数据结构实验报告

学院：电子与信息学部

专业：自动化科学与工程

班级：自动化 003

姓名：张文硕

学号：2206110686

目录

实验一：表达式求值

1. 实验目的

通过使用栈和队列两种数据结构对于表达式进行计算和求值。

2. 实验要求

输入：通过命令行参数输入一个数字表达式

输出：（1）表达式正确的情况下输出结果
（2）命令行参数不正确时输出字符串 ERROR_01
（3）表达式出现格式错误时输出 ERROR_02
（4）表达式在计算过程中出现逻辑错误时输出字符串 ERROR_03

简明实例：输入：（2+3）*（3-1）^3　　输出：40
输入：((2+3）*（3-1）^3　　输出：ERROR_02

3. 设计思路

算法基本思想如下：

（1） 首先将操作数栈 OPND 设为空栈，而将'#'作为运算符栈 OPTR 的栈底元素，这样的目的是判断表达式是否求值完毕。（当处理完全时，出现#连续出现）

（2） 依次读入表达式的每个字符，表达式须以'#'结尾。

- 若是操作数则入栈 OPND,
- 若是运算符,则将此运算符 c 与 OPTR 的栈顶元素 top 比较优先级后执行相应的操作

（具体操作如下：

(i) 若 top 的优先级小于 c，即 top<c，则将 c 直接入栈 OPTR，并读入下一字符赋值给 c;

(ii) 若 top 的优先级等于 c，即 top=c，则弹出 OPTR 的栈顶元素，并读入下一字符赋值给 c，这一步目的是进行括号操作;

(iii) 若 top 优先级高于 c，即 top>c，则表明可以计算，此时弹出 OPND 的栈顶两个元素，并且弹出 OPTR 栈顶的的运算符，计算后将结果放入栈 OPND 中。）

（3） 直至 opter 的栈顶元素和当前读入的字符均为'#',此时求值结束。

运算符优先级表如下：

```
char pre[10][10] = {
    /*运算符之间的优先级制作成一张表格*/
        {'>','>','<','<','<','>','>','<','>','<'},
        {'>','>','<','<','<','>','>','<','>','<'},
        {'>','>','>','>','<','>','>','<','>','<'},
        {'>','>','>','>','<','>','>','<','>','<'},
        {'<','<','<','<','<','=','0','0','0','<'},
        {'>','>','>','>','0','>','>','0','>','>'},
        {'<','<','<','<','<','0','=','<','0','<'},
        {'<','<','<','<','<','0','0','0','=','<'},
        {'>','>','>','>','0','0','>','0','0','>'},
        {'>','>','>','>','<','>','>','<','>','>'}};//如果输入是 []  OR  () 则为=
```

顺序依次是：+；-；*；/；(；)；#；[；]；^；

ERROR_01 的处理：

针对输入的 aggc 进行判断，当其为 2 时，认为入参正确，不做响应。

当其不为 2 时，认为入参出现问题，返回 ERROR_01。

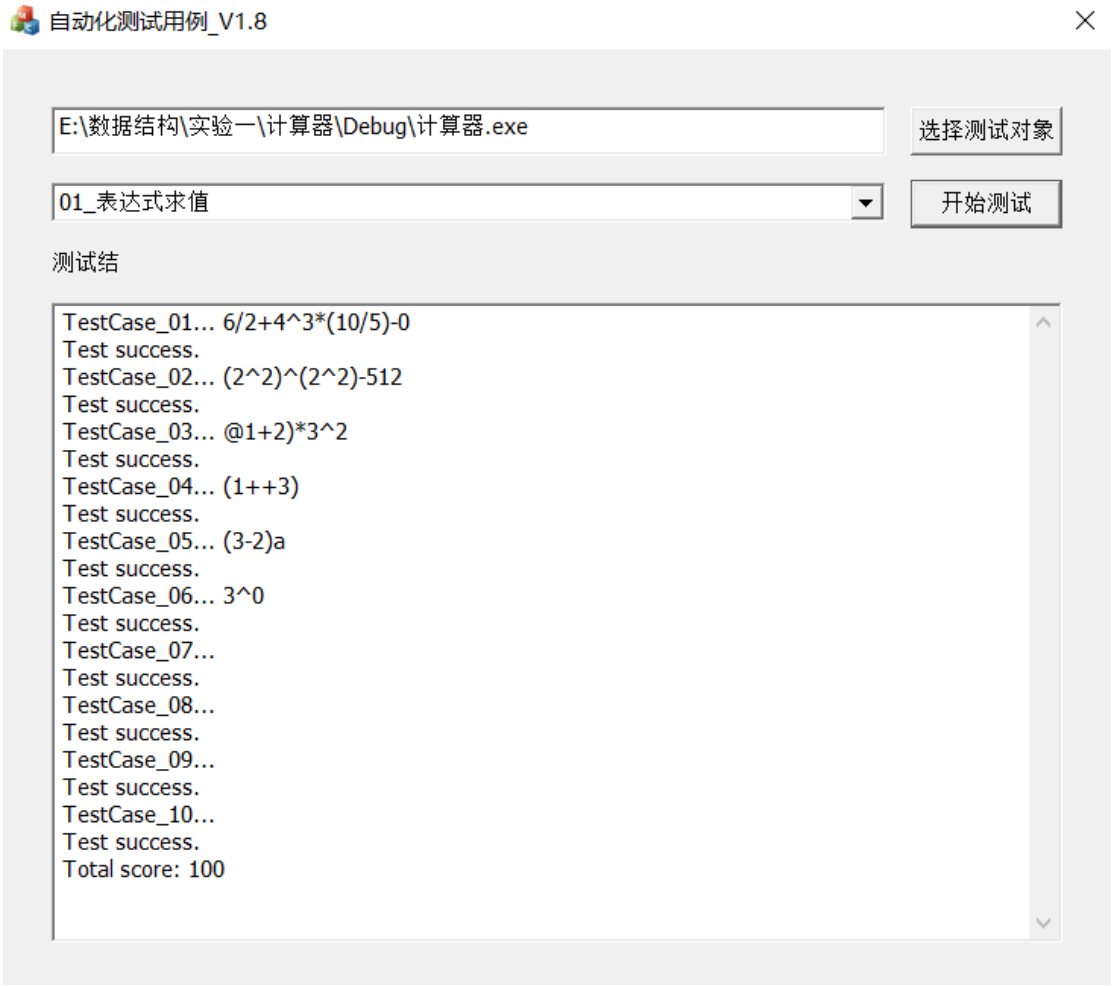
ERROR_02 的处理：

ERROR_02 属于输入的格式出现了错误，我们认为，当 OPTR 之中出现 0 时存在问题。（本来不用当成对出现在操作符之中的操作符成对出现了），即计算器之中的逻辑错误。

ERROR_03 的处理：

出现逻辑错误的情况只有可能为数字除以 0。即计算器之中的数学错误。

4. 运行结果



5. 问题与思考

本次代码中存在的问题：

1.文件名忘记更改为英文的了

2.所有的数字都不应该出现，应该出现的数字必须进行宏定义

3.Vs code 如果报警告的就去改自己的代码，不允许出现 #define

_CRT_SECURE_NO_WARNINGS.

4.malloc 函数在使用完之后必须判断是否分配成功，if(!p) return OVERFLOW

所有的指针变量在使用完之后应当 free 掉，不可以就放在内个地方。

5.不应当出现{<语句 1>;<语句 2>;<语句 3>;}这种情况，这种东西很不好看。

6.所有变量名都应当是清晰可见，直接易懂的，绝不允许出现 a,STACK1 这种，存啥东西名字里边一定要说清楚。

7.函数分配内存的时候应当合理分配，不要突然多了突然少了的。+10 也应当为+IncreaseAmount，使用宏定义处理，不允许出现任何一个数字。

8.不要自己造轮子了!!!!!!!!!!!!!!!!!!

一小部分思考：

① argv 只支持读取操作，不支持写入操作，会报系统的错误，答案正确但编译无法通过。同时 argv[0]为系统自行使用的，输入的参数为 argv[1]，这个为一个字符串。strlen(argv[1])可以直接读取其长度。

② aggc 为参数的个数，默认参数个数为 1. 当用户写入的时候，参数个数就变成 1+写入参数个数了。

6. 源代码


source.c

实验二：使用 KMP 算法实现字符串的匹配

1. 实验目的

熟练的掌握数据结构中串这种数据类型；学会使用相较于朴素的模式识别算法更加先进的 KMP 算法进行识别和匹配。
同时，在数据结构试验之中熟悉和了解串的性质和使用方法。

2. 实验要求

输入：通过命令行参数输入原字符串和模式字符串。
输出：　（1）命令行参数不正确输出字符串 ERROR_01；
　　（2）如果查找到模式串，输出关键字在字符串中的位置（计数从 1 开始）；
　　（3）如果未找到模式串则输出-1 。
实例：　输入：“select * from duaadual” “dual”；输出：19
　　输入：“select * from duaadual” “duel”；输出：-1
代码质量要求：
　　1、优选 C 语言，禁止直接调用 C++ STL 库；
　　2、除循环变量外，其它变量命名使用有明确含义的单词或缩写，不建议使用拼音；
　　3、禁止出现魔鬼数字；　　4、添加必要的程序注释；
　　5、统一代码格式，例如：{}和空行；6、变量初始化，不要依赖默认赋值；
　　7、入参检查，“外部输入输入不可靠”，指针判空（一级指针、二级指针……），循环变量上下限；
　　8、malloc 与 free 配对。

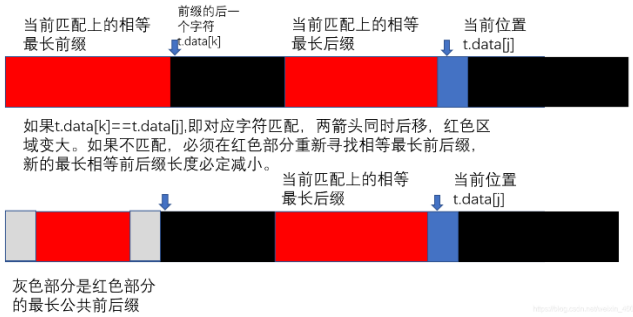
3. 实验原理

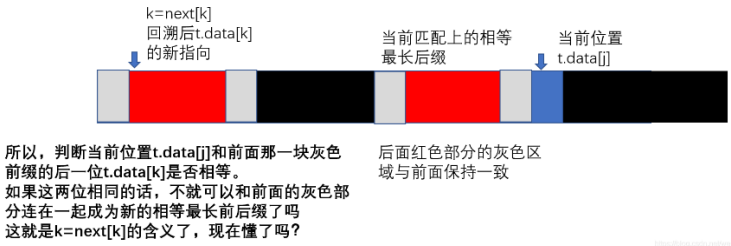
KMP 算法

- Next 数组的求解方法：

① 最长相等的前后缀
　字符串 abcdab
　前缀的集合：{a,ab,abc,abcd,abcda}，前缀为从前边开始计字符。
　后缀的集合：{b,ab,dab,cdab,bcdab}，后缀为从后边开始记字符，正常顺序。
　则其中相等的最长的前后缀为 ab

② 进行求解 next 数组的过程





最后形成公式：

$$next[j]=\begin{cases}\max\{k \mid 0<k<j,\ \text{且}\ \overbrace{t_0t_1\ldots t_{k-1}}^{\text{开头的}k\text{个字符}}=\overbrace{t_{j-k}t_{j-k+1}\ldots t_{j-1}}^{\text{后面的}k\text{个字符}}\} & \text{当此集合非空时}\\ -1 & \text{当}j=0\text{时}\\ 0 & \text{其他情况}\end{cases}$$

③ 具体的代码：

```
Status get_next(String* Target, int* next)
{
    if (!Target) return ERROR;
    if (!Target->base) return ERROR;
    int i = 1;//next的下标，从1开始。0位用于存储总串长度。
    next[i] = 0;
    next[0] = Target->CurLen;//0位置储存T串的总长度
    int k = 0;//k为可以满足条件的最大长度值。
    while (i < Target->CurLen)
    {
        if (k == 0 || Target->base[i] == Target->base[k])
        {
            i++;
            k++;
            next[i] = k;
        }//如果Pj=Pk或着直接为零，那么，直接加一就行了。
        else k = next[k];//当不满足上述条件的时候，直接进行处理，让k取到next【k】。
    }
    return OK;
}//定义取用下一位的操作
```

核心为：如果相等，则直接加一；如果不相等，那么让 k=next[k];（迭代之后形成的）

- 寻址的过程

```
while (i <= Station->CurLen && k <= Target->CurLen)
{
    if (k == 0 || Station->base[i] == Target->base[k])
    {
        ++k;
        ++i;
    }//如果为0，那么相当于向后移动一位，如果两个相等，向后移动一位是属于正常的现象。
    else
    {
        k = next[k];//如果匹配失败，直接进入next[k]位置进行匹配与判断。这个时候i是不动的。
    }
}
```

ERROR_01 的判定

针对输入的 aggc 进行判断，当其为 2 时，认为入参正确，不做响应。

当其不为 2 时，认为入参出现问题，返回 ERROR_01。

4. 运行结果

5. 问题与思考

本次实验之中存在最后一个用例长时间无法通过的问题。(用例为超过 8bit 存储长度的用例)

原因：

在对于目标串进行处理时，将目标串长度存在了目标串的[0]位。

相当于将一个数字存进了一个字符型的空间之中。Char 型变量空间长度为 8。当其为 8 位 1 时，其可以存储的最大数字为 255.当其长度超过 255 时，数字将脱离控制。

改进方法：

直接调用串结构之中的 Curlen 代替目标串[0]位存储目标传长度.

```
typedef struct str
{
    ElemType* base;
    int CurLen;
}String;//声明字符串结构体,主要是给Station和Target用的,base的第一个base[0]不可以用于存放字符串总长度Curlen(在next等地方用于占位)
```

6. 源代码

source.cpp

实验三：基于哈夫曼树的编码/译码

1. 实验目的

掌握二叉树的生成、遍历等操作，及哈夫曼编码/译码的原理。

2. 实验要求

输入：通过命令行参数输入字符串(长度>=20)和码字。

输出：（1）命令行参数不正确输出 ERROR_01;
（2）编码失败输出 ERROR_02;
（3）译码失败输出 ERROR_03;
（4）在同一行中输出编码结果和译码结果，中间使用空格隔开。

简明实例：

输入：“stupid is as stupid does” “101101001110”
输出：“00111011111011100011110100001100100001100011101111011100011110011101001010 0 pat”

其他要求：

① 基于该哈夫曼树，实现非递归的先序遍历算法，输出该树所有的节点、节点的权值、节点的度和节点所在的层数;
② 在实现时要求哈夫曼树的左右孩子的大小关系满足，左孩子节点权值小于右孩子节点权值，若左右孩子权值相等，按字母顺序排列（序号小的字母在左孩子）。

代码质量要求：

1、优选 C 语言，禁止直接调用 C++ STL 库;
2、除循环变量外，其它变量命名使用有明确含义的单词或缩写，不建议使用拼音;
3、禁止出现魔鬼数字;
4、添加必要的程序注释;
5、统一代码格式，例如：{}和空行;
6、变量初始化，不要依赖默认赋值;
7、入参检查，“外部输入输入不可靠”，指针判空（一级指针、二级指针……），循环变量上下限;
8、malloc 与 free 配对;
9、尽量少用全局变量;
10、编译错误解决，从前往后处理，提示出错的行不一定是错误的根因。

3. 实验原理

霍夫曼编码原理

1）将信源符号的概率按减小的顺序排队。
2）把两个最小的概率相加，并继续这一步骤，始终将较高的概率分支放在右边，直到最后变成概率 1 。
3）画出由概率 1 处到每个信源符号的路径，顺序记下沿路径的 0 和 1 ，所得就是该符号的霍夫曼码字。
4）将每对组合的左边一个指定为 0，右边一个指定为 1（或相反）。

树的遍历原理

- 递归遍历
  先访问根节点，再访问左孩子节点，最后访问右孩子结点

1. 访问根节点时直接调用 visit 函数
2. 访问左孩子节点的时候先将根节点压进栈中，再访问左孩子结点。
3. 当左孩子结点完全访问完了之后，取栈顶元素，访问右孩子结点。
4. 对于 1，2，3 步进行循环，直到栈空终止循环。

```
Status preOrder(BetreeNode* ht, infolist* infos)
{
    if (!ht || !infos) return ERROR;
    BetreeNode* stack[InitSize];//用一个数组实现了线性栈的功能
    int top = -1;//-1为标识符，没有实际的使用，下标不可能为-1
    BetreeNode* cursor = ht;
    while (cursor != NULL || top != -1) {
        if (cursor != NULL) {
            stack[++top] = cursor;
            InputInfoList(infos, cursor); //入栈时，访问输出
            cursor = cursor->lchild;
        }
        else {
            cursor = stack[top--];
            cursor = cursor->rchild;
        }
    }
    return OK;
}
```

解码原理：

<循环开始：条件为输入 01 串未被遍历完>

创建指针指向根节点，当为 0 时向左，当为 1 时向右。

当找到内容时将指针重置为根节点。

<循环终止>

ERROR_03 的处理:

和解码过程相配合，当出现最终的一位不在叶子节点上的时候，认为解码失败

ERROR_02 的处理：

当全部为 abcdefg……时，即所有的出现过的字符权重均为 1 时，

或者当输入的小于二十时

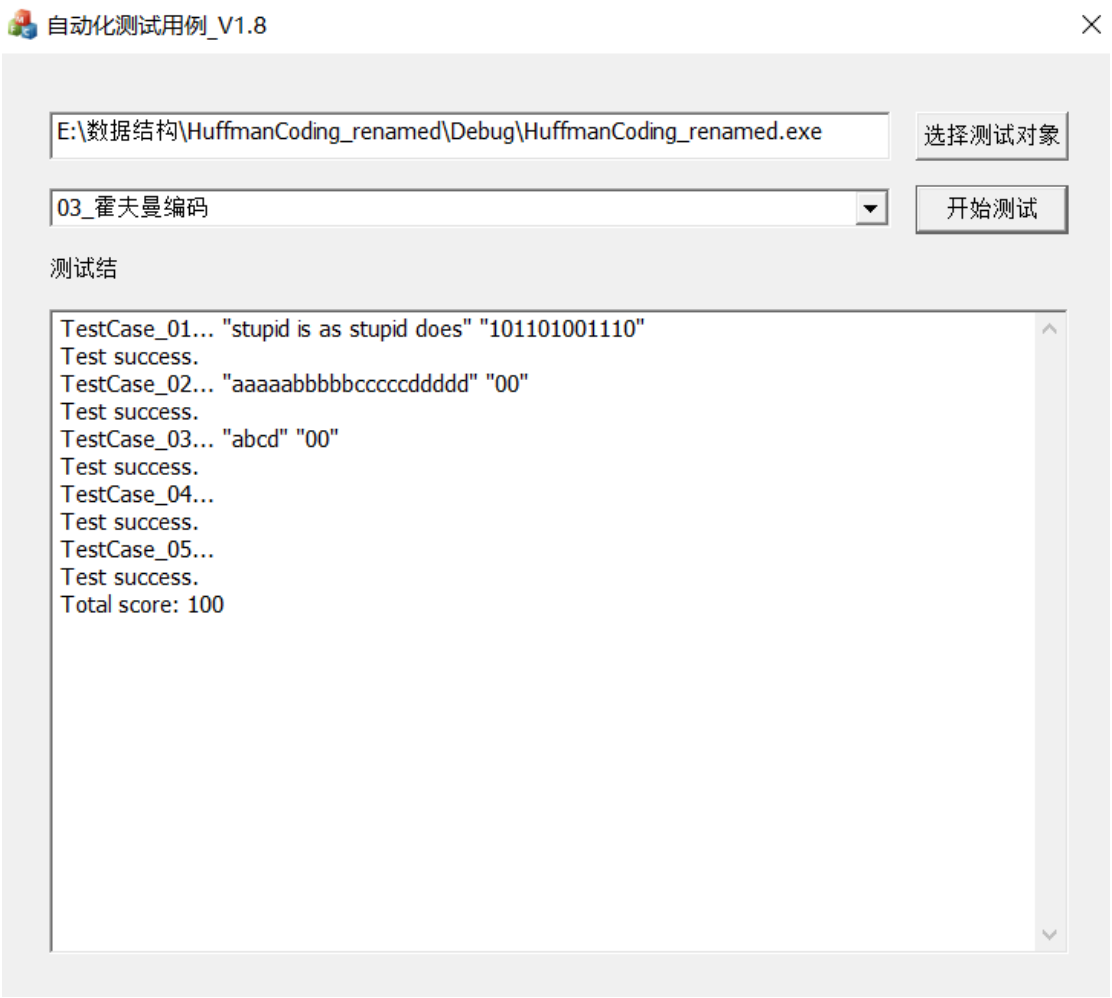
认为出现错误 ERROR_02

ERROR_01 的判定

针对输入的 aggc 进行判断，当其为 2 时，认为入参正确，不做响应。

当其不为 2 时，认为入参出现问题，返回 ERROR_01。

4. 实验结果



5. 问题与思考

问题：

本次虽然较好的解决了霍夫曼编码的逻辑和代码，但是存在代码过长的问题。

同时，由于代码过长的问题，整体的构架不好观察和分析。

解决办法：

① 学习使用 Cpp，使用面向对象的编程语言，将函数进行封装，便于调用和分析

② 将部分函数部分拆出来一个完整的头文件，或者直接拆成两个文件，便于之后的调用。

6. 源代码

source.cpp

实验四：无向图最短路径搜索

1. 实验目的

熟练掌握图的操作，掌握 Dijkstra 算法的原理。

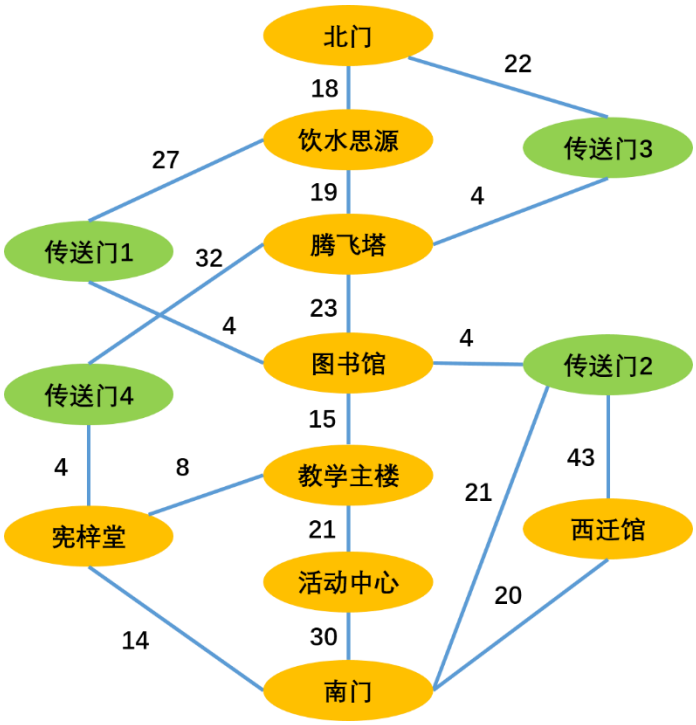
2. 实验要求

输入：通过命令行参数输入起点和终点的位置名称。
输出：（1）命令行参数不正确输出 ERROR_01;
（2）获取最短路径失败时输出 ERROR_02;
（3）获取最短路径成功时输出路径长度。

实际地图：

1 右图为校内知名建筑物示意平面图（其中“传送门”用于增加网络复杂度），以边表示建筑物间的路径，各条路径上方的数字表示路径长度。

2 针对该图进行构建数据结构和算法，通过命令行参数输入任意两建筑物的名称，可查询建筑物间的最短路径长度，并输出最短路径。



代码质量要求：
1、优选 C 语言，禁止直接调用 C++ STL 库;
2、除循环变量外，其它变量命名使用有明确含义的单词或缩写，不建议使用拼音;
3、禁止出现魔鬼数字;
4、添加必要的程序注释;
5、统一代码格式，例如：{}和空行;
6、变量初始化，不要依赖默认赋值;
7、入参检查，“外部输入输入不可靠”，指针判空（一级指针、二级指针……），循环变量上下限;
8、malloc 与 free 配对;
9、尽量少用全局变量;
10、编译错误解决，从前往后处理，提示出错的行不一定是错误的根因;

3. 实验原理

图的存储和头文件的引入：

为了使得代码可以被重复使用，即代码被二次利用时只需要更改图的模式识别代码，同时改变宏定义之中的最大次数，本次将图的文件和宏定义的部分单独的写成一个头文件，在书写主程序时对他进行引用即可。

```
#ifndef         _COMMON_H//和文件名一致，将.换成_；前面加_。
#define         _COMMON_H//避免重复声明，使用 if no define,define……endif

#include<stdio.h>//添加程序所需要的所有头文件
#include<string.h>

#define Status int//添加宏定义部分

static int a = 0 ； //可以定义静态变量，在所有.c 文件中使用。

extern int    test(char* p,int n)//声明所有子函数

#endif
```

迪杰特斯拉算法思想：

设 G=(V,E)是一个带权有向图，把图中顶点集合 V 分成两组，第一组为已求出最短路径的顶点集合（用 S 表示，初始时 S 中只有一个源点，以后每求得一条最短路径 ，就将加入到集合 S 中，直到全部顶点都加入到 S 中，算法就结束了），第二组为其余未确定最短路径的顶点集合（用 U 表示），按最短路径长度的递增次序依次把第二组的顶点加入 S 中。在加入的过程中，总保持从源点 v 到 S 中各顶点的最短路径长度不大于从源点 v 到 U 中任何顶点的最短路径长度。此外，每个顶点对应一个距离，S 中的顶点的距离就是从 v 到此顶点的最短路径长度，U 中的顶点的距离，是从 v 到此顶点只包括 S 中的顶点为中间顶点的当前最短路径长度。

具体步骤：

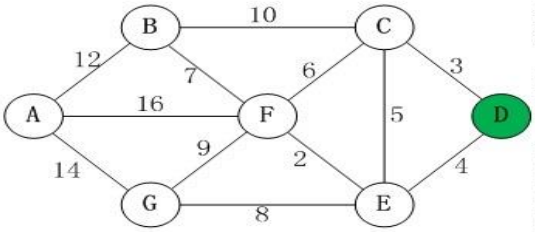
(1) 初始时，S 只包含起点 s；U 包含除 s 外的其他顶点，且 U 中顶点的距离为"起点 s 到该顶点的距离"[例如，U 中顶点 v 的距离为(s,v)的长度，然后 s 和 v 不相邻，则 v 的距离为∞]。

(2) 从 U 中选出"距离最短的顶点 k"，并将顶点 k 加入到 S 中；同时，从 U 中移除顶点 k。

(3) 更新 U 中各个顶点到起点 s 的距离。之所以更新 U 中顶点的距离，是由于上一步中确定了 k 是求出最短路径的顶点，从而可以利用 k 来更新其它顶点的距离；例如，(s,v)的距离可能大于(s,k)+(k,v)的距离。

(4) 重复步骤(2)和(3)，直到遍历完所有顶点。

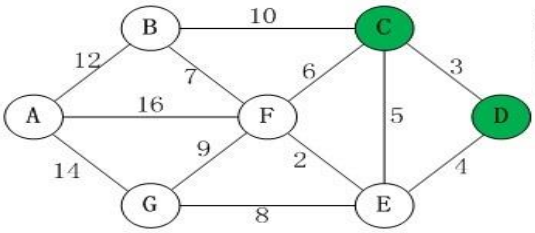
简明实例：



第1步：
选取顶点D

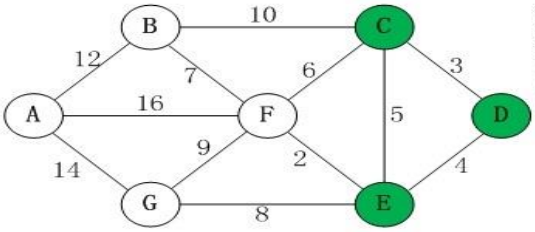
```
S={D(0)}
U={A(∞),B(∞),C(3),E(4),F(∞),G(∞)}
```

注：
(01) S是已计算出最短路径的定点的集合
(02) U是未计算出最短路径的定点的集合
(03) C(3)表示顶点C到起点D的最短距离是3



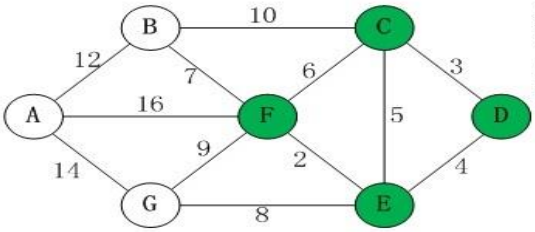
第2步：
选取顶点C

```
S={D(0),C(3)}
U={A(∞),B(13),E(4),F(9),G(∞)}
```



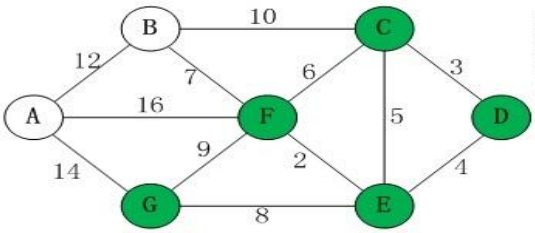
第3步：
选取顶点E

```
S={D(0),C(3),E(4)}
U={A(∞),B(13),F(6),G(12)}
```



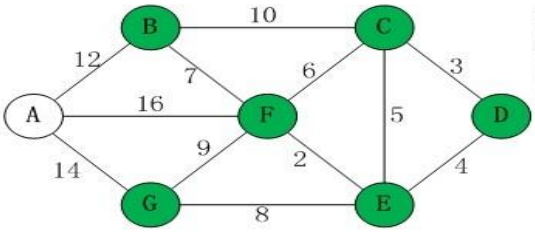
第4步：
选取顶点F

```
S={D(0),C(3),E(4),F(6)}
U={A(22),B(13),G(12)}
```



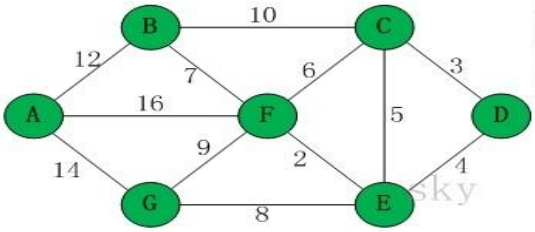
第5步：
选取顶点G

```
S={D(0),C(3),E(4),F(6),G(12)}
U={A(22),B(13)}
```



第6步：
选取顶点B

```
S={D(0),C(3),E(4),F(6),G(12),B(13)}
U={A(22)}
```



第7步：
选取顶点A

```
S={D(0),C(3),E(4),F(6),G(12),B(13),A(22)}
```

4. 实验结果

5. 问题与思考

思考：

对于 Git 的使用和一些其他的东西了解的不够深刻，对于编译器编译过程的理解依然不够深刻，这些都需要后续的学习去加以提升。

6. 源代码

头文件：

map.h

源文件：

source.cpp