



南京邮电大学
Nanjing University of Posts and Telecommunications

《算法设计与分析》课内实验报告

实验题目：_____贪心法_____

实验日期：_____2023 年 11 月 15 日_____

指导教师：_____张博_____

学生姓名：_____单家俊_____

学生学号：_____B21080526_____

实验成绩：_____

一、实验目的

本实验旨在通过贪心法实现最小代价生成树问题的求解，深入理解贪心算法在图论中的应用，以及掌握最小生成树的相关概念和算法。

二、实验内容

用贪心法实现最小代价生成树问题的求解。

三、实验过程描述

算法思想：

1. 贪心法

贪心法是一种通过每一步的局部最优选择来达到全局最优的算法思想。在最小生成树问题中，贪心法的基本思想是从图中选择一条边，将其添加到生成树中，然后选择下一条权重最小的边，直到生成树包含了所有的顶点为止。贪心法的关键在于每一步都选择当前局部最优的解，以期望最终得到全局最优解。

2. 最小代价生成树

最小代价生成树是一个连通无向图的生成树，其边的权重之和最小。典型的贪心算法解决该问题的方式是通过 Kruskal 算法或 Prim 算法。

- Kruskal 算法：按照边的权重从小到大的顺序选择边，加入生成树中，保证加入的边不构成环，直到生成树包含所有顶点。
- Prim 算法：从一个初始顶点开始，选择与当前生成树相邻的权重最小的边，将其加入生成树，然后选择新加入的顶点的相邻边中权重最小的边，依次类推，直到生成树包含所有顶点。

计算时间复杂度：

1. Kruskal 算法时间复杂度分析

设图的顶点数为 V ，边数为 E 。

- 排序所有边的时间复杂度： $O(E \log E)$
- 并查集操作的时间复杂度： $O(E \log V)$

因此，Kruskal 算法的总时间复杂度为 $O(E \log E + E \log V)$ 。

2. Prim 算法时间复杂度分析

设图的顶点数为 V ，边数为 E 。

- 优先队列插入和删除的时间复杂度： $O(E \log V)$
- Prim 算法的总时间复杂度为 $O(E \log V)$ 。

代码：

Kruskal 算法:

```
#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_VERTEX 10
#define MAX_EDGE 100
typedef int DataType;

typedef struct EdgeType {
    int from, to;
    int weight;
} EdgeType;

struct EdgeGraph {
    DataType* vertex;
    EdgeType* edge;
    int vertexNum, edgeNum;
};

void Graph(struct EdgeGraph* graph, int vertexNum, int arcNum);
void ranklist(struct EdgeGraph* graph);
void Kruskal(struct EdgeGraph graph, int* parent);
int findRoot(int* parent, int v);
void outputMST(EdgeType x);
void printGraph(struct EdgeGraph graph);

int main() {
    int vertexNum, arcNum;
    printf("请输入顶点个数和边的个数:");
    scanf("%d %d", &vertexNum, &arcNum);
    struct EdgeGraph graph;
    Graph(&graph, vertexNum, arcNum);
    ranklist(&graph);
    printf("输出通过边信息储存的图:\n");
    printGraph(graph);

    // 动态分配内存
    int* parent = (int*)malloc(vertexNum * sizeof(int));
    if (parent == NULL) {
        fprintf(stderr, "内存分配失败\n");
        return -1;
    }
}
```

```

printf("输出Kruskal算法得的最小生成树(from, to)weight:\n");
Kruskal(graph, parent);

// 释放动态分配的内存
free(parent);

return 0;
}

void Graph(struct EdgeGraph* graph, int vertexNum, int arcNum) {
    graph->edgeNum = arcNum;
    graph->vertexNum = vertexNum;

    // 动态分配内存
    graph->vertex = (DataType*)malloc(vertexNum * sizeof(DataType));
    graph->edge = (EdgeType*)malloc(arcNum * sizeof(EdgeType));

    if (graph->vertex == NULL || graph->edge == NULL) {
        fprintf(stderr, "内存分配失败\n");

        // 释放动态分配的内存
        free(graph->vertex);
        free(graph->edge);

        exit(-1);
    }

    printf("请逐个输入顶点的内容：");
    DataType x;
    for (int i = 0; i < vertexNum; i++) {
        scanf("%d", &x);
        graph->vertex[i] = x;
    }
    int count = 1;
    int a, b, ave;
    for (int i = 0; i < arcNum; i++) {
        printf("请输入第%d条边依附的两个顶点的编号和权值：", count++);
        scanf("%d %d %d", &a, &b, &ave);
        graph->edge[i].from = a;
        graph->edge[i].to = b;
        graph->edge[i].weight = ave;
    }
}

```

```

void ranklist(struct EdgeGraph* graph) {
    EdgeType temp;
    for (int i = 0; i < graph->edgeNum; i++) {
        for (int j = 0; j < graph->edgeNum - i - 1; j++) {
            if (graph->edge[j].weight > graph->edge[j + 1].weight) {
                temp = graph->edge[j];
                graph->edge[j] = graph->edge[j + 1];
                graph->edge[j + 1] = temp;
            }
        }
    }
}

```

```

void Kruskal(struct EdgeGraph graph, int* parent) {
    for (int i = 0; i < graph.vertexNum; i++)
        parent[i] = -1;
    int num, i, vex1, vex2;
    for (num = 0, i = 0; i < graph.edgeNum; i++) {
        vex1 = findRoot(parent, graph.edge[i].from);
        vex2 = findRoot(parent, graph.edge[i].to);
        if (vex1 != vex2) {
            outputMST(graph.edge[i]);
            parent[vex2] = vex1;
            num++;
            if (num == graph.vertexNum - 1)
                return;
        }
    }
}

```

```

int findRoot(int* parent, int v) {
    int t = v;
    while (parent[t] > -1)
        t = parent[t];
    return t;
}

```

```

void outputMST(EdgeType x) {
    printf("( %d, %d) %d\n", x.from, x.to, x.weight);
}

```

```

void printGraph(struct EdgeGraph graph) {
    printf("    no    from    to    weight\n");
}

```

```

        for (int i = 0; i < graph.edgeNum; i++) {
            printf("    edge[%d]    %d    %d    %d\n", i, graph.edge[i].from,
graph.edge[i].to, graph.edge[i].weight);
        }
    }
}

```

Prim 算法:

```

#define _CRT_SECURE_NO_WARNINGS 1
#include <stdio.h>
#include <limits.h>

#define MAX_VERTEX 10
typedef int DataType;

typedef struct PrimNode {
    int adjvex;
    int lowcost;
} PrimNode;

void MGraph(DataType* vertex, int arc[][MAX_VERTEX], int vertexNum, int
arcNum);
void Prim(int arc[][MAX_VERTEX], int vertexNum, int start, PrimNode*
shortEdge);
void outputSMT(int k, PrimNode* shortEdge);
int minEdge(PrimNode* shortEdge, int vertexNum);
void printMGraph(DataType* vertex, int arc[][MAX_VERTEX], int vertexNum);

int main() {
    DataType vertex[MAX_VERTEX];
    int arc[MAX_VERTEX][MAX_VERTEX];
    int vertexNum, arcNum;

    printf("请输入顶点个数和边的个数:");
    scanf("%d %d", &vertexNum, &arcNum);

    MGraph(vertex, arc, vertexNum, arcNum);

    printf("输出邻接矩阵信息:\n");
    printMGraph(vertex, arc, vertexNum);

    int x;
    printf("请输入Prim算法的起点:");
}

```

```

scanf("%d", &x);

PrimNode shortEdge[MAX_VERTEX];
printf("输出起点从编号%d开始的最小生成树:\n", x);
Prim(arc, vertexNum, x, shortEdge);

return 0;
}

void MGraph(DataType* vertex, int arc[][MAX_VERTEX], int vertexNum, int
arcNum) {
    printf("请逐个输入顶点的内容: ");
    DataType x;
    DataType vi, vj, ave;

    for (int i = 0; i < vertexNum; i++) {
        scanf("%d", &x);
        vertex[i] = x;
    }

    for (int i = 0; i < vertexNum; i++)
        for (int j = 0; j < vertexNum; j++)
            arc[i][j] = INT_MAX;

    int count = 1;
    for (int i = 0; i < arcNum; i++) {
        printf("请输入第%d条边依附的两个顶点的编号和权值: ", count++);
        scanf("%d %d %d", &vi, &vj, &ave);
        arc[vi][vj] = ave;
        arc[vj][vi] = ave;
    }
}

void Prim(int arc[][MAX_VERTEX], int vertexNum, int start, PrimNode*
shortEdge) {
    int k;
    for (int i = 0; i < vertexNum; i++) {
        shortEdge[i].lowcost = arc[start][i];
        shortEdge[i].adjvex = start;
    }

    shortEdge[start].lowcost = 0;

    for (int i = 0; i < vertexNum - 1; i++) {

```

```

        k = minEdge(shortEdge, vertexNum);
        outputSMT(k, shortEdge);
        shortEdge[k].lowcost = 0;

        for (int j = 0; j < vertexNum; j++) {
            if (arc[k][j] < shortEdge[j].lowcost) {
                shortEdge[j].lowcost = arc[k][j];
                shortEdge[j].adjvex = k;
            }
        }
    }
}

void outputSMT(int k, PrimNode* shortEdge) {
    printf("( %d, %d) %d\n", shortEdge[k].adjvex, k, shortEdge[k].lowcost);
}

int minEdge(PrimNode* shortEdge, int vertexNum) {
    int min, flag = 0, index;
    for (int i = 0; i < vertexNum; i++) {
        if (shortEdge[i].lowcost != 0) {
            if (flag == 0) {
                min = shortEdge[i].lowcost;
                index = i;
                flag = 1;
            }
            else if (min > shortEdge[i].lowcost) {
                min = shortEdge[i].lowcost;
                index = i;
            }
        }
    }
    return index;
}

void printMGraph(DataType* vertex, int arc[][MAX_VERTEX], int vertexNum) {
    printf("vertex:");
    for (int i = 0; i < vertexNum; i++) {
        printf("%d ", vertex[i]);
    }
    printf("\n");

    printf("arc:\n");
    for (int i = 0; i < vertexNum; i++) {

```



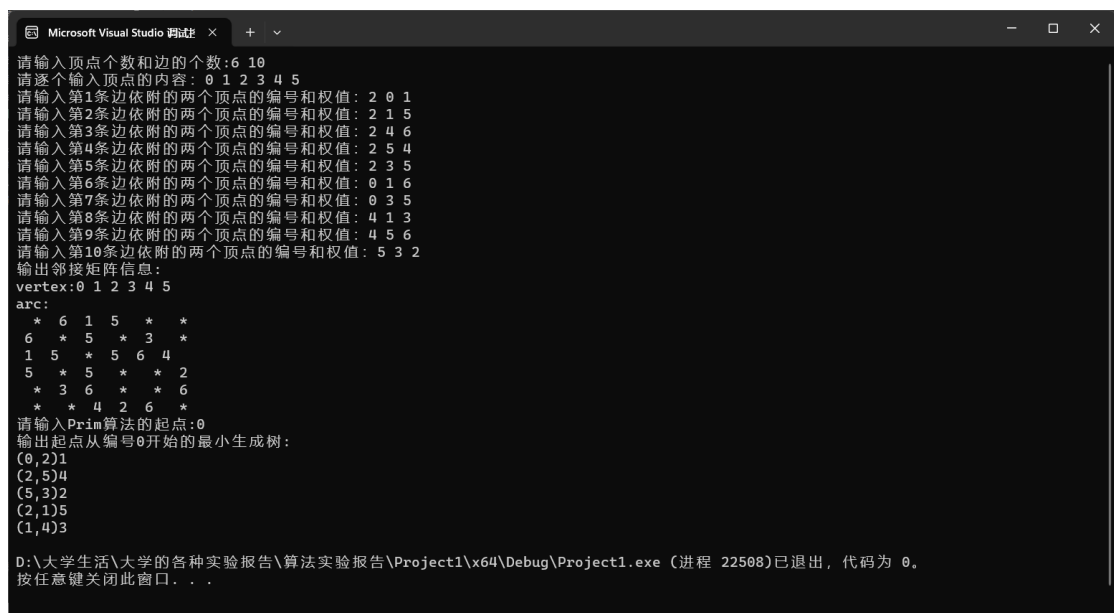
```

        for (int j = 0; j < vertexNum; j++) {
            if (j == vertexNum - 1) {
                if (arc[i][j] == INT_MAX)
                    printf(" *\\n");
                else
                    printf(" %d\\n", arc[i][j]);
            }
            else {
                if (arc[i][j] == INT_MAX)
                    printf(" * ");
                else
                    printf(" %d ", arc[i][j]);
            }
        }
    }
}

```

四、实验结果

Prim 算法:



```

Microsoft Visual Studio 调试
请输入顶点个数和边的个数:6 10
请逐个输入顶点的内容: 0 1 2 3 4 5
请输入第1条边依附的两个顶点的编号和权值: 2 0 1
请输入第2条边依附的两个顶点的编号和权值: 2 1 5
请输入第3条边依附的两个顶点的编号和权值: 2 4 6
请输入第4条边依附的两个顶点的编号和权值: 2 5 4
请输入第5条边依附的两个顶点的编号和权值: 2 3 5
请输入第6条边依附的两个顶点的编号和权值: 0 1 6
请输入第7条边依附的两个顶点的编号和权值: 0 3 5
请输入第8条边依附的两个顶点的编号和权值: 4 1 3
请输入第9条边依附的两个顶点的编号和权值: 4 5 6
请输入第10条边依附的两个顶点的编号和权值: 5 3 2
输出邻接矩阵信息:
vertex:0 1 2 3 4 5
arc:
* 6 1 5 * *
6 * 5 * 3 *
1 5 * 5 6 4
5 * 5 * * 2
* 3 6 * * 6
* * 4 2 6 *
请输入Prim算法的起点:0
输出起点从编号0开始的最小生成树:
(0,2)1
(2,5)4
(5,3)2
(2,1)5
(1,4)3

D:\大学生活\大学的各种实验报告\算法实验报告\Project1\x64\Debug\Project1.exe (进程 22508)已退出, 代码为 0。
按任意键关闭此窗口...

```

Kruskal 算法:

```
Microsoft Visual Studio 调试
请输入顶点个数和边的个数:6 10
请逐个输入顶点的内容: 0 1 2 3 4 5
请输入第1条边依附的两个顶点的编号和权值: 2 0 1
请输入第2条边依附的两个顶点的编号和权值: 2 1 5
请输入第3条边依附的两个顶点的编号和权值: 2 4 6
请输入第4条边依附的两个顶点的编号和权值: 2 5 4
请输入第5条边依附的两个顶点的编号和权值: 2 3 5
请输入第6条边依附的两个顶点的编号和权值: 0 1 6
请输入第7条边依附的两个顶点的编号和权值: 0 3 5
请输入第8条边依附的两个顶点的编号和权值: 4 1 3
请输入第9条边依附的两个顶点的编号和权值: 4 5 6
请输入第10条边依附的两个顶点的编号和权值: 5 3 2
输出通过边信息储存的图:
no      from    to    weight
edge[0]  2        0        1
edge[1]  5        3        2
edge[2]  4        1        3
edge[3]  2        5        4
edge[4]  2        1        5
edge[5]  2        3        5
edge[6]  0        3        5
edge[7]  2        4        6
edge[8]  0        1        6
edge[9]  4        5        6
输出Kruskal算法得的最小生成树(from,to)weight:
(2,0)1
(5,3)2
(4,1)3
(2,5)4
(2,1)5

D:\大学生活\大学的各种实验报告\算法实验报告\Project2\x64\Debug\Project2.exe (进程 14192)已退出, 代码为 0。
按任意键关闭此窗口。...
```

构造书本 P107 的无向图 G，两个算法生成最小代价生成树的结果与书本一致。

五、实验小结

通过本次实验，我们深入理解了贪心算法在最小生成树问题中的应用。Kruskal 算法和 Prim 算法都是有效的贪心法解决最小代价生成树问题的方法，其中选择适当的算法取决于具体的应用场景和图的特性。

在实验过程中，我们注意到贪心算法的局限性，即它无法解决所有优化问题。然而，在最小生成树问题中，贪心法的简单性和高效性使其成为一种常用的解决方法。

总体而言，本实验为我们提供了对贪心算法和最小生成树问题的深刻理解，为今后在图论和算法设计领域的学习奠定了基础。