BỘ GIÁO DỤC VÀ ĐÀO TẠO

ĐẠI HỌC UEH

KHOA CÔNG NGHỆ THÔNG TIN KINH DOANH

ĐỒ ÁN MÔN HỌC

ĐỀ TÀI

ĐỒ THỊ VÀ ỨNG DỤNG TRONG QUẢN LÝ KHO

Học Phần: Cấu Trúc Dữ Liệu & Giải Thuật

Danh Sách Nhóm:

1. PHAN TRẦN SƠN BẢO
2. ĐỖ XUÂN ĐỨC
3. LÊ HUỲNH HOÀNG KHANG
4. TRƯƠNG VĂN KIỆT
5. NGUYỄN ĐÌNH ĐẠI NHƠN

Chuyên Ngành: KHOA HỌC DỮ LIỆU

Khóa: K46

Giảng Viên: TS. Đặng Ngọc Hoàng Thành

Tp. Hồ Chí Minh, Ngày 11 tháng 12 năm 2021

MỤC LỤC

CHƯƠNG 1. ĐỒ THỊ

1.1. Các Khái Niệm Liên Quan

- Cấu trúc dữ liệu đồ thị là một tập hợp các nút có dữ liệu và được kết nối với các nút khác.
- Đồ thị (Graph) là một dạng cấu trúc dữ liệu được cấu tạo từ tập hợp các đỉnh V (vertex) và tập hợp các cạnh E (edge), được biểu diễn dưới dạng các cặp đỉnh có thứ tự (u, v). G=(V, E).
- Hai đỉnh được nối với nhau tạo thành một cặp đinh (pair). Mỗi cặp đỉnh như vậy tạo thành một cạnh. Một cạnh được tạo từ một đỉnh duy nhất gọi là khuyên. Nếu tồn tại hai cạnh trên cùng một cặp điểm, thì cặp cạnh đó gọi là cặp cạnh song song (hoặc cạnh bội).
- Đồ thị có hướng: Một đồ thị trong đó cạnh (u, v) không nhất thiết có nghĩa là cũng có cạnh (v, u). Các cạnh trong biểu đồ như vậy được biểu diễn bằng các mũi tên để chỉ hướng của cạnh. Và ngược lại, được gọi là Đồ thị vô hướng.
- Nếu mỗi cạnh của đồ thị được gán với một trọng số, thì đồ thị đó được gọi là :
  Đồ thị có trọng số (hoặc là mạng lưới network).
- Đơn đồ thị là đồ thị (có hướng hoặc vô hướng) không chứa khuyên hoặc cạnh bội.

1.2. Cấu Trúc và Cài Đặt Đồ Thị

➢ **Cấu trúc đồ thị :**

- Đỉnh (Vertex): Mỗi nút của hình được biểu diễn như là một đỉnh.
- Cạnh (Edge): Cạnh biểu diễn một đường nối hai đỉnh.
- Đường: Đường biểu diễn một dãy các cạnh giữa hai đỉnh.
- Kề nhau: Hai đỉnh là kề nhau nếu chúng được kết nối với nhau thông qua một cạnh.

➢ **Cài đặt đồ thị**

- Lớp Đỉnh Vertex

```
public class Vertex
{
```

```
        public bool wasVisited;
        public string label;
        public Vertex(string label)
        {
        this.label = label;
        wasVisited = false;
        }
    }
```

- Biểu Diễn Cạnh Edge

```
int nVertices = 0;
vertices[nVertices] = new Vertex("A");
nVertices++;
vertices[nVertices] = new Vertex("B");
nVertices++;
vertices [nVertices] = new Vertex("C");
nVertices++;
vertices [nVertices] = new Vertex("D");
adjMatrix[0,1] = 1;
adjMatrix[1,0] = 1;
adjMatrix[1,3] = 1;
adjMatrix[3,1] = 1;
```

- Lớp đồ thị Graph

```
public class Graph
{
        private const int NUM_VERTICES = 20 ;
        private Vertex[] vertices;
        private int[,] adjMatrix;
        int numVerts;
        public Graph()
```

```
        {
            vertices = new Vertex[NUM_VERTICES];
            adjMatrix = new int[NUM_VERTICES,
        NUM_VERTICES];
            numVerts = 0;
            for (int j = 0 ; j  <  NUM_VERTICES; j++)
            for (int k = 0 ; k < NUMVERTICES ; k++)
                adjMatrix[ j , k ] = 0;
        }
        public void AddVertex(string label)
        {
          vertices[numVerts] = new Vertex(label);
          numVerts++;
        }
        public void AddEdge (int start, int eend)
        {
          adjMatrix[start, eend] = 1;
          adjMatrix[eend, start] = 1;
        }
        public void ShowVertex(int v)
        {
          Console.Write(vertices[v].label + " ");
        }
    }
```

1.3. Các Thuật Toán Trên Đồ Thị

a) Thuật Toán Tìm Kiếm

- **Tìm kiếm theo chiều sâu – Depth-First Search:**

  B1. Tìm một đỉnh chưa viếng thăm nằm liền kề với đỉnh hiện tại, đánh dấu nó là đã viếng thăm và thêm vào ngăn xếp.

B2. Nếu không thể tìm thấy một đỉnh liền kề chưa được viếng thăm với đỉnh hiện tại, loại bỏ một đỉnh khỏi ngăn xếp, và bắt đầu lại.

B3. Nếu ở B2, ngăn xếp rỗng, thuật toán dừng.

- **Tìm kiếm theo chiều rộng – Breadth First Search**

B1. Chèn đỉnh gốc vào hàng đợi.

B2. Lấy ra đỉnh đầu tiên trong hàng đợi và quan sát nó.

B3. Nếu đỉnh này là đỉnh đích, dừng quá trình tìm kiếm và trả về kết quả.

B4. Nếu không phải thì chèn tất cả các đỉnh kề với đỉnh vừa thăm nhưng chưa được quan sát trước đó vào hàng đợi.

B5. Nếu hàng đợi là rỗng, thì tất cả các đỉnh có thể đến được đều đã được quan sát – dừng việc tìm kiếm và trả về "không thấy".

B6. Nếu hàng đợi không rỗng thì quay về B2.

b) Thuật Toán Dijkstra

B1. Chọn S = {} là tập các đỉnh bao gồm a và b. Với a là đỉnh đang được xét đến, b là các đỉnh đã được xét. a đầu tiên sẽ là đỉnh đích của bài toán tìm đường đi ngắn nhất.

B2. Khởi tạo giải thuật với a là đỉnh đích và cost(N) là giá trị của đường đi ngắn nhất từ N đến a.

B3. Xét các đỉnh N (các đỉnh kề với a) . Gọi d(a,N) là khoảng cách giữa các đỉnh kề với a . Với p = d(a,N) + cost (a). Nếu p < cost(N) thì cost(N) = p . Nếu không thì cost(N) giữ nguyên giá trị .

B4. Sau khi xét hết các đỉnh N kề với a, đánh dấu a thành b .

B5. Tìm a mới với 2 điều kiện: không phải b và cost(a) là nhỏ nhất

B6. Nếu tập S = {} chứa đủ các đỉnh của đồ thị thì dừng thuật toán. Nếu không thì quay trở lại B3 .

CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ LỚP

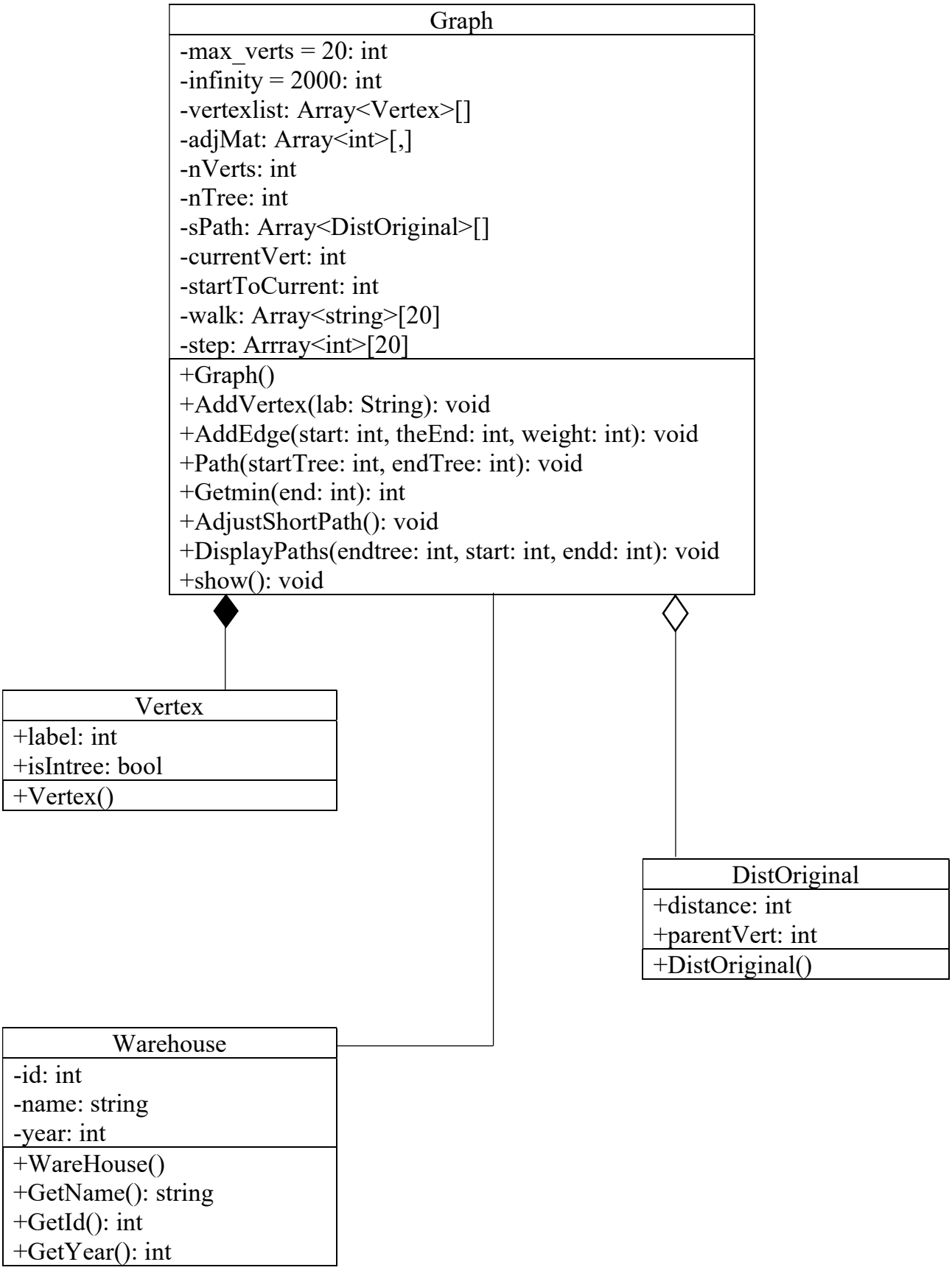
2.1. Phân Tích Bài Toán Tìm Độ Dài Đường Đi Bằng Đồ Thị

* Trong chương trình này, ta sẽ giải quyết vấn đề tìm đường vận chuyển sao cho chi phí của việc vận chuyển là nhỏ nhất. Thuật toán Dijkstra sẽ là một phương án giải quyết tối ưu nhất cho vấn đề này. Ở đây, chúng ta có thể coi như mỗi kho là 1 đỉnh của đồ thị, chi phí vận chuyển từ kho này đến kho khác sẽ giống như quãng đường từ đỉnh này đến đỉnh khác hay nói cách khác là chiều dài của cạnh và tất cả sẽ tạo nên đồ thị của mạng lưới kho. Từ đó, ta có thể tính được và in ra quãng đường vận chuyển để tối ưu hóa chi phí vận chuyển.

* Input: đồ thị có các đỉnh ứng với các kho, chiều dài của cạnh ứng với chi phí vận chuyển từ kho này đến kho kia.

* Output: chiều dài nhỏ nhất giữa hai đỉnh của đồ thị (chi phí nhỏ nhất để vận chuyển từ kho này đến kho khác) và các đỉnh sẽ đi qua (quãng đường di chuyển để đạt được chi phí nhỏ nhất).

2.2. Sơ Đồ Lớp


Graph
-max_verts = 20: int
-infinity = 2000: int
-vertexlist: Array<Vertex>[]
-adjMat: Array<int>[,]
-nVerts: int
-nTree: int
-sPath: Array<DistOriginal>[]
-currentVert: int
-startToCurrent: int
-walk: Array<string>[20]
-step: Arrray<int>[20]
+Graph()
+AddVertex(lab: String): void
+AddEdge(start: int, theEnd: int, weight: int): void
+Path(startTree: int, endTree: int): void
+Getmin(end: int): int
+AdjustShortPath(): void
+DisplayPaths(endtree: int, start: int, endd: int): void
+show(): void
Vertex
+label: int
+isIntree: bool
+Vertex()
DistOriginal
+distance: int
+parentVert: int
+DistOriginal()
Warehouse
-id: int
-name: string
-year: int
+WareHouse()
+GetName(): string
+GetId(): int
+GetYear(): int

2.3. Cài Đặt Lớp

- Lớp WareHouse1-1:

```
public class WareHouse
{
    private int id;
    private string name;
    private int year;
    public string GetName()
    {
        return name;
    }

    public int GetId()
    {
        return id;
    }

    public int GetYear()
    {
        return year;
    }

    public WareHouse(int id, string name, int year)
    {
        this.id = id;
        this.name = name;
        this.year = year;
    }

    override public string ToString()
    {
        return "Warehouse(" + id + "," + name + "," + year + ")";
    }
}
```

+ Vì các Kho đều có các thuộc tính giống nhau như mã kho (id),tên kho, năm thuê, chi phí nên ta sẽ tạo 1 lớp (vì lớp là kiểu dữ liệu tham chiếu nên ta có thể nhập nhiều Kho nhưng chỉ cần tạo 1 lớp) tên WareHouse1-1 với các attribute như id, name, year và các

phương thức như GetId, GetName, GetYear, WareHouse để lưu trữ mã kho, tên kho và năm thuê cũng như có thể lấy ra các biến đó khi cần sử dụng.

- Lớp Vertex:

+ Lớp Vertex dùng lưu trữ các đỉnh (ứng với tên của các Kho) của đồ thị và đánh dấu đỉnh đó là chưa xét xong (tương đương với false).

```
public class Vertex
   {
      public string label;
      public bool isIntree;
      public Vertex(string lab)
      {
         label = lab;
         isIntree = false;
      }
   }
```

- Lớp DistOriginal:

+ Lớp DistOriginal dùng để lưu trữ khoảng cách giữa đỉnh đang xét với các đỉnh kề nó để ta có thể tim được được đi ngắn nhất từ Kho đang xét đến các Kho kề nó.

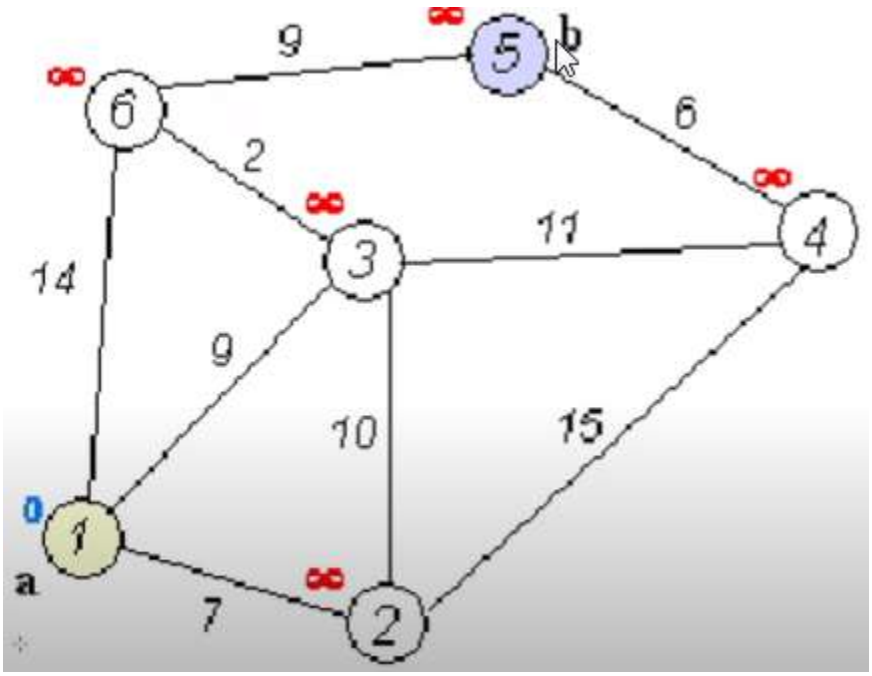
```
public class DistOriginal
   {
      public int distance;
      public int parentVert;
      public DistOriginal(int pv, int d)
      {
         distance = d;
         parentVert = pv;
      }
```

- Lớp Graph:

+ Đối với lớp Graph, ta tạo 1 mảng để lưu lại đường đi của các đỉnh để tìm đường đi ngắn nhất. Ban đầu ta cho trọng số của đỉnh xuất phát (Kho1) bằng 0 và trọng số của các đỉnh còn lại là vô

cùng. Sau đó, ta xét khoảng cách của Kho1 với các đỉnh kề nó để tìm được đường đi ngắn nhất từ Kho1 đến các Kho kề nó và chọn Kho có d(Kho1,KhoX) ngắn nhất đó là Kho tiếp theo để xét và tìm đường đi ngắn nhất đến các Kho kề KhoX, và chuyển trạng thái của Kho1 từ chưa thăm sang đã thăm và không xét nó nữa. Nếu trọng số của ban đầu của KhoX lớn hơn trọng số mới tính được của KhoX thì chọn trọng số mới tính được của KhoX và bỏ trọng số ban đầu đi. Tiếp tục làm như vậy cho đến khi tới Kho cuối cùng ta sẽ tìm được đường đi ngắn nhất từ Kho1 đến KhoN.

+ Ví dụ:



Ta tìm đường đi ngắn nhất từ Kho1 đến Kho5. Xét Kho1 và các Kho kề Kho1 là Kho2, Kho3 và Kho6. Ta thấy được d(Kho1,Kho2) = 7 trọng số của Kho2 từ vô cùng đổi thành 7; d(Kho1,Kho3) = 9 trọng số của Kho2 từ vô cùng đổi thành 9; d(Kho1,Kho6) = 14 trọng số của Kho2 từ vô cùng đổi thành 14; ta tìm được đường đi ngắn nhất từ Kho1 đến các Kho kề nó là Kho2 với d(Kho1,Kho2) = 7, và ta khóa Kho1 lại (chuyển sang trạng thái đã xét xong) và không xét nó nữa. Tiếp theo ta tìm đường đi ngắn nhất từ Kho2 tới các Kho kề nó là Kho3 và Kho4. Ta thấy d(Kho2,Kho3) = 7 + 10 = 17, nhưng trọng số mới của Kho3 lại lớn hơn trọng số cũ của nó (17 > 9) nên d(Kho2,Kho3) sẽ bằng 9 và trọng số của Kho3 vẫn giữ nguyên; d(Kho2,Kho4) = 7 + 15 = 22 và trọng số của Kho4 từ vô cùng đổi sang 22 (vì 22 nhỏ hơn vô cùng). Và vì 9 < 22 nên đường đi ngắn nhất từ

Kho2 đến các Kho kề nó là Kho3 và Kho4 là đến Kho3 (=9), sau đó ta khóa Kho2 lại (chuyển sang trạng thái đã xét xong). Tiếp tục ta xét Kho3 để tìm đường đi ngắn nhất từ Kho3 đến các Kho kề nó, cho tới khi ta khóa (chuyển sang trạng thái đã xét xong) hết cả 5 Kho thì ta tìm được đường đi ngắn nhất từ Kho1 đến Kho5 là 20.

```
public class Graph
   {
      private const int max_verts = 20;
      int infinity = 2000;
      Vertex[] vertexlist;
      int[,] adjMat;
      int nVerts;
      int nTree;
      DistOriginal[] sPath;
      int currentVert;
      int startToCurrent;
      string[] walk = new string[20];
      int[] step = new int[20];

      public Graph()
      {
         vertexlist = new Vertex[max_verts];
         adjMat = new int[max_verts, max_verts];
         nVerts = 0;
         nTree = 0;
         for (int i = 0; i <= max_verts - 1; i++)
            for (int j = 0; j <= max_verts - 1; j++)
            {
               adjMat[i, j] = infinity;
            }
         sPath = new DistOriginal[max_verts];
      }
      public void AddVertex(String lab)
      {
         vertexlist[nVerts] = new Vertex(lab);
         nVerts++;
      }
      public void AddEdge(int start, int theEnd, int weight)
      {
         adjMat[start - 1, theEnd - 1] = weight;
      }
      public void Path(int startTree, int endTree)
      {
         startTree--;
         endTree--;
         int start = 0;
```

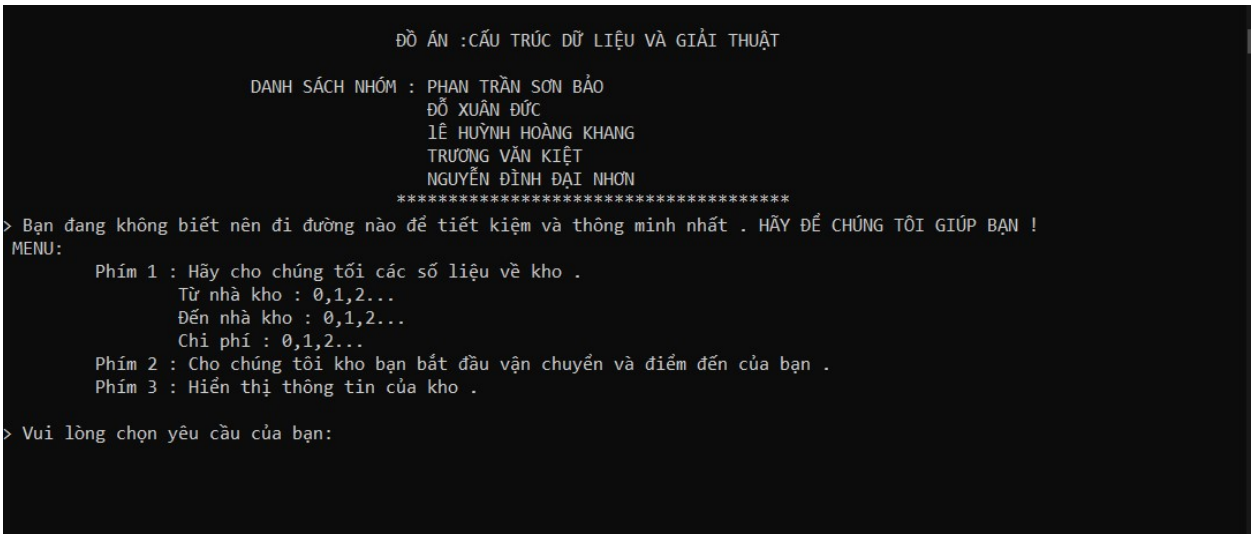
```
    int end = 0;
    if (startTree == endTree) Console.WriteLine("chi phí là 0 , vì hai nhà kho đó là một ");
    else
    {
        if (startTree > endTree)
        {
            start = endTree;
            end = startTree;
        }
        else
        {
            start = startTree;
            end = endTree;
        }
        vertexlist[start].isIntree = true;
        nTree = 1;
        for (int i = 0; i <= nVerts; i++)
        {
            int tempDist = adjMat[start, i];
            sPath[i] = new DistOriginal(start, tempDist);
        }
        currentVert = start;
        while (nTree < nVerts)
        {

        int indexMin = GetMin(end);
        int minDist = sPath[indexMin].distance;
        currentVert = indexMin;
        startToCurrent = sPath[indexMin].distance;
        vertexlist[currentVert].isIntree = true;
        nTree++;
        AdjustShortPath();
        }
        Console.Write("Từ " + vertexlist[startTree].label);
        DisplayPaths(endTree, start, end);
        nTree = 0;

        Console.WriteLine();
        for (int i = 0; i <= nVerts - 1; i++)
            vertexlist[i].isIntree = false;
        for (int i = 0; i < walk.Length; i++)
            walk[i] = null;
    }
}
public int GetMin(int end)
{
    int minDist = infinity;
```

```
        int indexMin = 0;
        for (int j = 0; j <= nVerts-1; j++)
            if (!(vertexlist[j].isIntree) && sPath[j].distance < minDist)
            {
                minDist = sPath[j].distance; indexMin = j;
            }

        return indexMin;
    }
    public void AdjustShortPath()
    {
        int column = 1;
        while (column < nVerts)
            if (vertexlist[column].isIntree) column++;
            else
            {
                int currentToFring = adjMat[currentVert, column];
                int startToFringe = startToCurrent + currentToFring;
                int sPathDist = sPath[column].distance;
                if (startToFringe < sPathDist)
                {
                    sPath[column].parentVert = currentVert;
                    sPath[column].distance = startToFringe;
                }
                column++;
            }
    }
    public void DisplayPaths(int endtree, int start, int endd)
    {
        if (sPath[endd].distance == infinity)
            Console.Write(" Không có con đường để đi ");
        else
            Console.Write(" phải chi khoản tiền nhỏ nhất là  " + sPath[endd].distance + " để");
        Console.Write(" đến " + vertexlist[endtree].label);
        Console.WriteLine();
        if (sPath[endd].distance != infinity)
        {
            int k = 0;
            Console.Write(">>con đường : ");
            walk[0] = vertexlist[endd].label;
            k++;
            int p = endd;
             do
             {
                walk[k] = vertexlist[sPath[p].parentVert].label;
                k++;
                 p = sPath[p].parentVert;
```

```
            }
            while (p!=start);
            for (int i = k-1; i >=0; i--)
                if (i != 0) Console.Write(walk[i] + " -> "); else Console.Write(walk[i]);
        }
    }
    public void show()
    {
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("**************");
        for (int i=0;i<adjMat.GetLength(0);i++)
            for(int j = 0; j < adjMat.GetLength(1); j++)
            {
                if (adjMat[i, j] != infinity) Console.WriteLine(" từ " + vertexlist[i].label + " đến " +
vertexlist[j].label + " là " + adjMat[i, j]);
            }
        Console.WriteLine("**************");
    }

}
```

CHƯƠNG 3. THIẾT KẾ GIAO DIỆN

3.1. Giao Diện Menu Chính



3.2. Chi Tiết Chức Năng

A. Phím 1 - Nhập liệu

- Nhập số lượng kho
- Nhập thông tin số năm thuê cho từng kho
- Nhập quảng đường vận chuyển giữa các kho

```
> Vui lòng chọn yêu cầu của bạn:  1
>> Nhập số lượng kho : 4
Mời bạn nhập tên nhà kho: A

Mời bạn nhập năm thuê: 3

Mời bạn nhập tên nhà kho: B

Mời bạn nhập năm thuê: 2

Mời bạn nhập tên nhà kho: C

Mời bạn nhập năm thuê: 4

Mời bạn nhập tên nhà kho: D

Mời bạn nhập năm thuê: 5

Nhà kho đi :1
Nhà kho đến :2
Chí phí di chuyển  :5
**********
Nhập số 1 để tiếp tục nhập dữ liệu
Hoặc nhập số khác bất kì để ngừng việc nhập dữ liệu!
1
**********
Nhà kho đi :2
Nhà kho đến :3
Chí phí di chuyển  :10
**********
Nhập số 1 để tiếp tục nhập dữ liệu
Hoặc nhập số khác bất kì để ngừng việc nhập dữ liệu!
```

B. Phím 2 – Tìm quảng đường ngắn nhất từ nhà kho này tới nhà kho kia

- Nhập vào ID của nhà kho đi
- Nhập vào ID của nhà kho đến
- In ra chi phí nhỏ nhất để di chuyển từ nhà kho này tới nhà kho kia
- In ra đường đi để có chi phí nhỏ nhất

```
> Vui lòng chọn yêu cầu của bạn:  2
*************
 >>> Với:
Chi phí di chuyển từ nhà kho 1 đến nhà kho 2 là 5
Chi phí di chuyển từ nhà kho 2 đến nhà kho 3 là 10
Chi phí di chuyển từ nhà kho 3 đến nhà kho 4 là 15
Chi phí di chuyển từ nhà kho 2 đến nhà kho 4 là 20
Chi phí di chuyển từ nhà kho 1 đến nhà kho 4 là 25
Chi phí di chuyển từ nhà kho 1 đến nhà kho 3 là 12
 >>> Ta có:
Vận chuyển từ nhà kho: 4
Đến nhà kho: 3
Từ nhà kho 4 phải chi khoản tiền nhỏ nhất là  15 để đến nhà kho 3
con đường : nhà kho 3
```

C. Phím 3 – Hiển thị thông tin kho theo yêu cầu

- Nhập vào ID của nhà kho muốn hiện thông tin
- Hiện thông tin về nhà kho đó (ID, tên, năm thuê)

```
> Vui lòng chọn yêu cầu của bạn:  3
Nhập kho bạn muốn hiển thị thông tin
1
Warehouse(1,A,3)

> Vui lòng chọn yêu cầu của bạn:  3
Nhập kho bạn muốn hiển thị thông tin
2
Warehouse(2,B,2)

> Vui lòng chọn yêu cầu của bạn:  3
Nhập kho bạn muốn hiển thị thông tin
3
Warehouse(3,C,4)

> Vui lòng chọn yêu cầu của bạn:  4
Nhập sai !
Nhấn 1 để quay lại menu lựa chọn
nhấn số bất kỳ để thoát chương trình
 >>
```

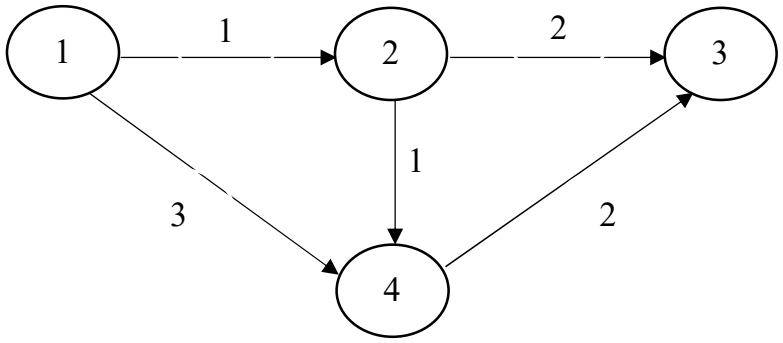
CHƯƠNG 4. THẢO LUẬN & ĐÁNH GIÁ

5.1. Các Kết Quả Nhận Được

- **Hiển thị thông tin nhà kho**

```
Nhập kho bạn muốn hiển thị thông tin
1
Warehouse(1,kho1,2002)
```

- **Hiển thị chi phí và đường đi có chi phí nhỏ nhất**



Với dữ liệu được cho trước như trên:

```
 >>> Với:
Chi phí di chuyển từ nhà kho 1 đến nhà kho 2 là 1
Chi phí di chuyển từ nhà kho 2 đến nhà kho 3 là 2
Chi phí di chuyển từ nhà kho 1 đến nhà kho 4 là 3
Chi phí di chuyển từ nhà kho 4 đến nhà kho 3 là 2
Chi phí di chuyển từ nhà kho 2 đến nhà kho 4 là 1
```

Sau khi nhập vào ta có thể tính được chi phí nhỏ nhất để di chuyển từ một kho tới một kho khác bất kì. Ví dụ như từ kho có ID 1 tới kho có ID 3, ta có thể di chuyển với chi phí nhỏ nhất là:

```
 >>> Ta có:
Vận chuyển từ nhà kho: 1
Đến nhà kho: 3
Từ nhà kho 1 phải chi khoản tiền nhỏ nhất là  3 để đến nhà kho 3
con đường : nhà kho 1 - > nhà kho 2 - > nhà kho 3
```

* Lưu ý: trong trường hợp chúng ta không nhập vào chi phí di chuyển giữa các nhà kho thì chương trình sẽ báo không có đường đi.

5.2. Một Số Tồn Tại

Về cơ bản chương trình đã giải quyết được một cách tổng quát yêu cầu của vấn đề bao gồm lưu trữ thông tin các của các nhà kho (ID, tên, năm thuê) và tính ra được chi phí, hiện thị đường đi sao cho chi phí nhỏ nhất từ nhà kho này đến nhà kho khác.

Đầu tiên, việc phải nhập tay của chương trình có thể tạo ra những sai sót không đáng có so với việc nhập dữ liệu từ các file có sẵn.

Về vấn đề thân thiện với người dùng, chương trình vẫn chưa đạt được như mong đợi khi có nhiều những bước nhập liệu chưa cần thiết dư thừa. Vì vậy nên khi có nhiều nhà kho người dùng sẽ phải nhập rất nhiều lần. Đặc biệt đối với các dạng mà giữa các nhà kho có cả đường đi lẫn đường về thì người dùng sẽ phải thêm 1 lần nhập.

Ngoài ra, chương trình chỉ cung cấp cho chúng ta khả năng tìm kiếm theo ID ứng với nhà kho nào được nhập liệu trước. Ví dụ: nhà kho nhập đầu tiên sẽ có ID là 1, nhà kho nhập thứ hai sẽ có ID là 2,...

Một khuyết điểm nhỏ nữa là việc lưu trữ thông tin của chương trình chưa được tối ưu. Sự rườm rà trong quá trình lưu thông tin của các nhà kho và chi phí di chuyển giữa các nhà kho đã khiến cho chương trình tiêu tốn thêm một lượng tài nguyên nhỏ những không cần thiết.

5.1. Hướng Phát Triển

Với việc áp dụng một cách hiệu quả và triệt để thuật toán Dijkstra, chương trình giúp ta tính toán được đường đi để có chi phí nhỏ nhất một cách chính xác nhất.

Đối với các dữ liệu về những nhà kho đã được lưu một thời gian nhất định, chúng ta có thể sử dụng các phương thức lưu trữ dữ liệu đám mây để tiết kiếm dung lượng.

Ngoài ra, ta có thể phát triển sao cho chương trình tương thức trên nhiều loại thiết bị để có thể trở nên phổ biến, rộng rãi.

Để chương trình được áp dụng vào thực tế và lâu dài hơn, ta cần phải xây dựng thêm một số phương thức như là tìm kiếm theo tên, nhập dữ liệu từ của file excel, hiển thị các loại hàng hóa chứa trong kho,... Về việc tối ưu hóa giao diện để tương tác giữa người dùng và chương trình hiệu quả hơn cũng là một vấn đề quan trọng cần được giải quyết và có thể được giải quyết bằng việc thiết kế giao diện thân thiện hơn.

PHỤ LỤC

- Link GitHub:

Dinopro3007/Team_Test: Bài nhóm KTHP CTDL (github.com)

- Hướng dẫn cách cài đặt để chạy:

- Để chạy được chương trình trên ta cần phải thực hiện các bước sau:
  - Tải phần mềm visual studio code
  - Tạo folder trong phần mềm visual studio code
  - Tạo tài khoản github
  - Truy cập vào đường link github ở trên
  - Tải các file trên github
  - Chuyển các file vừa tải vào folder đã tạo trong visual studio code
  - Sau đó ta có thể bắt đầu chạy chương trình

- Phân công công việc:

Thành Viên	Nhiệm Vụ
Phan Trần Sơn Bảo	Mục 1.3 Các thuật toán trên đồ thị, mục 2.2 Sơ đồ lớp, mục 2.3 Cài đặt lớp, làm Word
Đỗ Xuân Đức	Mục 1.1, 1.2 Các khái niệm,cấu trúc ,cài đặt đồ thị và Chương 3 Trình bày giao diện
Lê Huỳnh Hoàng Khang	Tổng hợp tất cả các file, xây dựng phần cài đặt, phụ lục, chỉnh sửa code, xây dựng github, viết code.
Trương Văn Kiệt	Xây dựng lớp Kho, chỉnh sửa code, chương 4.
Nguyễn Đình Đại Nhơn	Xây dựng lớp đồ thị, viết code áp dụng thuật toán tìm đường đi nhỏ nhất, xây dựng hàm main, các phương thức, chỉnh sửa word.

TÀI LIỆU THAM KHẢO

Dropbox. Tài liệu học tập Code theo buổi Học phần Cơ Sở Lập Trình, Đại học Kinh tế Thành Phố Hồ Chính Minh, 2021