TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

ĐỒ ÁN CUỐI KỲ

ĐỀ TÀI: A GENERALIZATION OF DIJKTRA'S ALGORITHM

GVHD: Huỳnh Xuân Phụng

MÃ LỚP HỌC: DS230179_22_1_08

NHÓM 14:

1. Trần Lâm Nhựt Khang - 21110497
2. Đỗ Thanh Khang - 21110492
3. Nguyễn Thanh Huy - 2111047

Thành phố Hồ Chí Minh, Tháng 11 năm 2022

DANH SÁCH NHÓM ĐỒ ÁN CUỐI KỲ MÔN
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT
HỌC KỲ I NĂM HỌC 2022 -2023

STT	Họ và tên	Mã số sinh viên	Tỷ lệ % tham gia
1	Trần Lâm Nhựt Khang	21110497	100%
2	Đỗ Thanh Khang	21110492	100%
3	Nguyễn Thanh Huy	21110473	100%

NHẬN XÉT CỦA GIẢNG VIÊN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Điểm: ……..

KÝ TÊN

MỤC LỤC

1. PHÂN TÍCH THUẬT TOÁN.

1.1. Một số định nghĩa và công thức.

Dijkstra's Algorithm: Thuật toán Dijkstra, mang tên của nhà khoa học máy tính Edsger Dijkstra, là một thuật toán giải quyết bài toán đường đi ngắn nhất từ một đỉnh đến các đỉnh còn lại của đồ thị có hướng, các cạnh mang trọng số không âm.

Generalization: Tổng quát hóa là một dạng trừu tượng hóa, theo đó các thuộc tính chung của các trường hợp cụ thể được hình thành dưới dạng các khái niệm hoặc tuyên bố chung. Khái quát hóa cho thấy sự tồn tại của một miền hoặc tập hợp các phần tử, cũng như một hoặc nhiều đặc điểm chung được chia sẻ bởi các phần tử đó.

Code-generation : Là một phần của chuỗi quy trình của trình biên dịch và chuyển đổi biểu diễn trung gian của mã nguồn thành một dạng (ví dụ: mã máy) mà hệ thống đích có thể dễ dàng thực thi.

Superior function: Superior function là hàm đơn điệu không giảm trong mỗi biến. Ví dụ trong đề tài ta gặp $g(X_1,\ldots, X_k)$.

Context-free grammar: Là một ngữ pháp chính thức được sử dụng để tạo ra tất cả các mẫu chuỗi có thể có trong một ngôn ngữ chính thức nhất định.

Nonterminal: được hiểu tương tự như một điểm xác định.

m(A): thể hiện độ dài ngắn nhất, hay giá trị nhỏ nhất từ điểm gốc đến điểm A.

$\nu[Y] = \min\{g(\mu[X_1],\ldots, (\mu[X_k]) \mid Y \rightarrow g(X_1,\ldots, X_k)$ is a production and $\{X_1,\ldots, X_k\} \subseteq D\}$: Gán giá trị $\nu[Y]$ bằng giá trị nhỏ nhất của sự kết hợp các con đường đã được xác định cộng với độ dài để đến được điểm Y.

1.2. Những đặc điểm quan trọng.

1.2.1. Tính hữu hạn.

Thuật toán được xác định hữu hạn. Thuật toán kết thúc sau 5 bước.

Với trường hợp không có điểm Y nào, thuật toán sẽ hiển nhiên kết thúc.

Với trường hợp có ít nhất một điểm Y:

Ở bước 3, luôn tìm gán được giá trị cho ν[Y] và vì thế luôn tìm được giá trị nhỏ nhất ν[Y] ở bước 4. Cuối cùng thêm giá trị Y đã xác định được ν[Y] nhỏ nhất vào D. Bằng cách này, các điểm sẽ lần lượt vào D. Và khi D chứa tất cả các điểm, thuật toán sẽ kết thúc.

1.2.2 Tính xác định.

Mỗi bước phải được xác định một cách chính xác, theo một trình tự nhất định.

Thuật toán sẽ hoạt động qua 2 phần tử ν[Y] và μ[Y] của các điểm Y (nonterminal). Khi kết thúc thuật toán, các giá trị μ[Y] tương ứng là các m(Y) (Độ dài ngắn nhất hay giá trị nhỏ nhất từ điểm gốc đến Y).

Bước 1: Tạo tập D rỗng: Để chứa các điểm Y đã được duyệt - điểm đã xác định được độ dài ngắn nhất.

Bước 2: Nếu tất cả điểm Y nằm trong D thì dừng thuật toán: Tạo điều kiện dừng, thuật toán sẽ kết thúc điều đó mang ý nghĩa là tất cả các điểm đã duyệt qua và đã tạo được đường đi ngắn nhất đến tất cả các điểm.

Bước 3: Với mỗi điểm $Y \notin D$, tính:

$$\nu[Y] = \min(\nu[Y], \mu[X_k] + d)$$

Y - Điểm đang xét qua

X_k - Các điểm xung quanh điểm Y mà đã xác định U[Y].

d - Trọng số cạch Y với X_k, hay giá trị để đạt được đến Y từ điểm X_k.

Ta so sánh được độ dài khi đi trực tiếp nếu có thể hay qua các đường đi ngắn nhất khác đã xác định sau đó mới qua điểm đó thì tạo được đường ngắn hơn.Sau đó ,ta gán giá trị tìm được.

Ban đầu khi danh sách rỗng, ν[Y] được thiết lập bằng vô cùng.

Bước 4: Chọn Y ∉ D với giá trị ν[Y] bé nhất và đặt μ[Y] = ν[Y] (

Xác định đường đi ngắn nhất từ gốc đến điểm Y, khi đó: m(Y)= μ[Y], cũng như tạo đường trung gian ngắn nhất cho các đường khác khi so sánh).

Bước 5: Thêm Y vào D: Thể hiện Y đã được duyệt qua, đã tìm được đường ngắn nhất đến Y, không duyệt lại điểm đó). Quay về bước 2 (để tiếp tục vòng lặp).

1.2.3. Input.

N: Số các điểm Y.

N điểm Y (nonterminal) và danh sách các đường đi (cạnh) kèm với trọng số của các đường đi đó. Thường được biểu diễn dưới dạng ma trận kề (hoặc danh sách kề).

Điểm làm gốc.

1.2.4. Output.

Danh sách độ dài ngắn nhất từ điểm xuất phát đến các điểm còn lại.

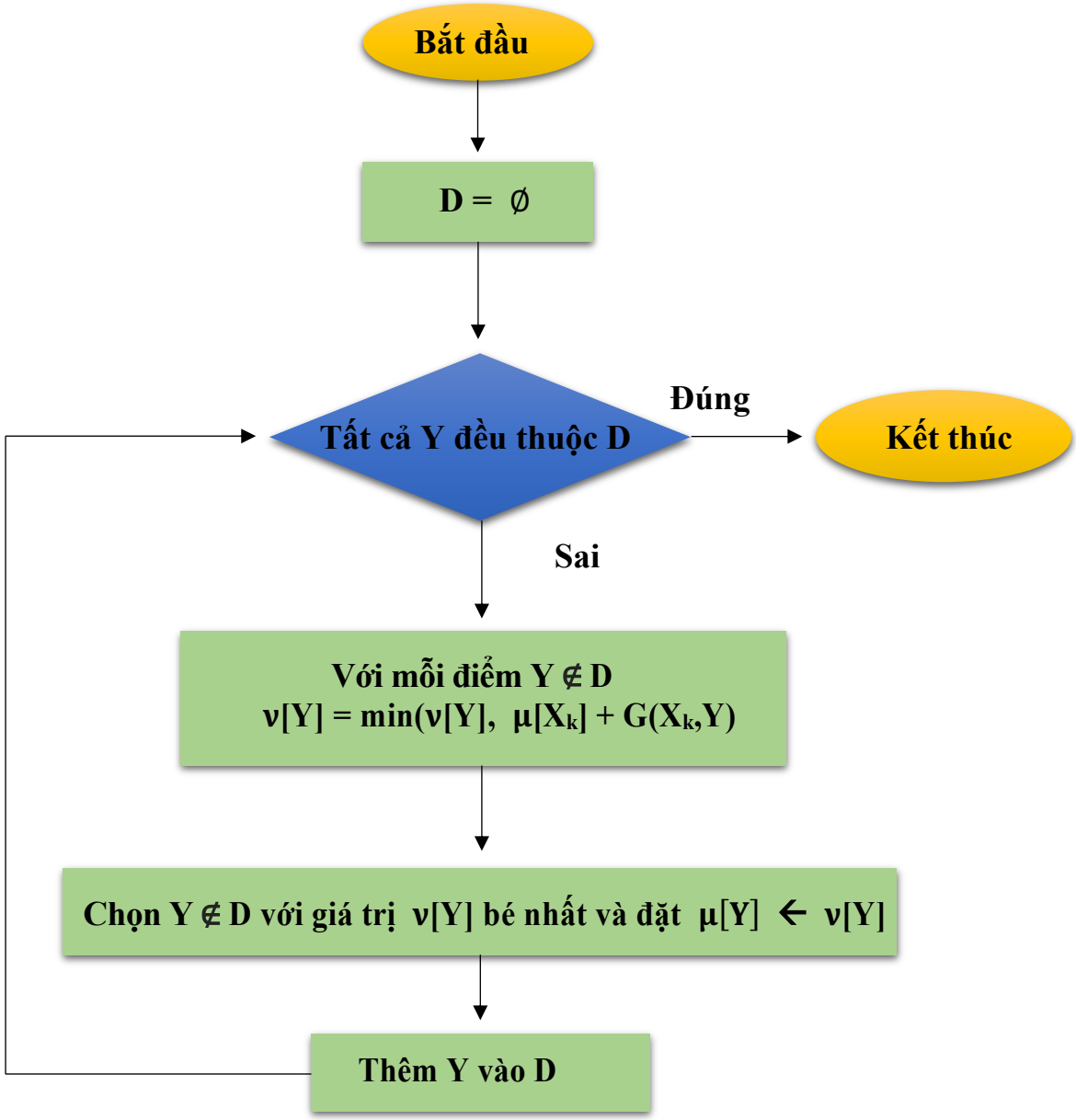
1.2.5. Tính hiệu quả.

Thời gian chạy của thuật toán được giới hạn bởi một số lần không đổi $T = m \log n + t$.

Trong đó m - số con đường (productions), n - số điểm Y (nonterminals) và t - tổng độ dài của các con đường (total length of all productions).

Để có được độ phức tạp thấp là đổi $T = m \log n + t$. Ta cần dùng Priority queue để lưu các phần tử chưa xác định được đường đi ngắn nhất. Từ đó lấy phần tử nhỏ nhất khỏi Priority queue, và tính các giá trị V[Y] mới (với các Y liền kề với điểm vừa được lấy ra).

1.3. Sơ đồ thuật toán.



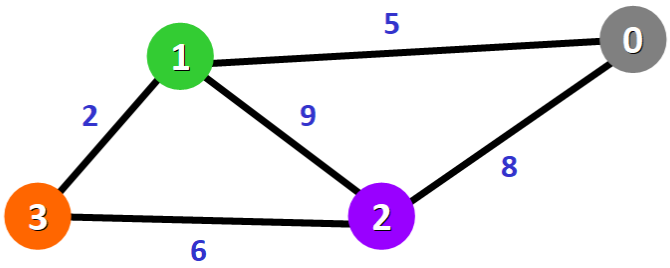
Trong đó:

D : là tập hợp chứa các điểm **Y** đã tìm được đường đi ngắn nhất.

$G(X_k,Y)$: là trọng số giữa **Y** với cạnh kề **X_k**.

v[Y] : là đường đi từ điểm gốc đến **Y**.

μ[Y] : là đường đi ngắn nhất từ điểm gốc đến **Y**.



1.4. Ví dụ.

Với điểm gốc 0, khoảng cách ngắn nhất đến các điểm khác được xác định bởi mảng μ[] sao khi thuật toán kết thúc.

μ[1]	μ [2]	μ [3]	ν [1]	ν [2]	ν [3]
-	-	-	5	8	∞
5	-	-		8	7
5	7	-		8	
5	7	8			

Sau cùng ta được: m(1) = 5, m(2) = 7, m(3) = 8

2. CHỨNG MINH CÔNG THỨC.

ν[Y] = min(ν[Y], μ[X_k] + d)

Y - Điểm đang xét qua

X_k - Các điểm xung quanh điểm Y mà đã xác định μ [Y].

d - Trọng số cạch Y với X_k, hay giá trị để đạt được đến Y từ điểm X_k.

Để thuật toán đúng, ta cần chứng minh m(Y) là giá trị nhỏ nhất từ gốc đến điểm Y.

Với N = 1: Có duy nhất một điểm không tồn tại mảng μ và ν.

Với N = 2: μ [1] được gán bằng ν [1] = d (khoảng cách từ điểm gốc đến Y). Cũng chính là m(1). Sau đó thuật toán kết thúc và hoàn toàn đúng.

Với N > 2: Thuật toán lần lượt xác định các giá trị đường ngắn nhất μ [Y], từ μ [Y] tạo nên các giá trị ngắn ν khác qua việc đi qua nó nếu có thể. Nên giá trị con đường ν[X] (X là điểm nằm xung quanh Y) sẽ dần trở nên nhỏ nhất có thể. Qua mỗi vòng lặp lại đưa giá trị ν nhỏ nhất thành μ. Việc lặp như vậy sẽ tìm được đường đi ngắn nhất một cách đúng đắn.

3. CÀI ĐẶT CHƯƠNG TRÌNH BẰNG NGÔN NGỮ C/C++.

Trước tiên cần tạo một Priority Queue để dễ dàng tìm v nhỏ nhất và thay thể điều kiện duyệt là kiểm tra Priority Queue có rỗng không thay cho tập D.

Tạo các Node:

```
struct QueueNode
{
    int V; //Node Y
    int Vy; // Sau khi kết thúc các Vy là giá trị nhỏ nhất của độ dài từ gốc đến Y

    void init(int V, int Vy){

        this->V  = V;
        this->Vy = Vy;
    }
};
```

Priority Queue gồm có các biến dữ liệu cơ bản:

```
struct PriorityQueue
{
    int currentSize;
    int maxSize;
    int *position; //Dùng để tra vị trí V trong mảng ptrToArray
    QueueNode **ptrToArray;

    //Khởi tạo PriorityQueue
    void init(int maxSize){

        this->currentSize = 0;
        this->maxSize = maxSize;
        position = new int[maxSize+1];
        ptrToArray = new QueueNode*[maxSize+1];
    }

    bool isEmpty()
    {
        return currentSize == 0;
    }
```

Tiếp theo cần phải có các hàm để lấy vị trí các điểm Y và hàm thay đổi chỗ.

```
//Đổi chỗ 2 Node
void swap(QueueNode *&nodeA, QueueNode *&nodeB)
{
    QueueNode *tmp = nodeA;
    nodeA = nodeB;
    nodeB = tmp;
}
// Lấy vị trí của Key Left
int getIndexLeft(int index){

    return (2*index + 1);
}

// Lấy vị trí của Key Right
int getIndexRight(int index){

    return (2*index + 2);
}
// Lấy vị trí của Key Parent
int getIndexParent(int index){

    return (index-1)/2;
}
```

Hàm decreaseKey để thay đổi giá trị của v. Qua việc tính công thức v[Y] = min(v[Y], $\mu[X_k] + d$)

```
//Hàm thay đổi giá trị Vy của một V nào đó
void decreaseKey(int V, int Vy)
{
    int index = position[V];
    ptrToArray[index]->Vy = Vy;

    //Khôi phục thuộc tính
    while(index && ptrToArray[getIndexParent(index)]->Vy > ptrToArray[index]->Vy){

        position[ptrToArray[index]->V] = getIndexParent(index);
        position[ptrToArray[getIndexParent(index)]->V] = index;

        swap(ptrToArray[index], ptrToArray[getIndexParent(index)]);
        index = getIndexParent(index);
    }
}
```

Thay vì thêm giá trị một cách thủ công, ta có thể tạo hàm thêm giá trị insertData, sau khi giá trị thêm vào, cần phải khôi phục thuộc tính của Priority Queue.

```
//Hàm để thêm Node mới vào priority queue
void insertData(int V, double Vy)
{
    int index = currentSize;
    currentSize++;
    QueueNode *nodeTmp = (QueueNode*)malloc(sizeof(QueueNode));
    nodeTmp->init(V, Vy);
    ptrToArray[index] = nodeTmp;
    position[V] = index;

    // Khôi phục thuộc tính nếu nó bị vi phạm bởi việc thêm vào trên
    while(index != 0 && ptrToArray[getIndexParent(index)]->Vy > ptrToArray[index]->Vy){
        position[ptrToArray[index]->V] = getIndexParent(index);
        position[ptrToArray[getIndexParent(index)]->V] = index;
        swap(ptrToArray[index], ptrToArray[getIndexParent(index)]);
        index = getIndexParent(index);
    }
}
```

Với đầu vào là ma trận, ta cần chuyển nó về dạng đồ thị. Đồ thị chứa N điểm Y, và mảng động listNode với phần tử listNode[i] sẽ mang một danh sách liên kết chứa các điểm kề với điểm i.

```
struct Node
{
    int dest;  //Node đích mà nó đến
    int cost;  //Giá trị của đường đi đến node đích
    Node *next; //Liên kết đến các đích khác của src
    bool CreateNode(int dest, int cost)
    {
        this -> dest = dest;
        this -> cost = cost;
        this -> next = nullptr;
        return true;
    }
};
struct ListNode
{
    Node *head;
};
```

Đồ thị (Graph) có hàm CreateGraph để khởi tạo đồ thị và hàm addEdge() để làm nên các cạch của đồ thị với các trọng số d. Việc thêm này sử dụng khả năng thêm vào đầu của danh sách liên kết với độ phức tạp O(1).

```
void Dijkstra(Graph *graph, int G, int V[], int Trace[])
{
    int index = graph -> N;
    bool *Approved = new bool[index]; // Dùng kiểm tra node đã tìm đc đường đi ngắn nhất ch
    int i;
    Node *tmp = graph ->listNode[G].head;
    PriorityQueue *queue = (PriorityQueue*)malloc(sizeof(PriorityQueue));
    queue->init(index);

    //Gán giá trị ban đầu cho các điểm V[Y]
    for (i = 0; i < index; i++)
    {
        V[i] = INT_MAX;
        Approved[i] = false;
        Trace[i] = G;
    }
    while (tmp != nullptr)
    {
        V[tmp->dest] = tmp->cost;
        tmp = tmp -> next;
    }
    V[G] = 0;
    for (i = 0; i < index; i++)
    {
        queue->insertData(i, V[i]);
    }
```

Đến bước này, ta đã đủ các điều kiện cần thiết để tạo thuật toán Dijkstra với độ phức tạp thấp nhất.

```
void Dijkstra(Graph *graph, int G, int V[], int Trace[])
{
    int index = graph -> N;
    bool *Approved = new bool[index]; // Dùng kiểm tra node đã tìm đc đường đi ngắn nhất ch
    int i;
    Node *tmp = graph ->listNode[G].head;
    PriorityQueue *queue = (PriorityQueue*)malloc(sizeof(PriorityQueue));
    queue->init(index);

    //Gán giá trị ban đầu cho các điểm V[Y]
    for (i = 0; i < index; i++)
    {
        V[i] = INT_MAX;
        Approved[i] = false;
        Trace[i] = G;
    }
    while (tmp != nullptr)
    {
        V[tmp->dest] = tmp->cost;
        tmp = tmp -> next;
    }
    V[G] = 0;
    for (i = 0; i < index; i++)
    {
        queue->insertData(i, V[i]);
    }
```

Ta đã cài đặt các giá trị ban đầu cho Priority Queue, phần code còn có thêm mảng Trace[] để dễ dàng truy vết đường đi ngắn nhất.

Tiếp theo, ta thực hiện vòng lặp cho đến khi Priority Queue rỗng, thì thuật toán kết thúc. Với mỗi vòng lắp ta lấy Node nhỏ nhất sau đó tính toán các giá trị và thay đổi thế nó nếu có giá trị nhỏ hơn.

```
    while (queue->isEmpty() == false)
    {
        QueueNode* nodeTmp = queue->getElementMin(); //node đã tìm đc độ dài nhỏ nhất đến src
        Approved[nodeTmp->V] = true;
        Node  *p = graph->listNode[nodeTmp->V].head; //trỏ p để duyệt qua các node xung quanh
        //Duyệt các node xung quanh
        while (p != nullptr)
        {   int Y = p -> dest;
            int d = p -> cost;  // = nodeTmp -> Vy; (Khoảng cách a đến b tương tự b đến a)
            if (V[nodeTmp->V] != INT_MAX && Approved[Y] == false && V[nodeTmp->V] + d < V[Y])
            {
                V[Y] = V[nodeTmp->V] + d;
                queue->decreaseKey(Y,V[Y]);
                Trace[Y] = nodeTmp->V;
                //Nếu giá trị đường đi nhỏ hơn thì thay đổi trong queue, đánh dấu lại bởi Trace
            }
            p = p -> next;
        }
        free(nodeTmp);
        free(p);
    }
    free(Approved);
    free(tmp);
    free(queue);
}
```

Dữ liệu nhập vào từ file Matrix.txt:

Matrix.txt

```
1    4 0
2    0 5 8 0
3    5 0 9 2
4    8 9 0 6
5    0 2 6 0
```

Ta cần dùng một hàm chuyển dạng ma trận sang Graph.

```
//Chuyển đầu vào là ma trận sang đồ thị
Graph *MatrixToGraph(int **Matrix, int NumberOfNonTerminal)
{
    Graph *graph = (Graph *)malloc(sizeof(Graph));
    graph->CreateGraph(NumberOfNonTerminal);
    for (int i = 0; i < NumberOfNonTerminal; i++)
    {
        for (int j = i+1 ; j < NumberOfNonTerminal; j++)
        {
            if (Matrix[i][j]!=0) graph->addEdge(i,j,Matrix[i][j]);
        }
    }
    return graph;
}
```

Ở main(), ta cần lấy dữ liệu từ Matrix.txt và thiết lập giá trị ban đầu cho các biến.

```cpp
int main()
{

    int NumberOfNonTerminal;
    int StartPoint;
    ifstream ifs("Matrix.txt");
    ifs >> NumberOfNonTerminal;
    ifs >> StartPoint;
    int *ResultTanks = new int[NumberOfNonTerminal];
    int *Trace = new int[NumberOfNonTerminal];
    int **Matrix;

    Matrix = (int**)malloc(NumberOfNonTerminal*sizeof(int*));
    for (int i = 0; i < NumberOfNonTerminal; i++)
    {
        Matrix[i] = (int*)malloc(NumberOfNonTerminal*sizeof(int)) ;
    }
    for (int i = 0; i < NumberOfNonTerminal; i++)
    {
        for (int j = 0; j < NumberOfNonTerminal; j++)
        {
            ifs>>Matrix[i][j];
        }
    }

    //
    //          (1) ----5----(0)
    //          /  |        /
    //         2   9      8
    //        /    |    /      Đồ thị để tạo ma trận trong file Matran.txt
    //       (3)---6-- (2)
    //
    //
```

Tạo đồ thị graph sao đó sử dụng thuật toán Dijsktra để lưu kết quả vào mảng ResultTanks[] và truy vết bằng mảng Trace[], sau cùng giải phóng vùng nhớ.

```cpp
    Graph *graph = MatrixToGraph(Matrix,NumberOfNonTerminal);
    Dijkstra(graph,0,ResultTanks,Trace);
    for (int i = 0; i < NumberOfNonTerminal; i++)
    {
        cout<<StartPoint<<" -> "<<i<<" cost: "<<ResultTanks[i]<<"\n";
        int vt = i;
        cout<<vt<<" , ";
        while (Trace[vt]!=StartPoint)
        {
            cout<<Trace[vt]<<" , ";
            vt = Trace[vt];
        }
        cout<<StartPoint<<"\n";
    }

    free(ResultTanks);
    free(Trace);
    for (int i=0; i<NumberOfNonTerminal; i++)  free(Matrix[i]);
        free(Matrix);
    free(graph->listNode);
    free(graph);
}
```

Kết quả chạy ở màng hình Console: 0 -> 3 cost: 7 mang ý nghĩa để đi từ điểm 0 đến điểm 3 tiêu tốn ít nhất một giá trị là 7. Để đạt được điều đó, ta cần đi từ 0 đến 1, sau đó từ 1 đến 3.

```
0 -> 0 cost: 0
0 , 0
0 -> 1 cost: 5
1 , 0
0 -> 2 cost: 8
2 , 0
0 -> 3 cost: 7
3 , 1 , 0
```

4. MỘT SỐ ỨNG DỤNG CỦA THUẬT TOÁN VÀ CHƯƠNG TRÌNH DEMO.

4.1. Ứng dụng của thuật toán.

4.1.1. Dịch vụ bản đồ Kỹ thuật số trong Google Maps (Digital Mapping Services in Google Maps).

Thuật toán Dijkstra là một trong những thuật toán được Google Maps sử dụng để tìm khoảng tối thiểu giữa hai vị trí dọc theo đường dẫn**.** Coi Việt Nam là một biểu đồ và biểu thị một thành phố/địa điểm có các đỉnh và tuyến đường giữa các thành phố/địa điểm đó là cạnh, sau đó bằng cách sử dụng thuật toán này, các tuyến đường ngắn nhất giữa hai thành phố/địa điểm bất kỳ hoặc từ một thành phố/địa điểm này đến một thành phố khác /địa điểm có thể được tính toán.

4.1.2. Ứng dụng mạng xã hội (Social Networking Applications).

Trong nhiều ứng dụng, bạn có thể đã thấy ứng dụng gợi ý danh sách bạn bè mà một người dùng cụ thể có thể biết. Bạn nghĩ làm thế nào để nhiều công ty truyền thông xã hội triển khai tính năng này một cách hiệu quả, đặc biệt là khi hệ thống có hơn một tỷ người dùng. Thuật toán Dijkstra tiêu chuẩn có thể được áp dụng bằng cách sử dụng đường dẫn ngắn nhất giữa những người dùng được đo thông qua bắt tay hoặc kết nối giữa họ. Khi biểu đồ mạng xã hội rất nhỏ, nó sử dụng thuật toán Dijkstra tiêu chuẩn cùng với một số tính năng khác để tìm đường đi ngắn nhất, tuy nhiên, khi biểu đồ ngày càng lớn hơn, thuật toán tiêu chuẩn sẽ mất vài giây để đếm và thay thế nâng cao. các thuật toán được sử dụng.

4.1.3. Con đường Robot (Robotic Path).

Ngày nay, máy bay không người lái và robot đã ra đời, một số là thủ công, một số là tự động. Máy bay không người lái/rô-bốt được tự động hóa và được sử dụng để vận chuyển các gói hàng đến một địa điểm cụ thể hoặc được sử dụng cho một nhiệm vụ được nạp mô-đun thuật toán này để khi biết nguồn và đích, rô-bốt/máy bay không

người lái di chuyển theo hướng đã định bằng cách làm theo con đường ngắn nhất để tiếp tục phân phối gói hàng trong một khoảng thời gian tối thiểu.

4.1.4. Chỉ định máy chủ tệp (Designate file server).

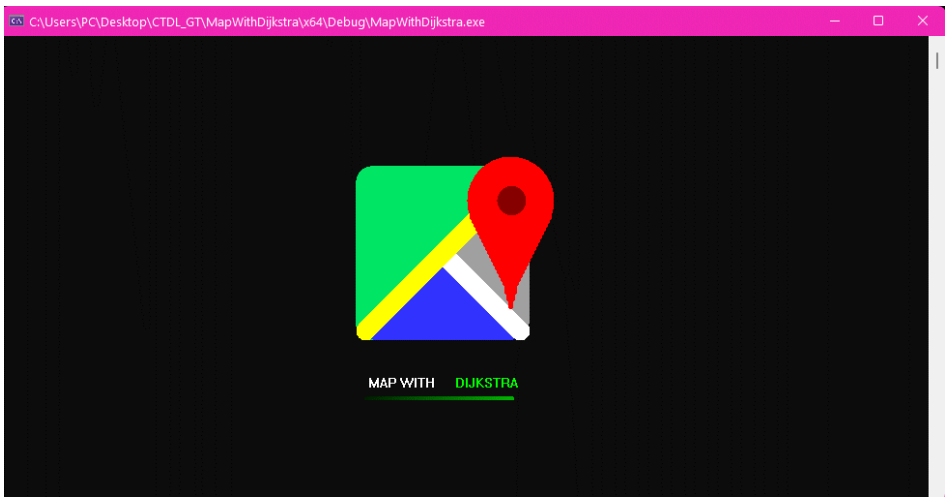
Để chỉ định một máy chủ tệp trong mạng LAN (mạng cục bộ) , có thể sử dụng thuật toán Dijkstra. Hãy xem xét rằng cần có một lượng thời gian vô hạn để truyền tệp từ máy tính này sang máy tính khác. Do đó, để giảm thiểu số "bước nhảy" từ máy chủ tệp đến mọi máy tính khác trên mạng, ý tưởng là sử dụng thuật toán Dijkstra để giảm thiểu đường đi ngắn nhất giữa các mạng dẫn đến số bước nhảy tối thiểu.

4.1.5. Chương trình chuyến bay (Flighting Agenda).

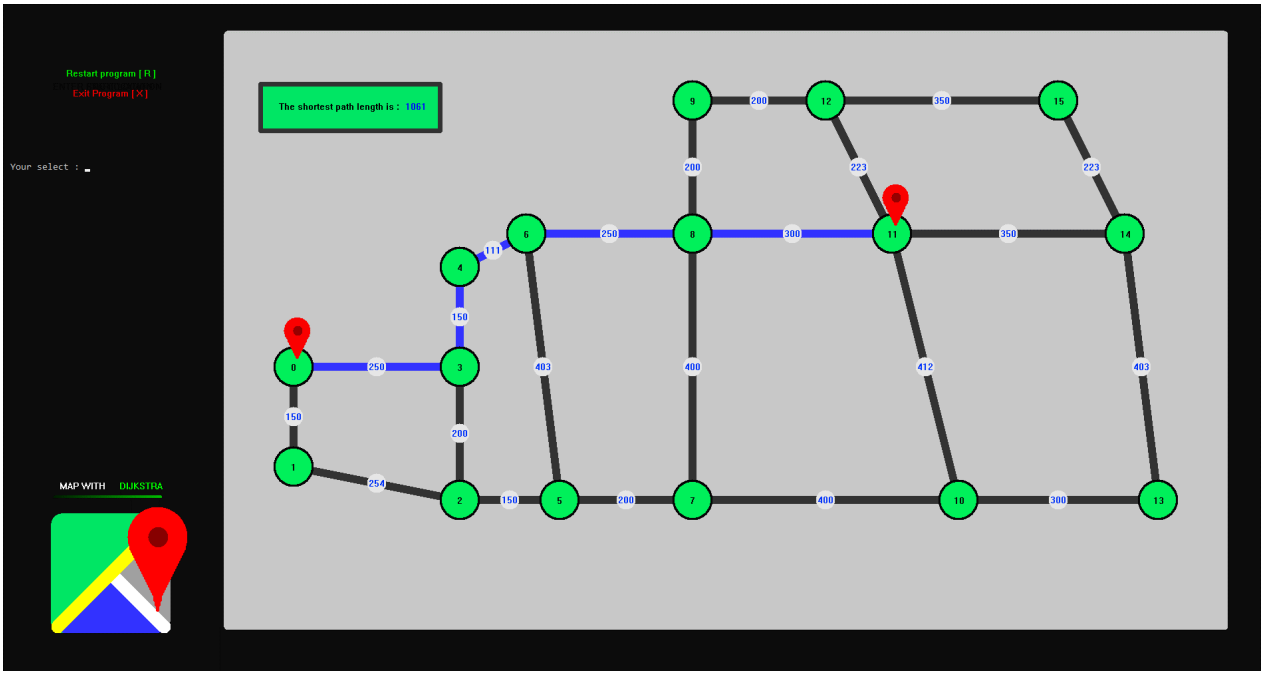
Ví dụ: Nếu một người cần phần mềm để lập lịch trình chuyến bay cho khách hàng. Đại lý có quyền truy cập vào cơ sở dữ liệu với tất cả các sân bay và chuyến bay. Bên cạnh số hiệu chuyến bay, sân bay xuất phát và điểm đến, các chuyến bay còn có giờ khởi hành và giờ đến. Cụ thể, đại lý muốn xác định thời gian đến sớm nhất cho điểm đến được cung cấp sân bay xuất phát và thời gian bắt đầu. Ở đó thuật toán này được sử dụng.

4.2. Một số chương trình demo ứng dụng của thuật toán Dijkstra trên qui mô nhỏ.

4.2.1. Dịch vụ bản đồ Kỹ thuật số trong Google Maps (Digital Mapping Services in Google Maps).



Chương trình được viết bằng ngôn ngữ lập trình C/C++, sử dụng thư viện graphics.h



Giải thích:

Thuật toán Dijkstra sẽ tìm đường dẫn ngắn nhất nối giữa tọa độ xuất phát (người dùng nhập) và tọa độ đến (người dùng nhập) trên Maps của chương trình.

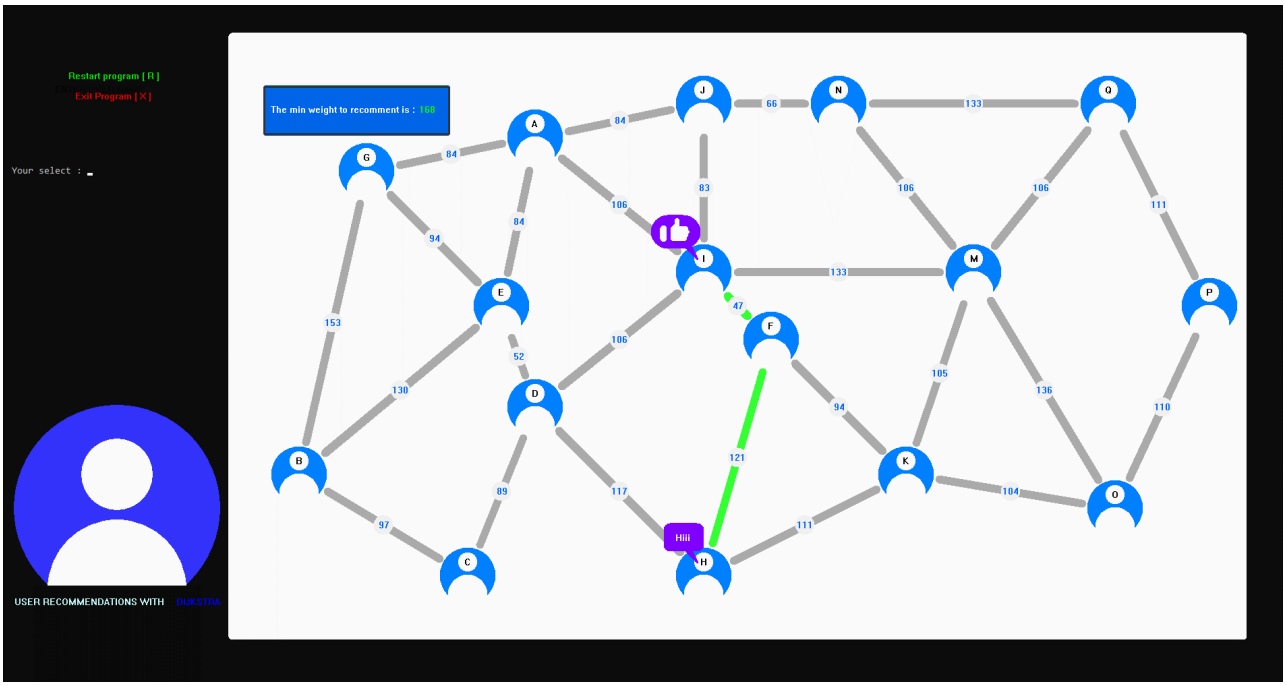
Maps chứa các điểm và khoảng cách (trọng số) giữa các điểm đó trên Maps nếu chúng được nối với nhau.

Trong hình thuật toán tìm đường dẫn ngắn nhất nối giữa 0 và 11 trên Maps, đường dẫn được đánh dấu bởi màu xanh dương và xuất hiện hộp thoại hiển thị độ dài của quảng đường ngắn nhát đấy.

*** Source code chương trình đính kèm trong bài nộp*

4.2.2. Ứng dụng mạng xã hội (Social Networking Applications).

Chương trình được viết bằng ngôn ngữ lập trình C/C++, sử dụng thư viện graphics.h



Giải thích:

Ta có thể nói rằng mạng xã hội là một biểu đồ trong đó:

Các nút là người dùng.

Các cạnh biểu thị nối quan hệ/kết nối.

Trọng lượng là sự gần gũi giữa những người sử dụng.

Để tính toán mức độ gần gũi giữa hai người dùng, nhóm đưa ra một VD về một thuật toán đơn giản:

Khi hai người tạo kết nối, trọng số của họ được gán một giá trị MAX.

Sau đó, bất cứ khi nào người dùng 1 thích bài viết của người dùng 2 hoặc ngược lại, trọng số giữa họ sẽ -2.

Khi người dùng 1 nhận xét bài đăng của người dùng 2 hoặc ngược lại, trọng số giữa họ 4.

=> Trọng số giữa hai người càng ít thì họ càng thân thiết.

Các bước để thuật toán Dijkstra cung cấp đề xuất cho người dùng được chỉ định:

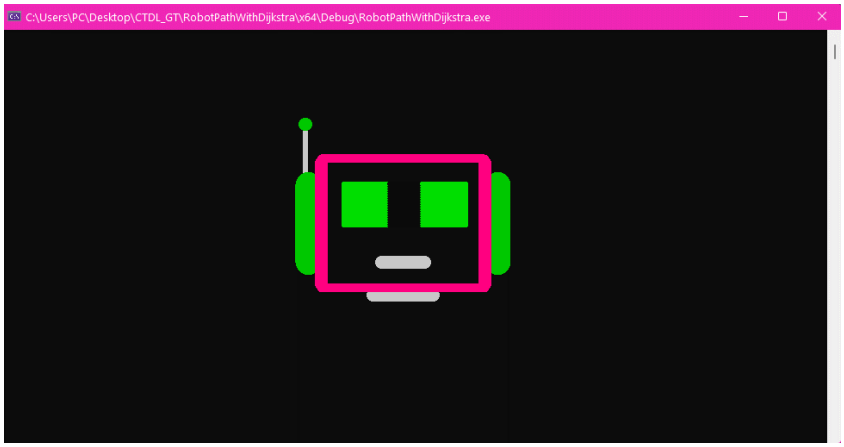
Thuật toán tìm đường dẫn ngắn nhất từ người dùng đó đến tất cả những người dùng khác trong mạng.

Sắp xếp chúng theo trọng lượng (trừ những người đã kết nối)

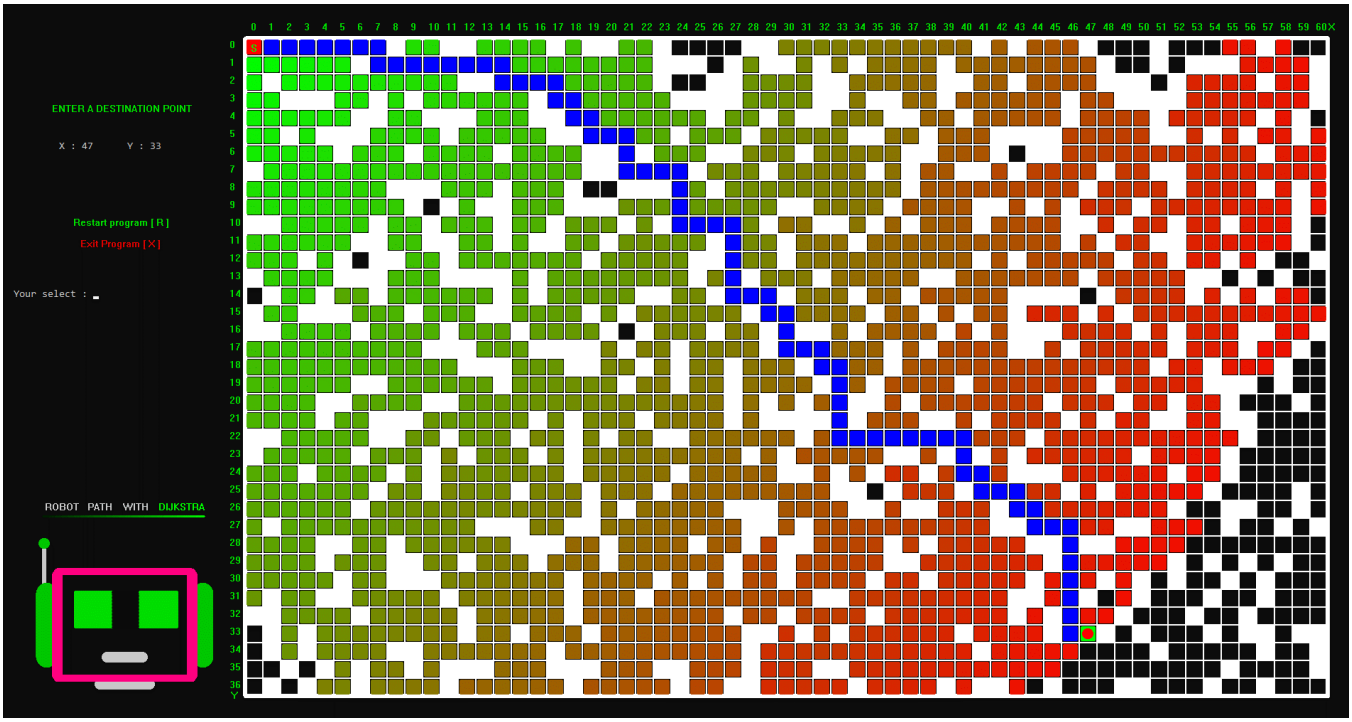
Bây giờ chúng ta sẽ có danh sách người theo thứ tự tăng dần khả năng người dùng sẽ biết họ (tăng dần theo trọng lượng).

Trong hình trên là VD đề xuất người dùng có thể quen biết cho người dùng H.

4.2.3. Con đường Robot (Robotic Path).



Chương trình được viết bằng ngôn ngữ lập trình C/C++, sử dụng thư viện graphics.h



Chú thích:

Thuật toán Dijkstra tìm đường dẫn có khoảng cách ngắn nhất nối từ điểm bắt đầu S (do người dùng nhập) đến điểm kết thúc (do người dùng nhập) trên Maps là ma trận hai chiều được chia thành các ô.

Robot có thể di chuyển giữa các ô theo hướng lên, xuống, trái, phải. Các ô màu trắng là chướng ngại vật, không thể di chuyển vào.

Thuật toán sẽ đi qua các điểm trên ma trận ,đánh dấu lại các điểm đã đi qua và tìm, lưu lại khoảng cách ngắn nhất từ điểm đó đến gốc S, màu của ô chuyển từ xanh sang đỏ khi càng ra xa S. Dừng khi duyệt đến điểm kết thúc.

Thực hiện truy vết từ điểm kết thúc để vẽ đường đi trở lại điểm bắt đầu S – đường đi được đánh dấu màu xanh dương.

5. Phân tích hướng phát triển.

Thuật toán Dijkstra được tạo ra vào năm 1956 và xuất bản năm 1959, được tạo ra để tìm đường đi ngắn nhất trên đồ thị vô hướng có các trọng số không âm. Mục tiêu của thuật toán Dijkstra là có thể mang lại được một bài toán và giải pháp mà kể cả những người không thuần việc tính toán vẫn có thể hiểu được. Ngày nay Dijkstra được sử dụng rộng rãi trong các giao thức định tuyến mạng, chẳng hạn như giao thức định tuyến IS-IS và OSPF. Hiện nay đã xuất hiện nhiều biến thể thuật toán vượt trội hơn Dijkstra cổ điển như Moore-Dijkstra, Dijkstra sử dụng hàng đợi ưu tiên, Dijkstra fibonacci. Trong tương lai Dijkstra vẫn sẽ được ứng dụng rộng rãi và chuyên sâu hơn trong nhiều ứng dụng như: Đặt xe online: grab, gojek, uber… hoặc các ứng dụng mạng xã hội như: facebook, instagram. Dijkstra sẽ được ứng dụng ngày càng nhiều trong việc học do tính dễ phổ cập của chúng.

6. Poster.