

项目说明文档

数据结构课程设计

——电网建设造价模拟系统

作者姓名：_____沈星宇_____

学 号：_____1951576_____

指导教师：_____张颖_____

学院、专业：_____软件学院 软件工程_____

同济大学

Tongji University

一、 分析

(1) 应用背景

假设一个城市有 n 个小区，要实现 n 个小区之间的电网都能够相互接通，构造这个城市 n 个小区之间的电网，使总工程造价最低。请设计一个能够满足要求的造价方案。

(2) 项目功能要求

在每个小区之间都可以设置一条电网线路，都要付出相应的经济代价。 n 个小区之间最多可以有 $n(n-1)/2$ 条线路，选择其中的 $n-1$ 条使总的耗费最少。

二、 设计

(1) 数据结构设计

```
class Graph{
    friend class MinSpanTree;
private:
    Node* NodeTable;
    bool* Vmst;
    int nodeNum;
    int edgeNum;
public:
    Graph() { ... }
    ~Graph() { ... }
    void setGraph() { ... }
    char* getName(int i) { ... }
    Node* getNodeTable() { ... }
    int getWeight(int v1, int v2) { ... }
    bool insertNode(const Node& newNode) { ... }
    bool insertEdge(int v1, int v2, int cost) { ... }
    int getFirstNeighbor(int v) { ... }
    int getNextNeighbor(int v, int w) { ... }
    bool isValid(int v) { ... }
    int findID(char* name) { ... }
    void edgeNumPlus() { ... }
    void prim(Graph& graph, char* u0, MinSpanTree& MST) { ... }
};

class Edge{
private:
    int dest;
    int cost;
    Edge* link;
public:
    friend class Graph;
    Edge() { ... }
    Edge(int dest, int cost, Edge* edge) { ... }
};

class Node{
private:
    char data[20];
    Edge* adj;
public:
    friend class Graph;
    Node() { ... }
    Node(char* data, Edge* adj) { ... }
    char* getData() { ... }
};
```

- 1、Graph 是图，本项目是通过构建无向有权图来实现的，存储方式是邻接表，故 Graph 的成员变量有一个 NodeTable，用来存储所有的结点。
- 2、Node 是图的结点，内部的成员变量 data 用于存储结点的名称，Edge* adj 指向后面的 Edge。
- 3、Edge 是图的边结构，内部的成员变量 dest 是该边的下一结点序号（因为 Node 当中存储了第一个结点），cost 是该边的权值大小，Edge*link 链接了下一个边结构。

```

class MinHeap {
private:
    MSTEdgeNode* heap;
    int MaxHeapSize;
    int CurrentSize;
    void FilterDown(int start, int endOfHeap) { ... }
    void FilterUp(int start) { ... }
public:
    MinHeap() { ... }
    MinHeap(int size) { ... }
    MinHeap(MSTEdgeNode arr[], int size) { ... }
    ~MinHeap() { ... }
    bool Insert(const MSTEdgeNode& x) { ... }
    bool RemoveMin(MSTEdgeNode& x) { ... }
    int isEmpty()const { ... }
    int isFull()const { ... }
    void MakeEmpty() { ... }
};

class MinSpanTree {
    friend class Graph;
private:
    MSTEdgeNode* edgeValue;
    int maxSize;
    int curSize;
public:
    MinSpanTree() { ... }
    ~MinSpanTree() { ... }
    void setMST(int nodeNum) { ... }
    bool insert(MSTEdgeNode& item) { ... }
    MSTEdgeNode* getNode() { ... }
};

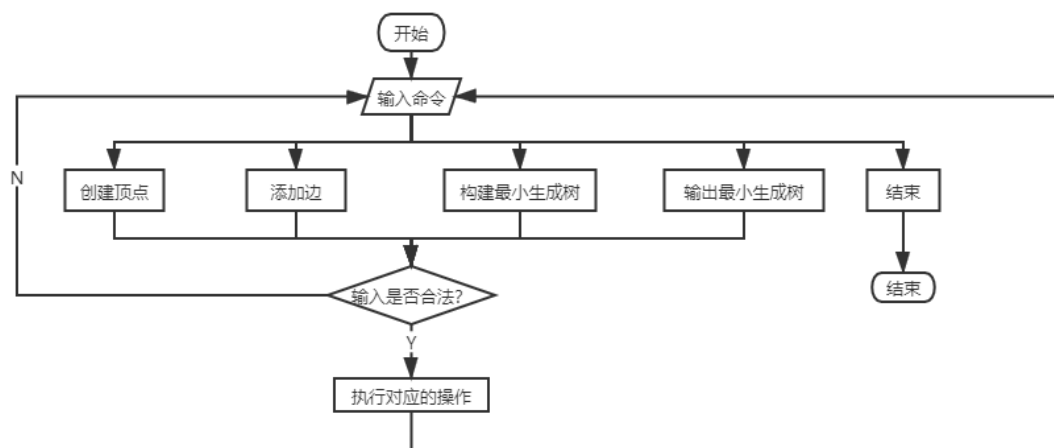
class MSTEdgeNode {
    friend class Graph;
    friend class MinHeap;
private:
    int tail, head;
    int key;
public:
    friend class MinSpanTree;
    MSTEdgeNode() { ... }
    MSTEdgeNode(int tail, int head, int key) { ... }
    int getId() { ... }
    int getKey() { ... }
    int getTail() { ... }
};

```

- 4、MinHeap 是最小堆，本项目通过最小堆排序来取出最小值。其中函数 `void FilterDown(int start, int endOfHeap)`; 是用于自上而下将其调整为最小堆; `void FilterUp(int i)`; 是将插入新节点后的结构从下往上调整为最小堆。函数 `bool Insert(const int& x)`; 是往树里插入新的节点，而函数 `bool RemoveMin(int& x)`; 是将根节点的最小值取出。
- 5、MinSpanTree 是最小生成树结构，其实就是一个单链表，用于存储最小生成树的结果。
- 6、MSTEdgeNode 是最小生成树的结点，head 和 tail 分别代表该边的头尾结点序号，key 代表边的权值。

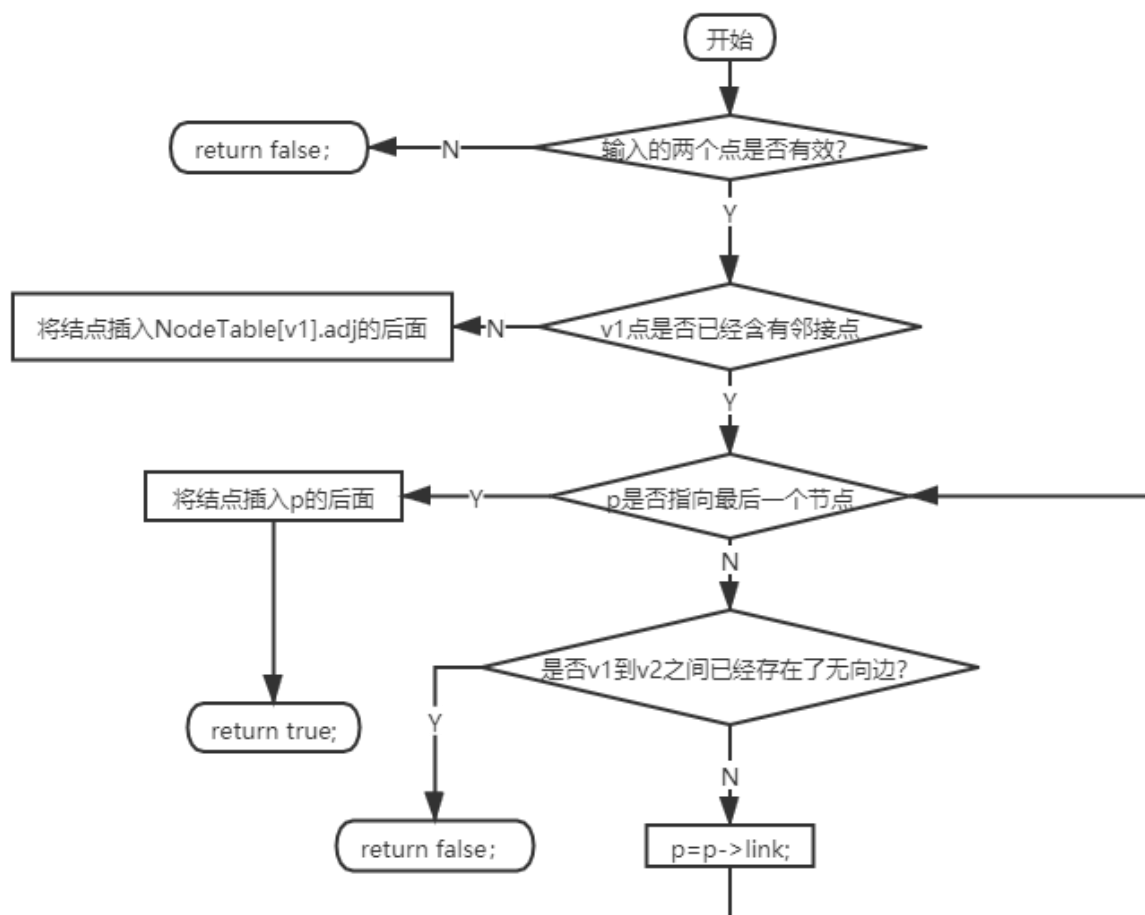
(2) 程序流程设计

- 1、首先程序会提示输入指令，会对指令的合法性进行判断。
- 2、合法性判断包括：
 - (1) 除 A/a/B/b/C/c/D/d/E/e 之外的所有指令均为非法，会被要求重新输入
 - (2) 本项目的执行其实是有顺序的，比如以创建顶点、添加边、构建最小生成树、输出最小生成树的顺序进行，所以如果先执行了后续的操作命令，则会要求重新输入
- 3、分别执行相应的操作



三、 实现

1、 Graph::insertEdge()的实现



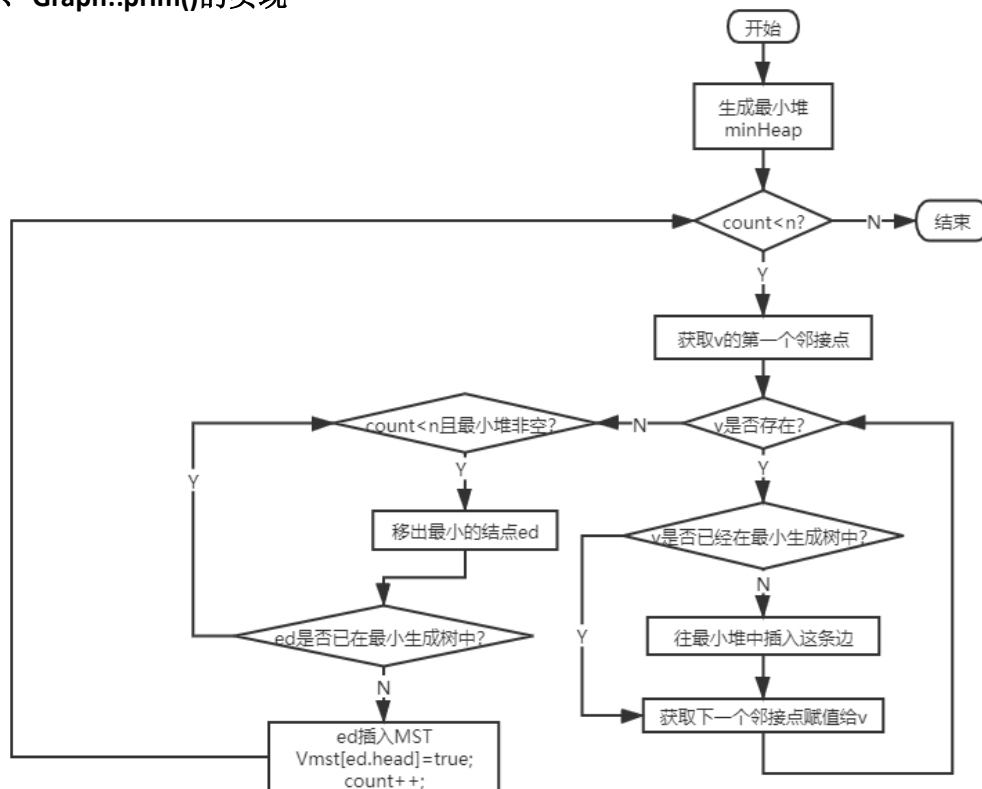
```

bool insertEdge(int v1,int v2, int cost) {
    if (isValid(v1) && isValid(v2)) {
        Edge* p = NodeTable[v1].adj;
        Edge* newV1 = (Edge*)malloc(sizeof(Edge));
        if (newV1 != NULL) {
            newV1->cost = cost;
            newV1->dest = v2;
            newV1->link = NULL;
        }
        //-----
        if (p == NULL) {
            NodeTable[v1].adj = newV1;
        }
        else {
            while (p->link != NULL) {
                if (p->dest == v2) {
                    std::cout << v1 << "到" << v2 << "之间的边已经存在!" << std::endl;
                    free(newV1);
                    return false;
                }
                p = p->link;
            }
            p->link = newV1;
        }
        return true;
    }
    else {
        std::cout << "越界! 不存在这样的点!" << std::endl;
        return false;
    }
}

```

- (1) 首先会判定插入的边的端点是否存在，防止往不存在的顶点中进行插入
- (2) 然后判断 v1 对应的点是否已经有邻接点，如果没有则直接插入在 NodeTable[v1].adj 的后面；如果已有，则找该链的最后进行插入

2、Graph::prim()的实现



```

void prim(Graph& graph, char* u0, MinSpanTree& MST) {
    MSTEdgeNode ed;
    int v, count;
    int n = nodeNum;
    int m = edgeNum;
    int u = findID(u0);
    MinHeap minHeap(edgeNum);
    Vmst = new bool[n];
    for (int i = 0; i < n; i++) {
        Vmst[i] = false;
    }
    Vmst[u] = true;
    count = 1;
    do {
        v = graph.getFirstNeighbor(u);
        while (v != -1) {
            if (Vmst[v] == false) {
                ed.tail = u;
                ed.head = v;
                ed.key = graph.getWeight(u, v);
                minHeap.Insert(ed);
            }
            v = graph.getNextNeighbor(u, v);
        }
        while (!minHeap.isEmpty() && count < n) {
            minHeap.RemoveMin(ed);
            if (!Vmst[ed.head]) {
                MST.insert(ed);
                u = ed.head;
                Vmst[u] = true;
                count++;
                break;
            }
        }
    } while (count < n);
}

```

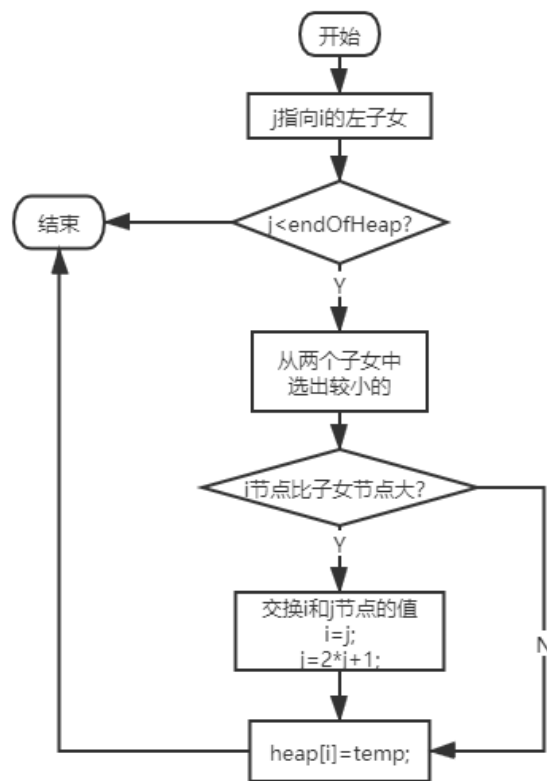
- (1) 图顶点集合为 U ，对任意选择的起始点 u_0 ，将该点加入集合 V ，再从集合 $U - V$ 中找到另一点 b 使得点 b 到 V 中任意一点的权值最小，将 b 点也加入集合 V ；以此类推，直至所有顶点全部被加入 V ，此时就构建出了一棵 MST。
- (2) 此处函数借用最小堆来寻找权值最小的边，用数组 $Vmst[]$ 来记录是否将边放入了最小生成树。

3、MinHeap::FilterDown()的实现

```

void MinHeap::FilterDown(int start, int endOfHeap) {
    int i = start, j = 2 * i + 1;
    int temp = heap[i];
    while (j <= endOfHeap) {
        //两子女中选小的
        if (j < endOfHeap && heap[j] > heap[j + 1]) {
            j++;
        }
        if (temp <= heap[j]) {
            break;
        }
        else {
            heap[i] = heap[j];
            i = j;
            j = 2 * j + 1;
        }
    }
    heap[i] = temp;
}

```



- (1) 于一个 i 来说, $j=2*i+1$ 就是 i 这个节点的左子女 (由堆的性质可得)
- (2) 当 j 到达堆的最后一个节点时结束循环
- (3) 从 i 的左右子女 (如果由右子女的话) 当中选出最小的那个, 与 i 节点的值进行比较, 如果 i 节点已经是最小的了, 则结束循环; 如果 i 的子女节点比 i 节点更小, 那么交换两个节点的值, 同时继续向下搜索 (即 i 指向 j , j 指向下一个左子女)

4、MinHeap::FilterUp()的实现

```

void MinHeap::FilterUp(int start) {
    int j = start, i = (j - 1) / 2;
    int temp = heap[j];
    while (j > 0) {
        if (heap[i] <= temp) {
            break;
        }
        else {
            heap[j] = heap[i];
            j = i;
            i = (i - 1) / 2;
        }
        heap[j] = temp;
    }
}
  
```

- (1) 本函数是针对新加入的节点，从下往上调整它的位置
- (2) j 指向当前节点，i 是 j 的父母节点，循环判断新的节点的值与其父母节点的大小关系，如果新的节点比其父母节点更小，则往上调整其位置；其他的情况则可以退出循环。
- (3) 基本上就是 MinHeap::FilterDown()的逆过程

5、MinHeap::Insert()的实现

- (1) 将插入的值放在数组的最末端，然后调用 FilterUp()函数将其调整为最小堆，同时调整最小堆的大小。

```
bool MinHeap::Insert(const int& x) {  
    if (CurrentSize == MaxHeapSize) {  
        std::cout << "堆已满" << std::endl;  
        return false;  
    }  
    heap[CurrentSize] = x;  
    FilterUp(CurrentSize);  
    CurrentSize++;  
    return true;  
}
```

6、MinHeap::RemoveMin()的实现


- (1) 移除根节点最小的那个数字，通过引用 x 来将其值传出
- (2) 同时对于 heap[]数组来说，要做的就是将最前面的数字给放到最后，同时让 CurrentSize--，确保该值不会再被访问。
- (3) 调用 FilterDown()函数，调整其重新成为最小堆。

```
bool MinHeap::RemoveMin(int& x) {  
    if (!CurrentSize) {  
        std::cout << "堆已空" << std::endl;  
        return false;  
    }  
    x = heap[0];  
    heap[0] = heap[CurrentSize - 1];  
    CurrentSize--;  
    FilterDown(0, CurrentSize - 1);  
    return true;  
}
```


四、 测试

1、合法性检测

- (1) 输入的指令不合法（即超出 A/a/B/b/C/c/D/d/E/e）,会要求重新输入。
- (2) 本项目的执行其实是有顺序的，比如以创建顶点、添加边、构建最小生成树、输出最小生成树的顺序进行，所以如果先执行了后续的操作命令，则会要求重新输入
- (3) 如果输入的顶点个数/权值不为正整数，则会要求重新输入
- (4) 如果输入的点不再列表中，则会要求重新输入

 选择Microsoft Visual Studio 调试控制台

```
=====
**          请选择要执行的操作：          **
**          A --- 创建电网顶点              **
**          B --- 添加电网的边              **
**          C --- 构造最小生成树            **
**          D --- 显示最小生成树            **
**          E --- 退出程序                  **
=====

请选择操作:q
找不到该命令，请重新输入！

请选择操作:B
请先进入A创建电网操作！

请选择操作:D
请先完成A和B的操作！

请选择操作:A
请输入顶点的个数: a
输入不合法，请输入正整数！

请选择操作:A
请输入顶点的个数: -1
输入不合法，请输入正整数！

请选择操作:A
请输入顶点的个数: 4
请依次输入各顶点的名称: a b c d

请选择操作:b
请输入两个顶点及边: e f 1
查找不到该顶点，请重新输入
```

2、一般情况的功能实现

```

**          电网造价模拟系统          **
=====
**          请选择要执行的操作:          **
**          A  ---  创建电网顶点          **
**          B  ---  添加电网的边          **
**          C  ---  构造最小生成树        **
**          D  ---  显示最小生成树        **
**          E  ---  退出程序              **
=====

请选择操作:A
请输入顶点的个数: 4
请依次输入各顶点的名称: a b c d

请选择操作:B
请输入两个顶点及边: a b 8
请输入两个顶点及边: b c 7
请输入两个顶点及边: c d 5
请输入两个顶点及边: d a 11
请输入两个顶点及边: a c 18
请输入两个顶点及边: b d 12
请输入两个顶点及边: exit

请选择操作:C
请输入起始顶点: a
生成prim最小生成树!

请选择操作:D
最小生成树的顶点及边为:
b-<8>->a
c-<7>->b
d-<5>->c

请选择操作:E

```