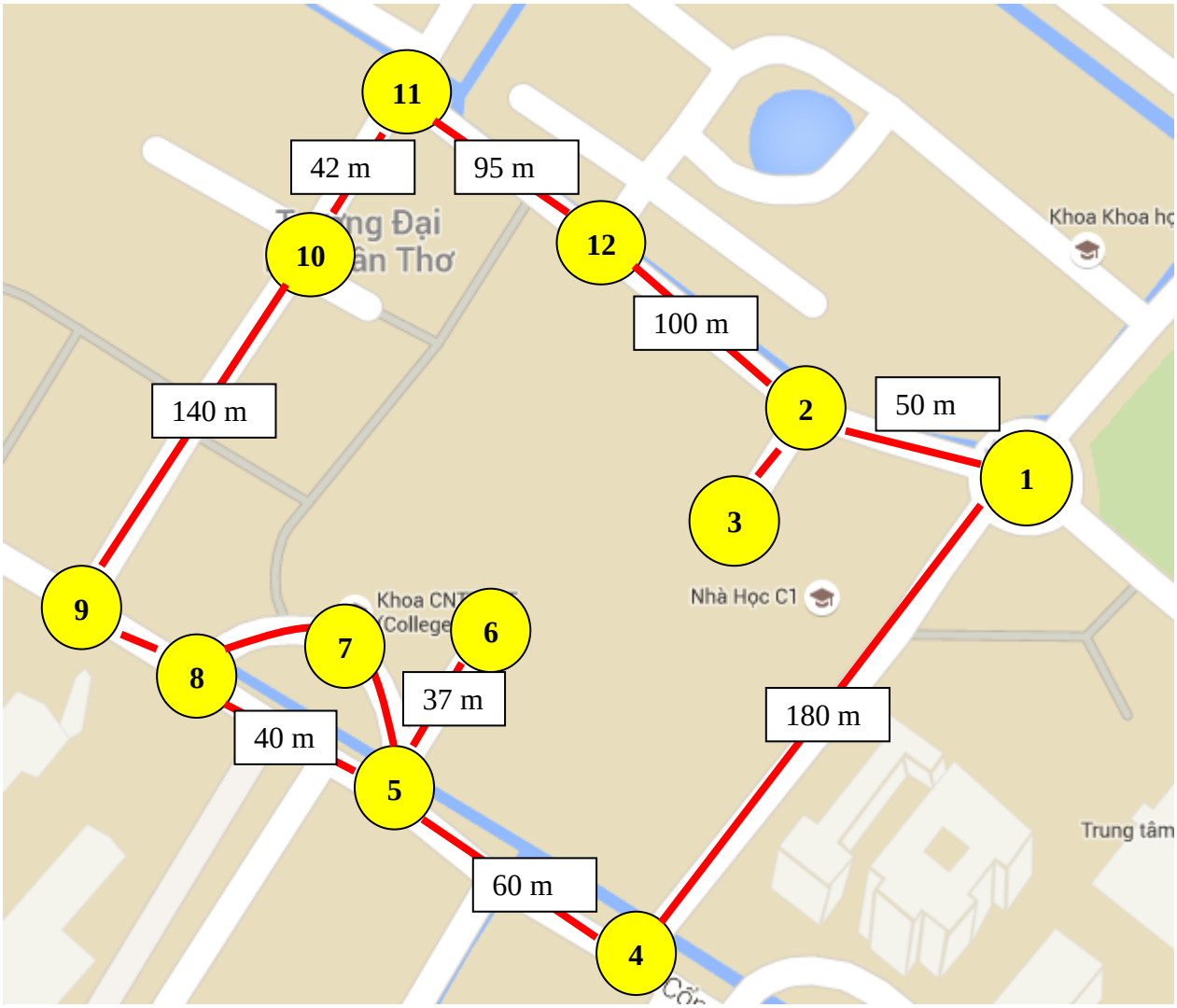
Buổi 3
Mục tiêu

- Biểu diễn đồ thị có trọng số
- Cài đặt giải thuật Moore - Dijkstra tìm đường đi ngắn nhất từ một đỉnh đến các đỉnh khác
- Cài đặt giải thuật Bellman – Ford
- Cài đặt giải thuật Floyd

Yêu cầu:

- Biểu diễn đồ thị
- Các phép toán cơ bản trên đồ thị

1. Đồ thị có trọng số



Xét bản đồ khu 2 ĐHCT như hình trên. Giả sử ta muốn tìm đường đi ngắn nhất từ giao điểm 1 đến giao điểm 8.
Ta mô hình hoá bản đồ về đồ thị với đỉnh là các giao điểm và cung là các con đường. Ta thu được đồ thị G = <V, E>.
Rõ ràng là để có thể biết được chiều dài của đường đi ta cần phải biết chiều của các con đường và phải biểu diễn chúng trong đồ thị.

Vì thế, với mỗi cung của đồ thị (ứng với một con đường) ta gán cho nó 1 con số. Con số này được gọi là chiều dài của cung hay tổng quát hơn là trọng số của cung. Một số đồ thị và các cung được gán trọng số gọi là đồ thị có trọng số.

Biểu diễn đồ thị có trọng số

Xét một đồ thị G = <V, E>, với mỗi cung e = (u, v) của G được gán một trọng số l(e).
Ta có thể mở rộng phương pháp Ma trận kề (đỉnh – đỉnh) để biểu diễn đồ thị có trọng số.

- Ma trận có số hàng và số cột bằng với số đỉnh
- Phần tử A[i][j] chứa trọng số (hay chiều dài) của cung (i, j)
- Nếu không có cung (i, j) phần tử A[i][j] chứa giá trị 0 hoặc một giá trị đặc biệt nào đó khác với trọng số của các cung, gọi giá trị này là NO_EDGE.

Chú ý:

- Phương pháp này chỉ sử dụng được với đồ thị có hướng và vô hướng
- Không sử dụng được với đồ thị có đa cung (vì chỉ có 1 ô A[i][j] duy nhất nên chỉ chứa được 1 giá trị)

Ví dụ:

```
#define MAXN 1000
#define NO_EDGE 0
//hoac gia tri dac biet nao do

typedef struct {
    int n;
    int L[MAXN][MAXN];
} Graph;
```

Khởi tạo đồ thị ta cho tất cả phần tử của L đều bằng NO_EDGE: không có cung nào.

```
void init_graph(Graph* G, int n) {
    G->n = n;
    int i, j;
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            G->L[i][j] = NO_EDGE;
}
```

Thêm cung (u, v) có trọng số (chiều dài) w vào đồ thị có thể thực hiện dễ dàng bằng phép gán:

```
…
        G.L[u][v] = w;
…
```

Hãy sử dụng lệnh này trong lúc đọc đồ thị từ tập tin.

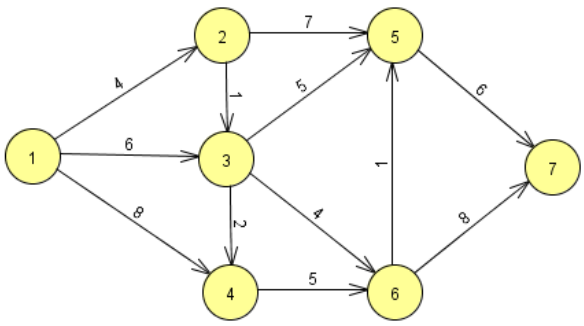
Lưu trữ đồ thị có trọng số trong tập tin

Tương tự như đồ thị không có trọng số, ta có thể lưu trữ đồ thị trong tập tin bằng cách thêm trọng số của cung kế bên cạnh cung. Định dạng của tập tin như sau:

```
m n
u1 v1 w1
u2 v2 w2
…
um vm w
```

Trong đó, n là số đỉnh, m là số cung. m dòng kế tiếp lưu thông tin của m cung, mỗi cung lưu đỉnh đầu, đỉnh cuối và trọng số của cung.

Ví dụ:




```
7 12
1 2 4
1 3 6
1 4 8
2 3 1
2 5 5
3 4 2
3 5 5
3 6 4
4 6 5
5 7 6
6 5 1
6 7 8
```

2. Bài toán tìm đường đi ngắn nhất

Cho đồ thị G = <V, E>, tìm đường đi ngắn nhất từ đỉnh u đến đỉnh v.

Đường đi ngắn nhất từ u đến v chỉ tồn tại nếu như trên đường đi không chứa chu trình âm.

Giải thuật Moore - Dijkstra tìm đường đi ngắn nhất từ s đến các đỉnh còn lại

- Ý tưởng: khởi tạo đường đi ngắn nhất trực tiếp từ s đến các đỉnh còn lại. Sau đó lần lượt cập nhật lại đường đi nếu đường đi mới tốt hơn đường đi cũ.

Các biến hỗ trợ:

- pi[i]: chiều dài đường đi ngắn nhất từ s đến i (tính đến thời điểm đang xét).
- p[i]: đỉnh liền trước đỉnh i trên đường đi ngắn nhất từ s đến i (tính đến thời điểm đang xét).
- L[i][j]: chiều dài (trọng số) của cung (i, j)
- mark[i]: cho biết đỉnh i đã được đánh dấu chưa.

Giải thuật:

Khởi tạo:

- pi[i] = oo với mọi i != s;
- pi[s] = 0;
- mark[i] = 0 với mọi i

Lặp (n-1 lần)

- Chọn đỉnh chưa đánh dấu (mark[i] == 0) có chiều dài của đường đi từ s đến nó (pi[i]) nhỏ nhất => i.
- Đánh đấu đã xét i bằng cách đặt mark[i] = 1
- Xem xét cập nhật pi[j] và p[j] các đỉnh kề của i chưa được xét (mark[j] == 0)
    - Một đỉnh sẽ được cập nhật nếu đường đi mới (thông qua i) tốt hơn đường đi cũ

```
if (pi[i] + L[i][j] < pi[j]) {
        pi[j] = pi[i] + L[i][j]
        p[j] = i
}
```

Cài đặt giải thuật:

```
#define INFINITY 9999999

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; //trước đỉnh s không có đỉnh nào cả

    // lặp n hoặc n-1 lần đều được
    for (it = 1; it < G->n; it++) {
        //1. Tìm i có mark[i] == 0 va có pi[i] nhỏ nhất
        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;
            }
        //Đánh dấu i đã xét
        mark[i] = 1;
        //2. Cập nhật pi và p của các đỉnh kề của i (nếu thoả)
        for (j = 1; j <= G->n; j++)
            if (G->L[i][j] != NO_EDGE && mark[j] == 0) {
                if (pi[i] + G->L[i][j] < pi[j]) {
                    pi[j] = pi[i] + G->L[i][j];
                    p[j] = i;
                }
            }

    }
}
```

Sau khi gọi hàm Dijkstra kết quả sẽ được lưu trong 2 mảng pi[] và p[]. Ta có thể in kết quả này ra , màn hình để kiểm tra:

```
for (i = 1; i <= G.n; i++)
   printf("pi[%d] = %d, p[%d] = %d\n", i, pi[i], i, p[i]);
```

Dựa vào kết quả này, ta có thể trả lời các câu hỏi thuộc 2 dạng sau:

Hỏi: "Chiều dài đường đi ngắn nhất từ s đến u là bao nhiêu ?"
Trả lời: pi[u]

Hỏi: "Đường đi ngắn nhất từ s đến u là đường nào ?"
Trả lời: ta có thể sử dụng đoạn chương trình bên dưới để in đường đi từ s đến u:

```
int path[MAXN]; // lưu các đỉnh trên đường đi
int k = 0;      // số đỉnh của đường đi
int current = u;

while (current != -1) {
     path[k] = current; k++;
     current = p[current];
}
int i;
for (i = k-1; i >=0; i--)
     printf("%d ", path[i]);
```

Bài tập 1. Viết chương trình đọc một đồ thị **có hướng** từ tập tin, tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại. In các thông tin pi[i] và p[i] của các đỉnh ra màn hình.

Bài tập 2. Viết chương trình đọc một đồ thị **vô hướng** từ tập tin, tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại. In các thông tin pi[i] và p[i] của các đỉnh ra màn hình. Chú ý: đối với đồ thị vô hướng khi thêm cung (u, v) vào đồ thị ta thêm luôn cung (v, u) vào đồ thị.

Bài tập 3 (ứng dụng) – Mê cung số (number maze)
Cho một mê cung số được biểu diễn bằng một mảng 2 chiều chứa các con số từ 0 đến 9 (xem hình bên dưới). Một con robot được đặt tại góc trên bên trái của mê cung và muốn đi đến góc dưới bên phải của mê cung. Con robot có thể đi lên, xuống, qua trái và qua phải 1 ô. Chi phí để đi đến một ô bằng với con số bên trong ô đó.

0	3	1	2	9
7	3	4	9	9
1	7	5	5	3
2	3	4	2	5

Hãy tìm cách giúp con robot đi đến ô ở góc dưới phải sao cho tổng chi phí thấp nhất.
Đường đi có chi phí thấp nhất cho ví dụ này bằng 24.

Dữ liệu đầu vào được cho trong một tập có định dạng như sau:
- Dòng dầu chứa 2 số nguyên M N (M: số hàng, N: số cột)
- M dòng tiếp theo mô tả các số trong mê cung

Ví dụ trên sẽ được lưu như sau:

```
4 5
0 3 1 2 9
7 3 4 9 9
1 7 5 5 3
2 3 4 2 5
```

Dữ liệu đầu ra: in chi phí thấp nhất để con robot đi từ góc trên bên trái về góc dưới bên phải.
Ví dụ trên, cần in ra màn hình:

```
24
```

Gợi ý giải:

- Mô hình hoá bài toán về đồ thị có hướng
  - Đỉnh ⇔ ô
  - Cung ⇔ hai ô cạnh nhau
  - Trọng số cung (u, v) ⇔ giá trị của ô tương ứng với đỉnh v.

Cách 1 - Đánh số ô:
Giả sử các ô trong mê cung được đánh chỉ số từ (0, 0): góc trên trái đến (M-1, N-1): góc dưới phải.

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20

- Ô (i,j) sẽ tương ứng với đỉnh (i*N + j) + 1
- Đỉnh u sẽ tương ứng với ô ở hàng (u-1)/N và cột (u-1)%N

Liệt kê các đỉnh kề của 1 đỉnh:
1 ô có thể kề với 4 ô xung quanh nó nên 1 đỉnh sẽ có nhiều nhất 4 đỉnh kề tương ứng. Ta sẽ dùng công thức biến đổi ô -> đỉnh và đỉnh -> bên trên để tìm đỉnh kề của 1 đỉnh.

Giả sử ta muốn tìm đỉnh kề của đỉnh u, ta sẽ làm như sau:

- Đổi u thành hàng i = (u – 1)/N và cột j = (u - 1)%N
- Tìm 4 ô xung quanh ô (i, j) là (i-1, j), (i+1, j), (i, j -1) và (i, j + 1)
- Đổi 4 ô thành 4 đỉnh (nếu ô vẫn nằm trong phạm vi (0,0) và (M-1, N-1).

Sử dụng khung chương trình bên dưới để cặp nhật pi và p của các đỉnh kề của đỉnh u trong giải thuật Dijkstra.

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

//Đổi đỉnh u thành ô (i, j)
int i = (u – 1)/N;
int j = (u – 1)%N;
//Duyệt qua các ô kề của ô (i, j)
for (k = 0; k < 4; k++) {
    ii = i + di[k];
    jj = j + dj[k];
    //Kiểm tra ô (ii,jj) có nằm trong mê cung không
    if (ii >= 0 && ii < M && jj >= 0 && jj < N) {
         v = ii*N + jj; //đổi ô (ii,jj) thành đỉnh v
         //v là đỉnh kề của đỉnh u
         …

    }
```

Cách 2 – sử dụng chỉ số hàng và côt của ô làm chỉ số đỉnh

- Không cần đổi ô thành đỉnh
- Sử dụng mảng pi[i][j] (2 chiều) thay vì pi[u] (1 chiều)

3. Giải thuật Bellman – Ford

Tương tự giải thuật Dijkstra, cho phép tìm đường đi ngắn nhất từ 1 đỉnh đến các đỉnh khác.

Ý tưởng:

- Khởi tạo: giống Dijkstra
  - pi[i] = oo với mọi i != s;
  - pi[s] = 0, p[s] = -1;
- Lặp n – 1 lần
  - Duyệt qua tất cả các cung (u, v) và cập nhật pi[v] và p[v] nếu thoả điều kiện

```
if (pi[u] + L[u][v] < pi[v]) {
     pi[v] = pi[u] + L[u][v];
     p[v] = u;
}
```

Về cơ bản, giải thuật Bellman – Ford có cùng nguyên lý như giải thuật Dijkstra. Điểm khác biệt là ở mỗi lần lặp: giải thuật Dijkstra chọn ra đỉnh có pi[u] bé nhất và cập nhật các đỉnh kề của nó; trong khi đó giải thuật Bellman – Ford ***duyệt qua kết tất cả các cung (u, v)*** và cập nhật các đỉnh v.

Vì thế để thuận tiện cho giải thuật Bellman – Ford ta phải biểu diễn đồ thị sao cho duyệt qua các cung của nó dễ dàng. Cách đơn giản nhất là lưu ***danh sách các cung*** của đồ thị.

```
typedef struct {
     int u, v; // đỉnh đầu v, đỉnh cuối v
     int w;    // trọng số w
} Edge;

typedef struct {
     int n, m;          // n: đỉnh, m: cung
     Edge edges[1000]; // lưu các cung của đồ thị
} Graph;
```

Khởi tạo đồ thị:

```
void init_graph(Graph* G, int n) {
     G->n = n;
     G->m = 0;
}
```

Thêm 1 cung vào đồ thị:

```
void add_edge(Graph* G, int u, int v, int w) {
     G->edges[G->m].u = u;
     G->edges[G->m].v = v;
     G->edges[G->m].w = w;
     G->m++;
}
```

Giải thuật Bellman – Ford:

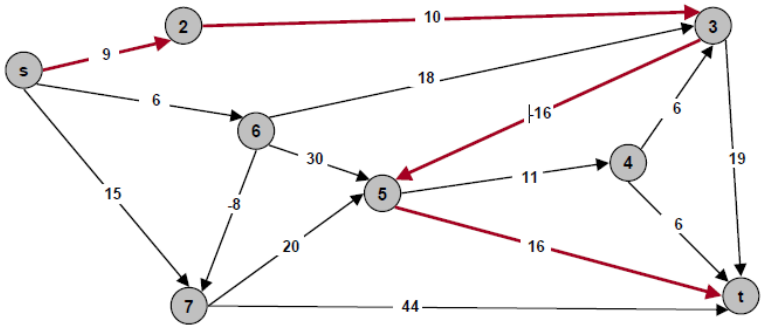
```
#define INFINITY 9999999

int pi[MAXN];
int p[MAXN];

void BellmanFord(Graph* G, int s) {
    int i, j, it;
    for (i = 1; i <= G->n; i++) {
        pi[i] = INFINITY;
    }
    pi[s] = 0;
    p[s] = -1; //trước đỉnh s không có đỉnh nào cả

    // lặp n hoặc n-1 lần đều được
    for (it = 1; it < G->n; it++) {
        // Duyệt qua các cung và cập nhật (nếu thoả)
        for (k = 0; k < G->m; k++) {
            int u = G->edges[k].u;
            int v = G->edges[k].v;
            int w = G->edges[k].w;
            if (pi[u] + w < pi[v]) {
                pi[v] = pi[u] + w;
                p[v] = u;
            }
        }
    }
}
```

Kết quả giải thuật Bellman – Ford sử dụng giống giải thuật Dijkstra.

Phát hiện chu trình âm:
Giải thuật Bellman – Ford hơn giải thuật Dijkstra ở chỗ có thể chạy được với đồ thị có trọng số âm. Sau khi chạy xong giải thuật, ta có thể phát hiện được chu trình âm bằng cách duyệt qua các cung một lần nữa nếu tiếp tục cải tiến được pi[v] thì có nghĩa là có chu trình âm.

```
// Duyệt qua các cung một lần nữa
for (k = 0; k < G->m; k++) {
     int u = G->edges[k].u;
     int v = G->edges[k].v;
     int w = G->edges[k].w;
     if (pi[u] + w < pi[v]) {
          // Có chu trình âm
     }
}
```

Bài tập 4. Cài đặt giải thuật Bellman – Ford tìm đường đi ngắn nhất từ đỉnh 1 đến các đỉnh còn lại của đồ thị (trọng số có thể âm). In các giá trị pi[i] và p[i] của các đỉnh ra màn hình. In đường đi ngắn nhất từ s đến t.
Hãy kiểm thử với đồ thị bên dưới (s: đỉnh 1, t: đỉnh 8):



Bài tập 5. Áp dụng giải thuật Bellman – Ford kiểm tra xem một đồ thị có chứa chu trình âm hay không. In kết quả YES (nếu đồ thị có chu trình âm) hoặc NO (trường hợp ngược lại).

Bài tập 6 (nâng cao). Tương tự bài tập 5, nhưng thay vì in ra YES/NO hãy in ra các đỉnh trong chu trình âm này.

4. Giải thuật Floyd – Warshall

Tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh (all pair shortest path).

Ý tưởng: áp dụng quy hoạch động tìm đường đi ngắn nhất giữa 2 đỉnh u, v thông qua các đỉnh trung gian {1, 2, .., k}

Gọi pi(u, v, k) là chiều dài đường đi ngắn nhất từ u đến v thông qua các đỉnh {1, 2, …, k}, ta có:

- pi(u, v, 0) = trọng số cung (u,v): đi trực tiếp từ u đến v.
- pi(u, v, k) = min{pi(u, v, k-1), pi(u, k, k-1) + pi(k, v, k-1)}

Đường đi ngắn nhất từ u đến v là: pi(u, v, n).

Biến hỗ trợ

- pi[u][v]: chiều dài đường đi ngắn nhất từ u đến v
- next[u][v]: đỉnh kế tiếp đỉnh u trên đường đi ngắn nhất từ u đến v

Giải thuật:

Khởi tạo:

- pi[u][v] = oo (vô cùng), với mọi u, v
- pi[u][u] = 0, với mọi u
- pi[u][v] = L[u][v], với mọi cung (u, v) của đồ thị
- next[u][v] = -1, với mọi cặp u, v.
- next[u][v] = v với mọi cung (u, v) của đồ thị

Lặp k = 1 đến n

- Với mọi cặp đỉnh (u, v), cập nhật lại pi[u][v] nếu thoả điều kiện

```
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];
}
```

Dựng lại đường đi từ u đến v dựa vào next[u][v].

```
path = [u];
while (u != v) {
        u = next[u][v];
        path = path + [u]; //Thêm u vào đường đi
        //Ta cũng có thể in u ra màn hình thay vì lưu vào đường đi path
}
```

Cài đặt giải thuật:
Sử dụng cách biểu diễn ma trận trọng số như giải thuật Moore – Dijkstra.

```
#define INFINITY 9999999

int pi[MAXN][MAXN];
int next[MAXN][MAXN];

void Floyd_Warshall(Graph* G) {
    int u, v, k;
    for (u = 1; u <= G->n; u++)
        for (v = 1; v <= G->n; v++) {
            pi[u][v] = INFINITY;
            next[u][v] = -1;
        }
    for (u = 1; u <= G->n; u++)
         p[u][u] = 0;

    for (u = 1; u <= G->n; u++)
        for (v = 1; v <= G->n; v++)
            if (G->L[u][v] != NO_EDGE) {
                pi[u][v] = G->L[u][v];
                next[u][v] = v;
            }

    for (k = 1; k <= G->n; k++)
        for (u = 1; u <= G->n; u++)
            for (v = 1; v <= G->n; v++)
                if (pi[u][k] + p[k][v] < pi[u][v]) {
                    pi[u][k] = pi[u][k] + pi[k][v];
                    next[u][v] = next[u][k];
                }
}
```

Bài tập 7. Viết chương trình đọc đồ thị có trọng số từ tập tin. Áp dụng giải thuật Floyd – Warshall tìm đường đi ngắn nhất giữa các cặp đỉnh. In chiều dài ngắn nhất giữa các cặp đỉnh ra màn hình theo dạng:

x -> y: chiều dài
…

Bài tập 8. Viết chương trình đọc đồ thị có trọng số từ tập tin. In đường đi ngắn nhất giữa các đỉnh theo dạng:

x -> y: x -> u1 -> u2 -> … -> y

Phát hiện chu trình âm

Sau khi chạy giải thuật Floyd – Warshall nếu đường chéo của ma trận chiều dài đường đi pi[u][u] < 0 (với u bất kỳ) thì đồ thị đã cho chứa ít nhất 1 chu trình âm.

Khi khởi tạo, ta đã gán pi[u][u] = 0. pi[u][u] sẽ chỉ được cập nhật khi đường đi từ u -> … -> u có chiều dài âm. Tức là có chu trình âm !

```
int negative_cycle = 0;
for (u = 1; u <= G->n; u++)
    if (pi[u][u] < 0) {
        //Đồ thị có chứa chu trình âm
        negative_cycle = 1;
        break;
    }
```

Bài tập 9. Viết chương trình đọc đồ thị có trọng số từ tập tin. Áp dụng giải thuật Floyd – Warshall kiểm tra đồ thị có chưa chu trình âm nào không. Nếu có in ra chu trình âm đó, nếu không in ra "No negative cycle".