

实验 7 图的经典问题

目录

实验 7 图的经典问题.....	1
一、 问题分析	3
二、 数据结构和算法设计	3
抽象数据类型设计	3
物理数据对象设计	5
算法思想	9
请用题目中样例，基于所设计的算法，详细给出样例求解过程	10
关键功能的算法步骤	11
三、 算法性能分析	12
四、 测试用例	13
五、 实验心得：	15
六、 实验说明：	15

【问题描述】

小明和小芳出去乡村玩，小明负责开车，小芳来导航。

小芳将可能的道路分为大道和小道。大道比较好走，每走 1 公里小明会增加 1 的疲劳度。小道不好走，如果走小道，小明的疲劳值会快速增加，走 s 公里小明会增加 s^2 的疲劳度。所有的小道不相交。

例如：有 5 个路口，1 号路口到 2 号路口为小道，2 号路口到 3 号路口为大道，3 号路口到 4 号路口为大道，4 号路口到 5 号路口为小道，相邻路口之间的距离都是 2 公里。如果小明从 1 号路口到 5 号路口，则总疲劳值为 $2^2+2+2+2^2=4+2+2+4=12$ 。

现在小芳拿到了地图，请帮助她规划一个开车的路线，使得按这个路线开车小明的疲劳度最小。

【输入形式】

输入的第一行包含两个整数 n, m ，分别表示路口的数量和道路的数量。路口由 1 至 n 编号，小明需要开车从 1 号路口到 n 号路口。

接下来 m 行描述道路，每行包含四个整数 t, a, b, c ，表示一条类型为 t ，连接 a 与 b 两个路口，长度为 c 公里的双向道路。其中 t 为 0 表示大道， t 为 1 表示小道。保证 1 号路口和 n 号路口是连通的。

【输出形式】

输出一个整数，表示最优路线下小明的疲劳度。

【样例输入】

```
6 7
1 1 2 3
0 2 3 2
0 1 3 30
0 3 4 20
0 4 5 30
1 3 5 6
0 5 6 1
```

【样例输出】

```
48
```

【数据规模和约定】

对于 25% 的评测用例，不存在小道；

对于所有评测用例， $1 \leq n \leq 8$ ， $1 \leq m \leq 10$ ， $1 \leq a, b \leq n$ ， t 是 0 或 1， $c \leq 100$ 。保证答案不超过 100。

一、问题分析

要求：

分析并确定要处理的对象（数据）是什么。

分析并确定要实现的功能是什么。

分析并确定处理后的结果如何显示。

本题给定 N 个结点和 M 个二元组，二元组 $\langle a, b \rangle$ 代表 a 与 b 之间有一条双向通路。根据题目所给数据，构建一个无向图，并根据题目中 t 的值确定边是大路还是小路，进而确定边权。然后所求问题就可以转换为图的最短路径问题，求出起点到终点的最短路径的长度，然后打印输出，即为最优路线下小明的疲劳度。

二、数据结构和算法设计

抽象数据类型设计

```
template <typename VertexType>
class Graph {
private:
    void operator =(const Graph&) {}          // Protect assignment
    Graph(const Graph&) {}                    // Protect copy constructor

public:
    Graph() {}                                // Default constructor
    virtual ~Graph() {} // Base destructor

    // Initialize a graph of n vertices
    virtual void Init(int n) =0;
```

```

// Return: the number of vertices and edges
virtual int n() =0;
virtual int e() =0;

// Return v's first neighbor
virtual int first(int v) =0;

// Return v's next neighbor
virtual int next(int v, int w) =0;

//设置图的类型（有向图或无向图）
virtual void setType(bool flag)=0;
//获取图的类型
virtual bool getType()=0;
//找到(包含实际信息的)顶点在图中的位置
virtual int locateVex(VertexType u) =0;
//返回某个顶点的值(实际信息)
virtual VertexType getVex(int v)=0;
//给某个顶点赋值
virtual void setVex(int v,VertexType value) =0;

// Set the weight for an edge
virtual void setEdge(int v1, int v2, int wght) =0;

// Delete an edge
// i, j: The vertices
virtual void delEdge(int v1, int v2) =0;

// Determine if an edge is in the graph

```

```

// i, j: The vertices
// Return: true if edge i,j has non-zero weight
virtual bool isEdge(int i, int j) =0;

// Return an edge's weight
// i, j: The vertices
// Return: The weight of edge i,j, or zero
virtual int weight(int v1, int v2) =0;

// Get and Set the mark value for a vertex
// v: The vertex
// val: The value to set
virtual int getMark(int v) =0;
virtual void setMark(int v, int val) =0;
};

```

物理数据对象设计

物理存储结构采用邻接矩阵来实现。

```

#define MAX_VERTEX_NUM 40
#define UNVISITED 0
#define VISITED 1
using namespace std;

template <typename VertexType>
class Graphm : public Graph<VertexType> {
private:
    int numVertex, numEdge; //顶点数和边数
    bool undirected; // true if graph is undirected, false if directed
    VertexType vexs[MAX_VERTEX_NUM]; //存储顶点信息

```

```

int **matrix;           // Pointer to adjacency matrix
int *mark;              // Pointer to mark array
public:
    Graphm(int numVert)    // Constructor
    { Init(numVert); }

    ~Graphm() {           // Destructor
        cout<<"gramat delete";
        delete [] mark; // Return dynamically allocated memory
        for (int i=0; i<numVertex; i++)
            delete [] matrix[i];
        delete [] matrix;
    }

    void Init(int n) { // Initialize the graph
        int i;
        numVertex = n;
        numEdge = 0;
        mark = new int[n];    // Initialize mark array
        for (i=0; i<numVertex; i++)
            mark[i] = UNVISITED;
        matrix = (int**) new int*[numVertex]; // Make matrix
        for (i=0; i<numVertex; i++)
            matrix[i] = new int[numVertex];
        for (i=0; i< numVertex; i++) // Initialize to 0 weights
            for (int j=0; j<numVertex; j++)
                matrix[i][j] = 0;
    }
}

```

```

int n() { return numVertex; } // Number of vertices
int e() { return numEdge; }    // Number of edges

// Return first neighbor of "v"
int first(int v) {
    for (int i=0; i<numVertex; i++)
        if (matrix[v][i] != 0) return i;
    return numVertex;          // Return n if none
}

// Return v's next neighbor after w
int next(int v, int w) {
    for(int i=w+1; i<numVertex; i++)
        if (matrix[v][i] != 0)
            return i;
    return numVertex;          // Return n if none
}

//设置图的类型（有向图或无向图）
void setType(bool flag){
    undirected=flag;
}

//获取图的类型
bool getType(){
    return undirected;
}

/**返回顶点在图中的位置**/
int locateVex(VertexType u){
    for(int i=0;i<numVertex;i++){
        if(Comp(u,vexs[i])) //Comp 模板函数写在 book.h 中
            return i;
    }
}

```

```

    }

    return -1;
}

/**返回某个顶点的值(实际信息) **/
VertexType getVex(int v){
    return vexs[v];
}

/**给某个顶点赋值**/
void setVex(int v,VertexType value){
    vexs[v]=value;
}

// Set edge (v1, v2) to "wt"
void setEdge(int v1, int v2, int wt) {
    Assert(wt>=0, "Illegal weight value");
    if (matrix[v1][v2] == 0)
        numEdge++;
    matrix[v1][v2] = wt;
    if(undirected){
        matrix[v2][v1] = wt;
    }
}

void delEdge(int v1, int v2) { // Delete edge (v1, v2)
    if (matrix[v1][v2] != 0){
        numEdge--;
        matrix[v1][v2] = 0;
        if(undirected){
            matrix[v2][v1] = 0;
        }
    }
}

```



```

    }
}

bool isEdge(int i, int j) // Is (i, j) an edge?
{ return matrix[i][j] != 0; }

int weight(int v1, int v2) { return matrix[v1][v2]; }
int getMark(int v) { return mark[v]; }
void setMark(int v, int val) { mark[v] = val; }
};

```

算法思想

1. 首先根据输入数据构建一个无向图。
2. 再通过 Dijkstra 算法求出各点的最短路径值。
3. 将终点的最短路径长度输出。

关于 Dijkstra 算法

● 算法特点：

迪杰斯特拉算法使用了广度优先搜索解决赋权有向图或者无向图的单源最短路径问题，算法最终得到一个最短路径树。该算法常用于路由算法或者作为其他图算法的一个子模块。

● 算法的思路：

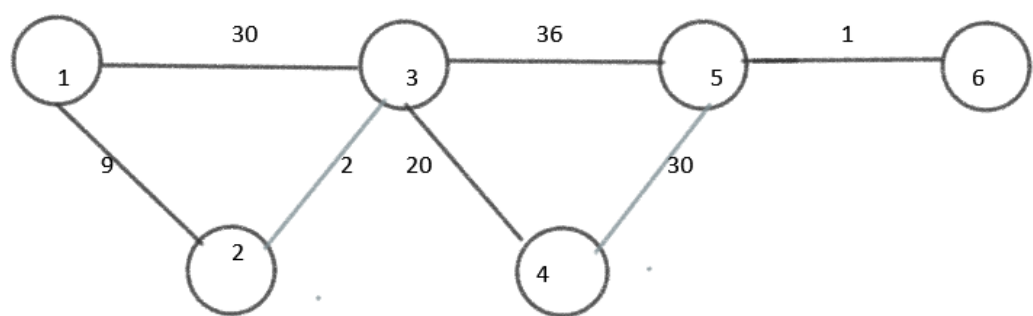
Dijkstra 算法采用的是一种贪心的策略，声明一个数组 `dis` 来保存源点到各个顶点的最短距离和一个保存已经找到了最短路径的顶点的集合 `T`。初始时，原点 `s` 的路径权重被赋为 0 ($dis[s] = 0$)。若对于顶点 `s` 存在能直接到达的边 (s, m) ，则把 `dis[m]` 设为 $w(s, m)$ ，同时把所有其他 (`s` 不能直接到达的) 顶点的路径长度设为无穷大。初始时，集合 `T` 只有顶点 `s`。然后，从 `dis` 数组选择最小值，则该值就是源点 `s` 到该值对应的顶点的最短路径，并且把该点加入到 `T` 中，OK，此时完成一个顶点。

然后，我们需要看看新加入的顶点是否可以到达其他顶点并且看看通过该顶点到达其他点的路径长度是否比源点直接到达短，如果是，那么就替换这些顶点在 `dis` 中的值。

然后，又从 `dis` 中找出最小值，重复上述动作，直到 `T` 中包含了图的所有顶点。

请用题目中样例，基于所设计的算法，详细给出样例求解过程

1. 根据题目数据，构建无向图如下。



2. 无向图的邻接矩阵为

	1	2	3	4	5	6
1	0	9	30	∞	∞	∞
2	9	0	2	∞	∞	∞
3	30	2	0	20	36	∞
4	∞	∞	20	0	30	∞
5	∞	∞	36	30	0	1
6	∞	∞	∞	∞	1	0

3. Dijkstra 算法的处理步骤为

	1	2	3	4	5	6
初始值	0	∞	∞	∞	∞	∞
处理 1	0	9	30	∞	∞	∞
处理 2	0	9	11	∞	∞	∞
处理 3	0	9	11	31	47	∞
处理 4	0	9	11	31	47	∞

处理 5	0	9	11	31	47	48
处理 6	0	9	11	31	47	48

4. 由表可知，源点 1 到终点 6 的最短路径长度为 48。

关键功能的算法步骤

//数据输入

```
for(int i=0;i<m;i++)
```

```
{
```

```
    cin>>t>>a>>b>>c;
```

```
    if(t) G->setEdge(a-1,b-1,c*c); //建立边并确定边权
```

```
    else G->setEdge(a-1,b-1,c);
```

```
}
```

//数据处理

```
int minVertex(Graphm<int>* G,int* D)
```

//返回在未标记的结点中 D 值最小的结点

```
{
```

```
    int i,v=-1;
```

```
    for(i=0;i<G->n();i++)
```

```
    {
```

```
        if(G->getMark(i)==0)
```

```
        {
```

```
            v=i;
```

```
            break;
```

```
        }
```

```
    }
```

```
    for(i++;i<G->n();i++)
```

```
    {
```

```
        if(G->getMark(i)==0&&D[i]<D[v])
```

```
        {
```

```

        v=i;
    }
}
return v;
}

void Dijkstra(Graphm<int>* G,int* D)
{
    int i,v,w;
    for(i=0;i<G->n();i++)
    {
        v=minVertex(G,D); //返回在未标记的结点中 D 值最小的结点
        if(D[v]==INFINITY) return; //不可到达该结点
        G->setMark(v,1); //标记该结点，表示已经找到最短路径
        for(w=G->first(v);w<G->n();w=G->next(v,w))
        {
            if(D[w]>D[v]+G->weight(v,w))
            {
                D[w]=D[v]+G->weight(v,w); //更新邻居结点的 D 值
            }
        }
    }
}

//数据输出
cout<<D[n-1]; //输出起点到终点的最短路径的长度

```

三、算法性能分析

1. 数据输入部分

for 循环共执行 n 次，所以时间复杂度为 $O(n)$ 。

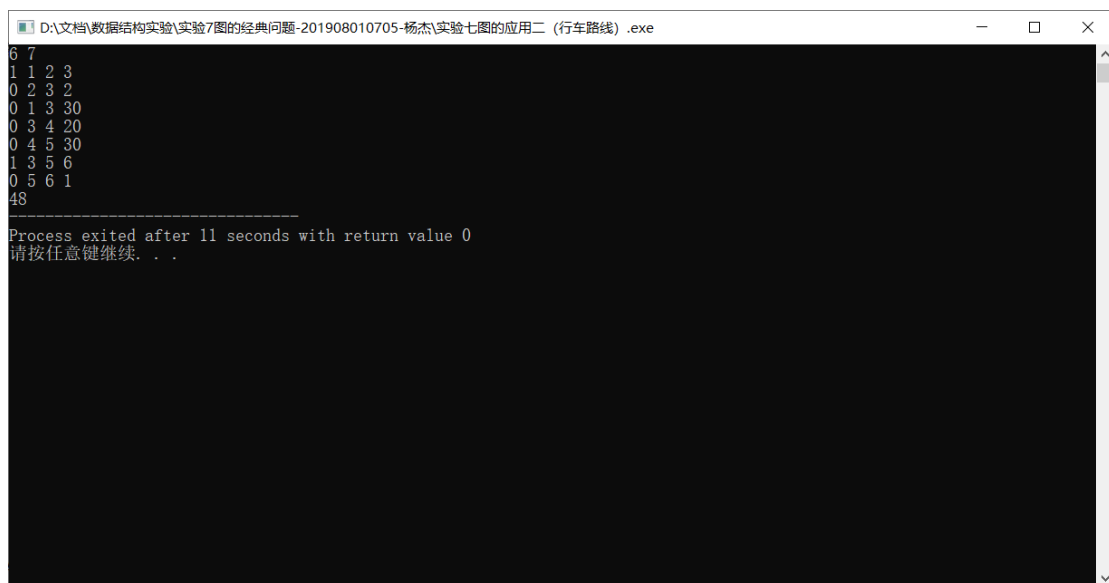
2. 数据处理部分

Dijkstra 算法，外层 for 循环共执行 $|V|$ 次，minVertex 函数中 for 循环共执行 $|V|$ 次，两层循环嵌套，再加上每条边都要遍历一次，所以函数体总的时间复杂度为 $O(|V|^2+|E|)=O(|V|^2)$ ，因为 $|E|$ 在 $O(|V|^2)$ 中。

3. 数据输出部分

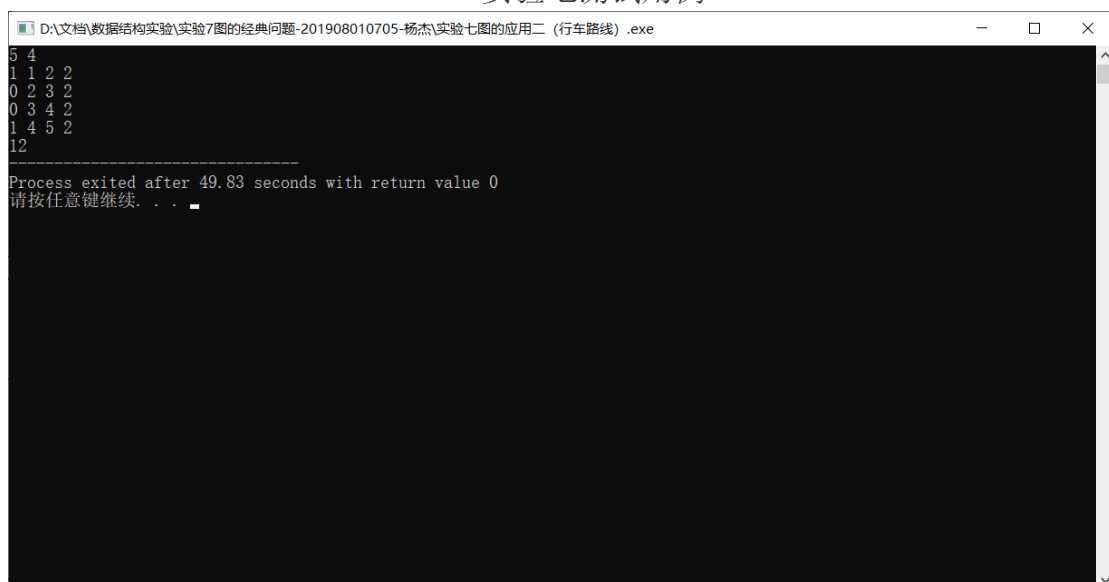
时间复杂度为 $O(1)$ 。

四、测试用例



```
D:\文档\数据结构实验\实验7图的经典问题-201908010705-杨杰\实验七图的应用二 (行车路线) .exe
6 7
1 1 2 3
0 2 3 2
0 1 3 30
0 3 4 20
0 4 5 30
1 3 5 6
0 5 6 1
48
-----
Process exited after 11 seconds with return value 0
请按任意键继续. . .
```

实验七测试用例



```
D:\文档\数据结构实验\实验7图的经典问题-201908010705-杨杰\实验七图的应用二 (行车路线) .exe
5 4
1 1 2 2
0 2 3 2
0 3 4 2
1 4 5 2
12
-----
Process exited after 49.83 seconds with return value 0
请按任意键继续. . .
```

实验七测试用例

```
D:\文档\数据结构实验\实验7图的经典问题-201908010705-杨杰\实验七图的应用二 (行车路线) .exe
7 9
0 1 2 10
0 1 3 10
1 1 4 6
1 2 5 5
0 3 5 9
1 3 6 7
0 4 6 11
0 5 7 15
0 6 7 10
34
-----
Process exited after 1.427 seconds with return value 0
请按任意键继续. . .
```

实验七测试用例

```
D:\文档\数据结构实验\实验7图的经典问题-201908010705-杨杰\实验七图的应用二 (行车路线) .exe
5 5
0 1 2 16
0 2 3 20
1 1 3 6
0 3 4 4
1 4 5 5
65
-----
Process exited after 1.791 seconds with return value 0
请按任意键继续. . .
```

实验七测试用例

```
D:\文档\数据结构实验\实验7图的经典问题-201908010705-杨杰\实验七图的应用二 (行车路线) .exe
8 9
0 1 2 20
0 1 3 10
0 2 4 10
0 3 4 30
0 4 5 20
0 4 6 40
0 5 7 40
0 6 7 35
0 7 8 10
100
-----
Process exited after 1.805 seconds with return value 0
请按任意键继续. . .
```

实验七测试用例

五、实验心得：

通过该实验，我更加深入全面地理解了图这一数据结构，并且能够运用图解决一定的实际问题。通过对实验错误的查找和改正，让我在以后的编程中对指针的使用更加的小心细致。读了书本的理论知识基础，知道如何运用知识来解决实际问题才是关键。同时，图的最短路径问题除了 Dijkstra 算法以外，还有 Floyd 算法、Bellman-Ford 算法等等，这些算法都很巧妙，对编程思维的提高很有帮助。

六、实验说明：

开发语言：C++

开发平台：Dev-C++