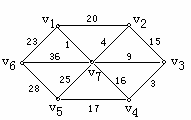
# 《离散数学》课程实验报告4 最小生成树

# 题目简介；

如下图所示的赋权图表示某七个城市，预先计算出它们之间的一些直接通信道路造价（单位：万元），试给出一个设计方案，使得各城市之间既能够保持通信，又使得总造价最小，并计算其最小值。

1. 
2. 图1 七个城市赋权图

# 二.解题思路；：

# 首先，根据题目描述，需要设计一个方案，使得七个城市之间既能够保持通信，又使得总造价最小。这是一个典型的最小生成树（Minimum Spanning Tree, MST）问题。最小生成树是一个连通图中生成树中边的权值之和最小的生成树，它可以保证图中所有节点都连通，并且总权值最小。

# 解题思路如下：

# 1. 首先，根据给定的城市之间的直接通信道路造价，可以将这些数据构建成一个带权的无向连通图。

# 2. 採用普里姆（Prim）算法或者克鲁斯卡尔（Kruskal）算法来求解最小生成树。这两种算法都可以找到连通图中的最小生成树，其中普里姆算法适合于稠密图，而克鲁斯卡尔算法适合于稀疏图。

# 3. 通过选择合适的算法，计算得到最小生成树，并计算其总造价。

# 具体步骤如下：

# 1. 构建带权的无向连通图，其中城市为图的节点，通信道路造价为边的权值。

# 2. 选择普里姆算法或者克鲁斯卡尔算法来求解最小生成树。

# 3. 找到最小生成树后，计算其所有边的权值之和，即为总造价的最小值。

# 通过以上步骤，就可以设计出一个方案，使得各城市之间既能够保持通信，又使得总造价最小，并计算出其最小值。

# 三.数据结构；

1. 数据结构：

- `p[N]`: 用于存储父节点，表示最小生成树中每个节点的父节点。

- `key[N]`: 用于存储节点的关键值，表示当前节点到最小生成树的距离。

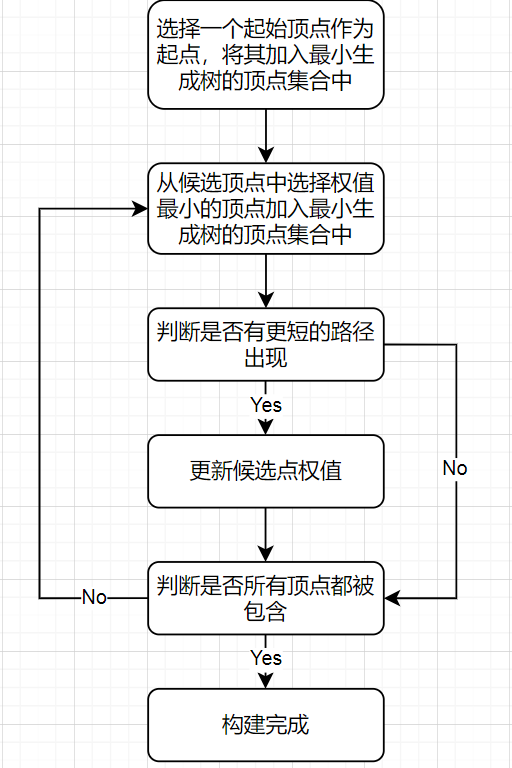
- `tb[N][N]`: 二维数组，用于存储图的邻接矩阵，表示节点之间的边权值。

2. 算法：

- Prim算法：用于求解带权无向图的最小生成树。Prim算法是一种贪心算法，从一个初始节点开始，逐步扩展最小生成树的节点，直到包含所有节点。在每一步中，选择与当前最小生成树相连的节点中距离最小的节点加入最小生成树。

具体来说，代码中的`prim`函数是用Prim算法来计算最小生成树的权值，`p`数组和`key`数组分别用于存储节点的父节点和关键值，`tb`数组用于存储图的邻接矩阵。

在`main`函数中，通过输入顶点数目和边的数目，以及输入每条边的节点序号和权值，然后调用`prim`函数计算最小生成树的权值。这段代码的目的是实现Prim算法，求解带权无向图的最小生成树，并输出最小生成树的权值。



# 四. 实验原理和实现过程（算法描述）；

## 1.实验原理

## Prim算法基本原理如下：

## 1. 初始化：选择一个起始顶点作为起点，将其加入最小生成树的顶点集合中。然后初始化与其相邻的顶点的权值（或距离），这些顶点作为候选的下一个加入最小生成树的顶点。

## 2. 循环操作：重复以下操作，直到最小生成树包含了图中的所有顶点。

## - 从候选顶点中选择权值最小的顶点加入最小生成树的顶点集合中。

## - 更新候选顶点的权值（或距离），如果有更短的路径出现。

## 3.最小生成树构建：当最小生成树的顶点集合包含了图中的所有顶点时，算法结束。此时最小生成树就构建完成了。

## Prim算法的主要特点是每次选择与当前最小生成树连接的权值最小的顶点，以逐步构建最小生成树。这种贪心策略保证了每一步都是最优的，最终得到的最小生成树具有全局最优解。

## 总的来说，Prim算法通过不断选择与当前最小生成树连接的权值最小的顶点来构建最小生成树，其实现原理能够保证得到最小生成树的权值是最小的。

## 2.实验过程

## - 首先选择一个起始顶点，将其加入最小生成树的顶点集合中，并将其关联边的权值初始化为0。

## - 从候选顶点中选择与当前最小生成树连接的权值最小的顶点，将其加入最小生成树的顶点集合中。

## - 更新其他候选顶点的权值，如果通过新加入的顶点可以获得更小的权值，则更新这些候选顶点的权值。

## - 重复上述步骤，直到最小生成树的顶点集合包含了图中的所有顶点。

# 五.部分核心代码；

#include <iostream>

#include <climits>

using namespace std;

#define N 100

int p[N], key[N], tb[N][N]; // 定义存储父节点、关键值和图的邻接矩阵的数组

int prim(int v, int n) // Prim算法实现

{

int minValue = 0; // 最小生成树的总权值

int i, j;

int min;

for (i = 1; i <= n; i++) // 初始化

{

p[i] = v; // 将所有节点的父节点初始化为v

key[i] = tb[v][i]; // 将所有节点的关键值初始化为与v相连的边的权值

}

key[v] = 0; // 将起始节点的关键值设为0，表示已加入最小生成树

for (i = 2; i <= n; i++) // 逐步构建最小生成树

{

min = INT\_MAX; // 初始化最小值为最大整数

for (j = 1; j <= n; j++)

{

if (key[j] > 0 && key[j] < min) // 找到关键值最小的节点

{

v = j; // 更新当前节点为关键值最小的节点

min = key[j]; // 更新最小值

}

}

cout << "最小耗费是:" << p[v] << "和" << v << " " << endl; // 输出加入最小生成树的边

minValue += tb[v][p[v]]; // 更新最小生成树的总权值

key[v] = 0; // 将当前节点加入最小生成树

for (j = 1; j <= n; j++)

{

if (tb[v][j] < key[j]) // 更新与当前节点相邻的节点的关键值

{

p[j] = v; // 更新父节点

key[j] = tb[v][j]; // 更新关键值

}

}

}

return minValue; // 返回最小生成树的总权值

}

int main()

{

int n, m;

int i, j;

int u, v, w;

cout << "请输入所求图的顶点数目和边的数目(以空格分隔各个数，输入两个0结束):\n"; //输入所求的顶点数目和边数

while (cin >> n >> m)

{

if (m != 0 && n != 0)

{

for (i = 1; i <= n; i++)

{

for (j = 1; j <= n; j++)

{

tb[i][j] = INT\_MAX; // 初始化邻接矩阵

}

}

cout << "请输入两条边的节点序号以及它们的权值(以空格分隔各个数):\n";

while (m--)

{

cin >> u >> v >> w; //输入边数以及它们的权值

tb[u][v] = tb[v][u] = w; // 更新邻接矩阵

}

cout << "最小生成树权值为" << prim(1, n) << endl; // 输出最小生成树的总权值

cout << "请输入所求图的顶点数目和边的数目(以空格分隔各个数，输入两个0结束):\n"; //输入所求的顶点数目和边数

}

else if (n == 0 && m == 0)

return 0; // 结束程序

}

return 0;

}