

## 

**《离散数学》课程实验报告**

# 题目 最小生成树

## 姓 名 尹诚成

学 号 2351279

学 院 计算机科学与技术学院

专 业 软件工程

教 师 唐剑锋

二〇二四 年 十二 月 二十 日

**目录**

1 实验目的1

2 实验内容1

3 实验环境1

4 实验原理1

4.1 最小生成树1

4.2 构建最小生成树的算法2

5 实验过程2

5.1 实验思路2

5.2 实验设计2

5.2.1 数据结构设计2

5.2.2 结构体Edge设计3

5.2.3 类 MinimumSpanningTree设计3

5.2.4 程序主体架构设计4

5.3 程序功能实现5

5.3.1 输入城市个数与城市之间距离功能的实现 5

5.3.2 退出程序功能的实现6

5.3.3 动态内存申请失败后异常处理的实现6

5.4 核心算法实现6

5.4.1并查集相关算法实现6

5.4.2 Kruskal算法的实现7

6 实验数据分析7

7 实验心得9

8 程序源文件9

1. **实验目的**

本次实验主要目的是理解和实现最小生成树的算法，通过实际编程来理解算法的工作原理，并将理论应用于实际问题解决中。本实验旨在加强学生对离散数学中图论部分的理解，并提高使用计算机语言解决问题的能力。

1. **实验内容**

问题描述：如下图所示的路网赋权图表示七个城市之间的一些道路距离，要求给出一个路网设计方案，使得各城市之间既能够保持连接，又使得路网距离之和最小，并计算其最小值。

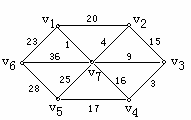


图 2.1 路网赋权图

1. **实验环境**

程序开发语言：C++

集成开发环境：Microsoft Visual Studio 2022 (Release模式)

编译运行环境：本项目适用于 x86 架构和 x64 架构

1. **实验原理**

4.1.最小生成树

最小生成树（MST）是图论中的一个重要概念。在一个加权连通图中，生成树是一个由所有顶点构成的一个无环联通子图，并且具有最少的边。如果这个树的边具有权重，则最小生成树的所有边的权重之和是所有生成树中最小的。

最小生成树具有一些重要性质。对于一个边权各不相同的连通图，它的最小生成树是唯一的。如果图中有相同权重的边，可能存在多个最小生成树。最小生成树是全连通的，这意味着树中的每个顶点都与其他所有顶点间接或直接相连。作为树的一种，最小生成树不会包含任何环。

最小生成树在多个领域有着广泛的应用。在设计电信网络、计算机网络或其他类型的网络时，MST可以用来最小化网络的总线长度或成本。在数据科学中，MST可用于构建聚类算法。

4.2.构建最小生成树的算法

构建最小生成树的两种著名算法是Prim算法和Kruskal算法。其中Prim算法适用于边密集的图，Kruskal算法适用于边稀疏的图。

Prim 算法从图的一个顶点开始，逐步增长最小生成树。在每个步骤中，它都添加最小权重的边，这条边连接树与图中的某个未加入树的顶点。这个过程持续进行，直到所有顶点都被包含在树中。

Kruskal 首先将图中的所有边按照权重进行排序，然后从最小的边开始添加到生成树中，但会避免形成环。该过程一直持续到生成树包含了所有的顶点。

1. **实验过程**

5.1.实验思路

本实验的核心思路是通过实现和应用Kruskal算法来构建最小生成树，以此解决一个具体的问题——在给定城市间建设成本最低的路网。实验将从理解最小生成树的概念出发，选择Kruskal算法作为构建最小生成树的算法，然后通过编程实践在一个加权连通图中寻找最小生成树。

Kruskal 算法的核心思想如下：

(1)假设图G是不连通的，对图的所有边按权值进行非降序排序；

(2)依次考虑排序后的边，如果当前边加入最小生成树T不会形成回路，则加入T，否则，丢弃该边；

(3)持续此过程直至生成树T包含的所有边都被考虑过；

(4)输出最小生成树T。

5.2.实验设计

5.2.1.数据结构设计

在使用Kruskal算法建立最小生成树的过程中，项目采用邻接矩阵的数据结 构来表示图，主要基于以下几个考虑：

(1)邻接矩阵适用于稠密图的场景，即图中大部分顶点之间都有边相连，类 似于本项目中的城市路网场景；

(2)便于实现Kruskal 算法中的关键操作；

(3)提高算法的运行效率，尤其是在处理大量节点时；

(4)简化算法的实现，使代码更加易于理解和维护。

5.2.2.结构体Edge设计

Edge 结构体是为图论算法（如最小生成树算法）中表示图的边而设计的。 它用于表示图中的一条边，包括边的起点、终点和权重。此外，它通过重载<运算符来比较两条边的权重，使其能够用于需要边权重比较的算法中，例如在实现 Kruskal 算法时对边进行排序。

Edge 结构体的数据成员、运算符重载定义及含义如下：

int start：边的起点

int end：边的终点

int weight：边的权重

bool operator<(const Edge& other) const { return this->weight < other.weight; }

重载<运算符。它使得可以直接比较两个Edge实例。比较基于边的权重，如 果当前边的权重小于另一条边的权重，则返回true；否则返回false。

5.2.3.类MinimumSpanningTree设计

MinimumSpanningTree类是为寻找加权连通图的最小生成树而设计的。此类具体实现了Kruskal算法，在图的所有边中选取最小权重的边来构建生成树，同时避免形成任何环，从而达到总权重最小化的目标。它包括了初始化图结构、处理边的权重、执行Kruskal算法核心步骤以及提供最终结果的相关方法。此外，类中还包含了对并查集数据结构的实现，用于辅助实现Kruskal算法中对边的选择和环的检测。其数据成员、成员函数定义及含义如下：

int\*\* costMatrix：二维数组，用于存储路网中各个城市间的道路距离

int vertexNum：顶点数量，代表路网中的城市数目

int minTotalCost：最小生成树的总权重

std::vector<Edge> edges：存储途中所有边的向量

std::vector<int> parents：并查集中每个元素的父节点

std::vector<int> heights：并查集中每个元素所在的树的高度

MininumSpanningTree(const int v)：构造函数，接受顶点数量作为参数，初始化图结构和相关数组。

~MininumSpanningTree()：析构函数，释放动态分配的内存资源

int findSet(int v)：再并查集中查找并返回i的根节点

void mergeSet(int v1, int v2)：再并查集中合并两个集合，这两个集合分别包含顶点v1和v2。

bool insertEdge(const int v1, const int v2, const int cost)：设置图中两个顶点之间的边权重，

void Kruskal()：使用Kruskal算法生成最小生成树并计算其总权重

int getMinTotalCost()：返回当前最小生成树的总权重

5.2.4.程序主体架构设计

程序主体架构设计为：

(1)初始化和创建路网：当程序开始执行时，首先进入main函数。在这里，程序首先执行初始化操作，包括清除屏幕并显示用户界面。接下来，程序请求用户输入城市（节点）的数量，这是通过调用inputInteger函数实现的。用户输入后，使用这个数值来创建MinimumSpanningTree实例，此实例代表整个路网；

(2)输入边的权重：一旦路网初始化完成，程序接着引导用户输入任意两个城市（即图中的节点）之间的距离（权重）。这些权重通过insertEdge方法设置到MinimumSpanningTree 的实例中。对于所有城市对，都需要执行这个步骤，以确保图中的所有边都被考虑；

(3)构建最小生成树：输入完所有边的权重后，main函数调用Kruskal方法来构建最小生成树。此方法内部实现了Kruskal算法，它首先排序所有边， 然后逐一选择边，构建无环的最小生成树；

(4)输出结果：构建完最小生成树后，程序通过getMinTotalCost方法获取并输出总权重，展示了构建的最小生成树的总成本。最后，main 函数中包含用户退出程序的选项。一旦用户选择退出，程序将清理所有资源并结束执行。

(5)异常处理：在整个程序中，还包括了对不同情况的异常处理，例如内存分配失败或用户输入无效数据。这确保了程序的健壮性和用户友好性。

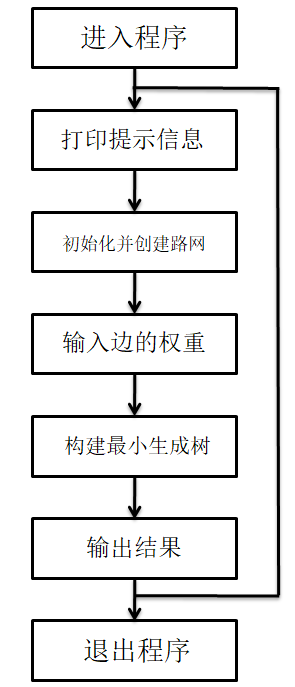


图 5.2.4 程序主体架构流程设计图

5.3 程序功能实现

5.3.1.输入城市个数和城市间距离功能的实现

程序通过调用 inputInteger 函数输入城市个数与城市间距离。

inputInteger 函数用于获取用户输入的整数，同时限制输入必须在指定的范围内。inputInteger 函数对输入非法的情况进行了处理，代码具体执行逻辑如下：

(1)进入一个无限循环，它会一直运行直到用户提供有效的输入；

(2)用户的输入被读取到num变量中，这里采用double类型来接收输入以便后续检查；

(3)进行输入验证：std::cin.fail()检查输入流的状态是否异常，确保没有发生数据类型输入错误，num==static\_cast<int>(num)检查 用户输入是否为整数，通过将其转换为整数再比较，num < minNum || num > maxNum确保输入在指定的范围内；

(4)合法输入处理：如果用户提供了合法的输入，函数会返回转换后的整数值；

(5)非法输入处理：如果用户提供的输入不合法，函数会输出错误消息，清除输入流的错误状态，丢弃输入缓冲区中的内容，并继续循环以等待用户提供合法的输入。

5.3.2.退出程序功能的实现

退出程序的功能是通过在main函数内部的一个循环结构实现的。此功能允许用户在完成操作后选择是否退出程序。

程序功能实现思路：

(1)程序使用 do-while 循环来反复执行程序和询问用户是否退出程序。在每次循环的开始，使用system("cls")清除屏幕，以提供清晰的界面；

(2)在完成一轮操作后，程序会输出询问用户是否退出程序的提示：“是否退出程序 [y/n]: ”。此时，程序再次调用inputLogicalValue函数，这次用于接收用户的退出决定。参数被设置为'n'（不退出）和'y'（退出）；

(3)如果用户输入'y'（对应真值），则inputLogicalValue函数返回true，导致 do-while 循环结束，程序随之退出。如果用户输入'n'（对应假值），inputLogicalValue 函数返回 false，do-while 循环继续，用户可以进行新一轮操作。

5.3.3 动态内存申请失败后异常处理功能的实现

在进行MinimumSpanningTree 类中私有数据成员等的动态内存申请时，程序使用new(std::nothrow)来尝试分配内存。new(std::nothrow)在分配内存失败时不会引发异常，而是返回一个空指针（NULL或nullptr），代码检查指针是否为空指针，如果为空指针，意味着内存分配失败，这时程序将执行以下操作：

(1)向标准错误流 std::cerr 输出一条错误消息"Error: Memory allocation failed."，指出内存分配失败；

(2)调用exit函数，返回错误码 MEMORY\_ALLOCATION\_ERROR（通过宏定义方式定义为-1），用于指示内存分配错误，并导致程序退出。

5.4 核心算法实现

5.4.1.并查集相关算法的实现

findSet 是一个并查集算法的核心部分，用于查找给定元素（这里是顶点v）所属的集合的根节点。

当调用findSet(v)时，算法检查parents[v]是否大于0，即该元素是否是其集合的根节点。如果不是根节点，则递归地调用findSet 在并查集的树结构中向上追溯，直到找到根节点。在查找的过程中，实现了路径压缩，即parents[v]被更新为其根节点，这有助于加快未来的查找操作。

mergeSet函数用于合并包含两个不同顶点v1和v2的两个集合。

首先，使用findSet 函数找到 v1 和 v2 的根节点，分别称为 root1 和root2。如果root1和root2相同，意味着x和y已经在同一个集合中，无需合并。如果root1和root2不同，则需要将两个集合合并。这是通过更新parent数组实现的，将一个集合的根节点指向另一个。合并时考虑到了树的高度。总是将较小树的根指向较大树的根，以保持树的平衡，优化性能。

5.4.2.Kruskal 算法的实现

kruskalMST 函数实现了 Kruskal 算法，用于在加权图中构建最小生成树。 kruskalMST 函数的实现思路如下：

(1)初始化parents和heights数组，为每个顶点设置父节点为无效节点，并将度置0；

(2)将所有的边按权重进行排序；

(3)创建一个空的results向量来存储最小生成树中的边；

(4)遍历每条边，使用findSet检查当前边的两个顶点是否属于不同的集合；

(5)如果是，将该边添加到results 中，并使用mergeSet合并两个集合；

(6)这个过程一直持续，直到加入了vertexNum-1条边或考虑了所有的边；

(7)通过累加results中边的权重，更新minTotalCost来得到最小生成树的总权重；

(8)打印或返回最小生成树的边和总权重。

1. **实验数据分析**

实验数据分析1：分别输入超过上下限的整数、浮点数、字符、字符串，可以验证程序对输入非法的情况进行了处理，并要求用户重新输入城市个数。

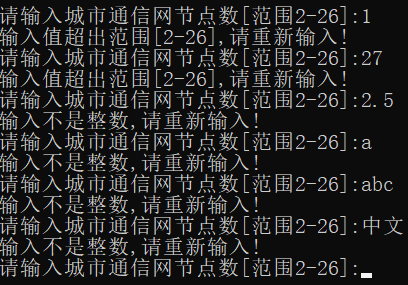


图 6.1 实验数据 1

实验数据分析2：分别输入超过上下限的整数、浮点数、字符、字符串，可以验证程序对输入非法的情况进行了处理，并要求用户重新输入城市间距离。

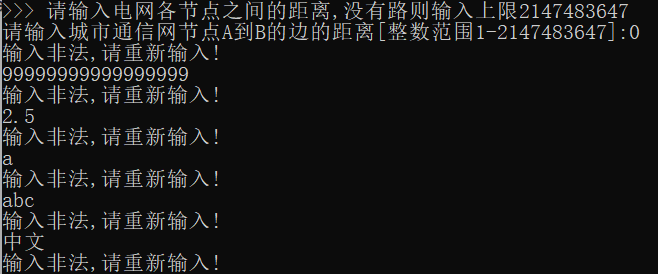


图 6.2 实验数据 2

实验数据分析3：创建路网并输入城市之间的距离，可以建立Kruskal最小生成树。程序成功建立Kruskal最小生成树，并输出Kruskal最小生成树及其总权重。

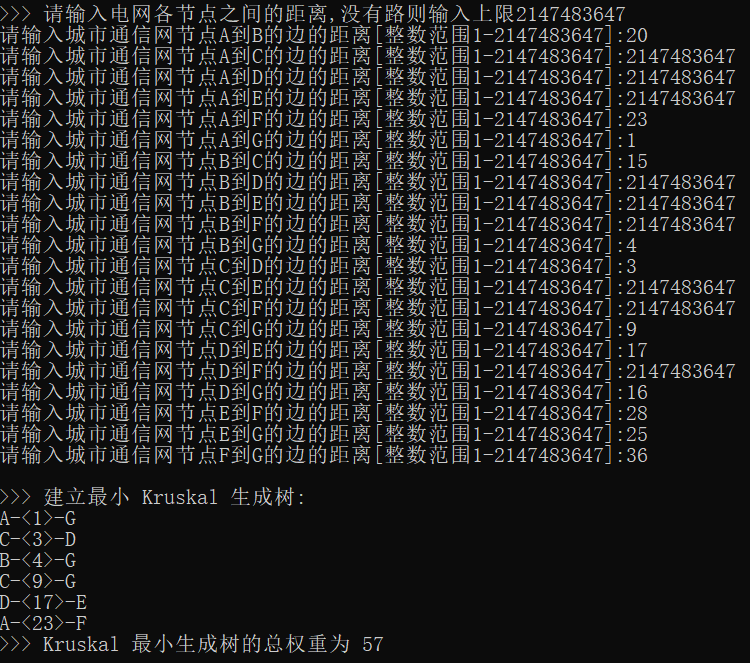


图 6.3 实验数据 3

实验数据分析4：当输入n时，程序清空屏幕并执行下一次最小生成树的运算。

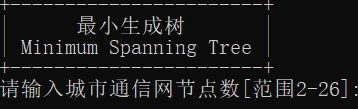


图 6.4 实验数据 4

实验数据分析5：当输入y时，程序退出。

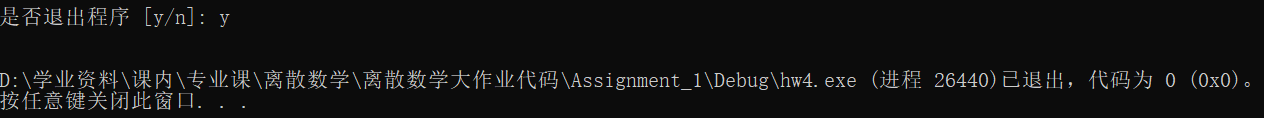


图 6.5 实验数据 5

1. **实验心得**

在完成最小生成树实验后，我体会到了理论与实践结合的重要性。本次实验中，我不仅复习了最小生成树的理论知识，特别是Kruskal算法的原理和实现，还通过编程实践将这些理论应用到了实际问题解决中。

实验过程中，我面临的最大挑战是将理论转化为实际的代码。设计数据结构以表示图，理解并实现并查集数据结构来支持Kruskal算法，这些都需要深入的理解和细心的编程。这个过程让我更加熟悉了C++语言，尤其是面向对象编程的思想，如类的使用、数据封装和函数重载等。 通过处理输入数据和异常情况，我认识到了编写健壮且用户友好的程序的重要性。

看到自己的程序能够成功地解决实际问题，计算给定城市的最小生成树总权重，让我感到非常满足。这不仅证明了我的编程能力，也加深了我对离散数学中图论部分的理解。

1. **程序源文件**

#include<iostream>

#include <vector>

#include<limits>

#include<conio.h>

#include<algorithm>

#define MAX\_VERTEX\_NUM 26

#define MAX\_COST INT\_MAX

#define MEMORY\_ALLOCATION\_ERROR -1

#define INVAVID\_PARENT\_NODE -1

/\*

\* Function Name: inputInteger

\* Function: input an integer within a range

\* Input Parameters:T minNum

\* T maxNum

\* const char\* prompt

\* Return Value: int

\*/

template<typename T>

int inputInteger(T minNum, T maxNum, const char\* prompt = "") {

double num;

while (true) {

if (prompt != "")

std::cout << "请输入" << prompt << "[范围" << minNum << "-" << maxNum << "]:";

std::cin >> num;

if (std::cin.fail()||static\_cast<int>(num)!= num|| num < minNum || num > maxNum) {

std::cout << "输入非法,请重新输入!" << std::endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

}

else

return num;

}

}

/\*define the Edge struct\*/

struct Edge {

int start;

int end;

int weight;

bool operator<(const Edge& other) const {

return this->weight < other.weight;

}

};

/\*define the MininumSpanningTree class\*/

class MininumSpanningTree {

private:

int\*\* costMatrix;

int vertexNum;

int minTotalCost;

std::vector<Edge> edges;

std::vector<int>parents;

std::vector<int> heights;

int findSet(int v);

void mergeSet(int v1, int v2);

public:

MininumSpanningTree(const int v);

~MininumSpanningTree();

bool insertEdge(const int v1, const int v2, const int cost);

void Kruskal();

int getMinTotalCost();

};

/\*

\* Function Name: findSet

\* Function: find the root of the set that v belongs to

\* Input Parameters:int v

\* Return Value: int

\*/

int MininumSpanningTree::findSet(int v) {

if (parents[v] < 0)

return v;

else

findSet(parents[v]);

}

/\*

\* Function Name: mergeSet

\* Function: merge two sets that v1 and v2 belong to

\* Input Parameters:int v1

\* int v2

\* Return Value: void

\*/

void MininumSpanningTree::mergeSet(int v1, int v2) {

int root1 = findSet(v1), root2 = findSet(v2);

if (heights[root1] > heights[root2])

parents[root2] = root1;

else if (heights[root1] < heights[root2])

parents[root1] = root2;

else if (parents[root1] == parents[root2]) {

parents[root2] = root1;

heights[root1]++;

}

}

/\*

\* Function Name: MininumSpanningTree

\* Function: the constructor of the MininumSpanningTree class

\* Input Parameters:int V

\* Return Value: void

\* Notes: Class external implementation of member functions

\*/

MininumSpanningTree::MininumSpanningTree(const int v) {

/\*cost matrix initialization\*/

vertexNum = v;

minTotalCost = 0;

parents.resize(vertexNum, INVAVID\_PARENT\_NODE);

costMatrix = new(std::nothrow) int\* [vertexNum];

if (costMatrix == NULL) {

std::cerr << "Error: Memory allocation failed." << std::endl;

exit(MEMORY\_ALLOCATION\_ERROR);

}

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

costMatrix[i] = new(std::nothrow) int[vertexNum];

if (costMatrix[i] == NULL) {

std::cerr << "Error: Memory allocation failed." << std::endl;

exit(MEMORY\_ALLOCATION\_ERROR);

}

for (int j = 0; j < vertexNum; j++) {

costMatrix[i][j] = 0;

}

}

std::cout << std::endl;

std::cout << ">>> 城市通信道路网节点";

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

std::cout << static\_cast<char>(i + 'A') << " ";

}

std::cout << "创建成功!" << std::endl << std::endl;

/\*input edge distance\*/

std::cout << ">>> 请输入电网各节点之间的距离,没有路则输入上限" << MAX\_COST << std::endl;

for (size\_t i = 0; i < vertexNum; i++) {

for (size\_t j = i + 1; j < vertexNum; j++) {

std::cout << "请输入城市通信网节点" << static\_cast<char>(i + 'A') << "到" << static\_cast<char>(j + 'A')

<< "的边的距离[整数范围1-" << MAX\_COST << "]:";

int cost = inputInteger(1, MAX\_COST);

insertEdge(i, j, cost);

}

}

}

/\*

\* Function Name: ~MininumSpanningTree

\* Function: the destructor of the MininumSpanningTree class

\* Input Parameters:void

\* Return Value: void

\* Notes: Class external implementation of member functions

\*/

MininumSpanningTree::~MininumSpanningTree() {

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

delete[] costMatrix[i];

}

delete[] costMatrix;

}

/\*

\* Function Name: insertEdge

\* Function: insert an edge into the cost matrix

\* Input Parameters:const int v1

\* const int v2

\* const int cost

\* Return Value: true / false

\* Notes: Class external implementation of member functions

\*/

bool MininumSpanningTree::insertEdge(const int v1, const int v2, const int cost) {

if (v1 < 0 || v1 >= vertexNum || v2 < 0 || v2 >= vertexNum || cost < 0 || cost > MAX\_COST) {

return false;

}

costMatrix[v1][v2] = costMatrix[v2][v1] = cost;

edges.push\_back({ v1,v2,cost });

return true;

}

/\*

\* Function Name: Kruskal

\* Function: create the minimal spanning tree using Kruskal algorithm

\* Input Parameters:void

\* Return Value: void

\* Notes: Class external implementation of member functions

\*/

void MininumSpanningTree::Kruskal() {

/\* Initialize the parents and heights of each vertex to -1 and 0 respectively \*/

parents.resize(vertexNum,INVAVID\_PARENT\_NODE);

heights.resize(vertexNum, 0);

/\* Sort the edges by weight \*/

std::sort(edges.begin(), edges.end());

int count = 0;

std::vector<Edge>results;

/\* Iterate through the edges and add the edge to the MST if it does not form a cycle \*/

for (const auto& edge : edges) {

int x = findSet(edge.start);

int y = findSet(edge.end);

if (x != y) {

results.push\_back(edge);

mergeSet(x, y);

count++;

if (count == vertexNum - 1)

break;

}

}

/\* Output the MST \*/

std::cout << std::endl<<">>> 建立最小 Kruskal 生成树:" << std::endl;

for (const auto& result : results) {

std::cout << static\_cast<char>(result.start + 'A') << "-<" << result.weight << ">-" << static\_cast<char>(result.end + 'A') << std::endl;

minTotalCost += result.weight;

}

}

/\*

\* Function Name: getMinTotalCost

\* Function: get the total cost of the minimal spanning tree

\* Input Parameters:void

\* Return Value: int

\* Notes: Class external implementation of member functions

\*/

int MininumSpanningTree::getMinTotalCost() {

return minTotalCost;

}

/\*

\* Function Name: inputLogicalValue

\* Function: input a logical value from user

\* Input Parameters:char FalseValue = '0'

\* char TrueValue = '1'

\* Return Value: true / false

\*/

bool inputLogicalValue(char FalseValue = '0', char TrueValue = '1')

{

while (true) {

char InputChar = \_getch();

if (InputChar == 0 || InputChar == -32)

InputChar = \_getch();

else if (InputChar == FalseValue || InputChar == TrueValue) {

std::cout << InputChar << std::endl << std::endl;

return InputChar == TrueValue;

}

}

}

/\*

\* Function Name: makeChoice

\* Function: choose the start node

\* Input Parameters:const char\* prompt

\* int vertexNum

\* Return Value: int

\*/

int makeChoice(const char\* prompt, int vertexNum)

{

std::cout << prompt;

while (true) {

char choice = \_getch();;

if (choice == 0 || choice == 32)

choice = \_getch();

if ('A' <= choice && choice <= static\_cast<char>(vertexNum + 'A' - 1)) {

std::cout << " [" << choice << "]" << std::endl << std::endl;

return choice;

}

}

}

/\*

\* Function Name: main

\* Function: the main function of the program

\* Return Value: int

\*/

int main() {

do {

/\*output the title\*/

system("cls");

std::cout << "+-----------------------+" << std::endl;

std::cout << "| 最小生成树 |" << std::endl;

std::cout << "| Minimum Spanning Tree |" << std::endl;

std::cout << "+-----------------------+" << std::endl;

/\*input the number of vertices\*/

int vertexNum = inputInteger(2, MAX\_VERTEX\_NUM, "城市通信网节点数");

MininumSpanningTree mininumSpanningTree(vertexNum);

/\*create minimal spanning tree\*/

mininumSpanningTree.Kruskal();

std::cout << ">>> Kruskal 最小生成树的总权重为 " << mininumSpanningTree.getMinTotalCost() << std::endl << std::endl;

std::cout << "是否退出程序 [y/n]: ";

} while (!inputLogicalValue('n', 'y'));

/\*wait for user input to quit\*/

return 0;

}