《离散数学》课程大作业实验报告

姓名：封钰震

学号：1951362

专业：信息管理与信息系统

2022 年 5 月 23 日

《离散数学》课程大作业实验报告

封钰震

目录

注：由于担心字符编码问题，程序中所有中文提示语都使用英文进行表述。

1 实验一

1.1 问题 A

1.1.1 题目简介

从键盘输入两个命题变元 p 和 q 的真值，求它们的合取、析取、蕴含和双向蕴含的真值。

1.1.2 解题思路

合取和析取可以直接通过编程语言的逻辑运算符得到，蕴含和双向蕴含需要进行等值转换。即

$$p \rightarrow q \Leftrightarrow \neq p \vee q \tag{1}$$

$$p \leftrightarrow q \Leftrightarrow (\neg p \vee q) \wedge (\neg q \vee p) \tag{2}$$

该小题较为简单，无需用到特殊的数据结构或算法。程序运行结果如图 1所示。

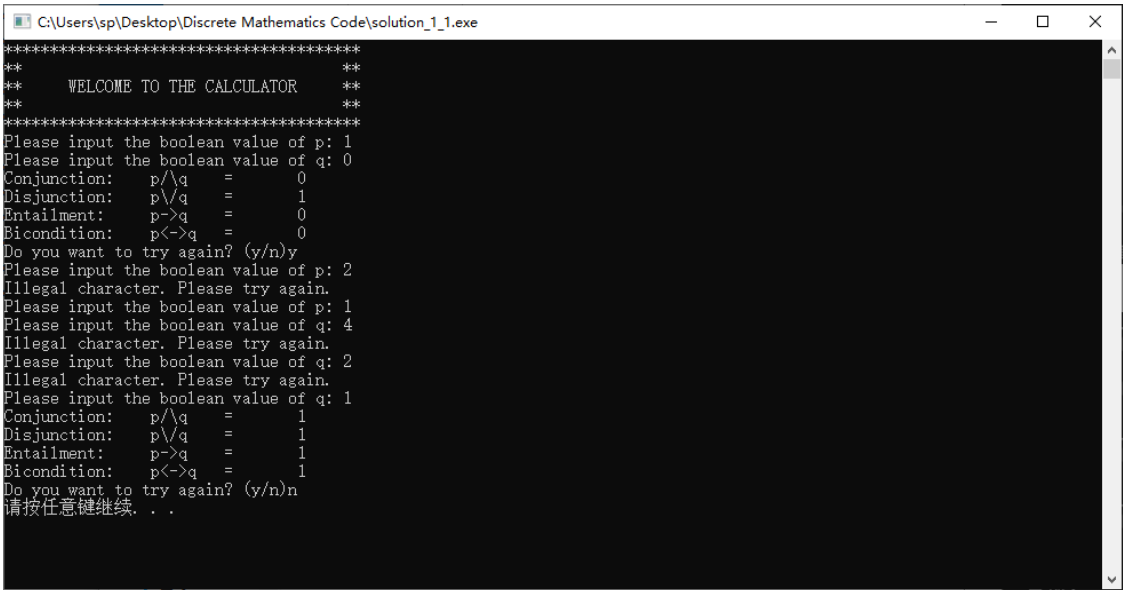

图 1: 实验一 A 程序运行结果

1.2 问题 B&C

1.2.1 题目简介

求任意一个命题公式的真值表，并根据真值表求主范式。

1.2.2 解题思路

该题可分为如下几步：

1. 将输入的中缀表达式转化为不需要括号来标定优先级的后缀表达式；
2. 在后缀表达式中计算变量个数 n；
3. 从 $k = 0, 1, \ldots, 2^n - 1$ 进行遍历，将 k 转化为二进制后，代入后缀表达式进行求解；
4. 求解的结果若为真，则 m_k 为极小项；若为假，则 M_k 为极大项。

1.2.3 核心问题

首先，我们需要考虑如何将中缀表达式转化为后缀表达式。后缀表达式没有二义性，不需要使用括号来标识优先级，代入计算更加便捷。有关前、中、后缀表达式可见《Discrete mathematics and its applications》[1]。在转化时，我们将使用数据结构——栈，转化的算法如 Procedure 1所示。

Procedure 1 中缀表达式转化为后缀表达式

```
Input: infix notation IS
Output: postfix notation PS
ST ← empty stack
PS ← empty string
for each character c ∈ IS do
    if c is a letter then                              ▷ 如果遇到字母，则直接添加到后缀表达式中
        PS ← PS + c
    else if c = '(' then                               ▷ 如果遇到左括号，则将其入栈
        Push(ST, c)
    else if c = ')' then ▷ 如果遇到右括号，则将栈中第一个左括号之前的运算符都加入至后缀表达式中
        while not IsEmpty(ST) and Top(ST) ≠ '(' do
            PS ← PS+ Top(ST)
            Pop(ST)
        end while
        if Top(ST) = '(' then
            Pop(ST)
        end if
    else         ▷ 如果遇到其他运算符，则将栈顶优先级不小于它的运算符都加入至后缀表达式中
                                   ▷ 此处为忽略左括号，我们定义左括号的优先级最小
        while not IsEmpty(ST) and Priority(c) ⩽ Priority(Top(ST)) do
            PS ← PS+ Top(ST)
            Pop(ST)
        end while
    end if
end for
while not IsEmpty(ST) do                               ▷ 将栈中剩余字母按顺序添加到后缀表达式中
    PS ← PS+ Top(ST)
    Pop(ST)
end while
```

其次，要统计表达式中变元的个数，我们将遍历后缀表达式，这里我们使用标准模板库的

```
std::map<char, int>
```

按照各变元在后缀表达式中出现的顺序，将各变元映射到 $0, 1, \ldots, n-1$，其中 n 为变元的数目。

接着，要生成 2^n 个解。我们让 k 从 0 到 $2^n - 1$ 开始遍历，使用标准库函数

```
std::bitset<MAX_VAR_NUM>(k).to_string().substr(MAX_VAR_NUM - n)
```

将其转化为一串长度为 n 的 0-1 序列。

然后，我们把这串 0-1 序列作为解代入后缀表达式进行求解。求解同样会使用数据结构——栈，求解的算法如 Procedure 2所示。

在计算结果的同时，记录下成真赋值和成假赋值对应的 k。求解的结果若为真，则 m_k 为极小项；若为假，则 M_k 为极大项。最后，根据极小项和极大项，输出主析取范式和主合取范式。

Procedure 2 求解后缀表达式

Input: postfix notation PS, solution S
Output: value v

$ST \leftarrow$ empty stack
for each character $c \in PS$ **do**
  if c is a letter **then** ▷ 如果遇到字母，则将其的值入栈
    Push(ST, $S[c]$)
  else
    if c = '!' **then** ▷ 如果遇到否定运算符，则计算栈顶的值的否定后，将结果入栈
      $x \leftarrow$ not Top(ST)
      Pop(ST)
      Push(ST, x)
    else ▷ 如果遇到其他二元运算符，则需要栈顶两个值，进行计算后，将结果入栈
      $y \leftarrow$ Top(ST)
      Pop(ST)
      $x \leftarrow$ Top(ST)
      Pop(ST)
      Push(Calculate(x, c, y))
    end if
  end if
end for
$v \leftarrow$ Top(ST) ▷ 最后的解即为栈剩余的一个值

程序运行结果如图 2，3，4所示。

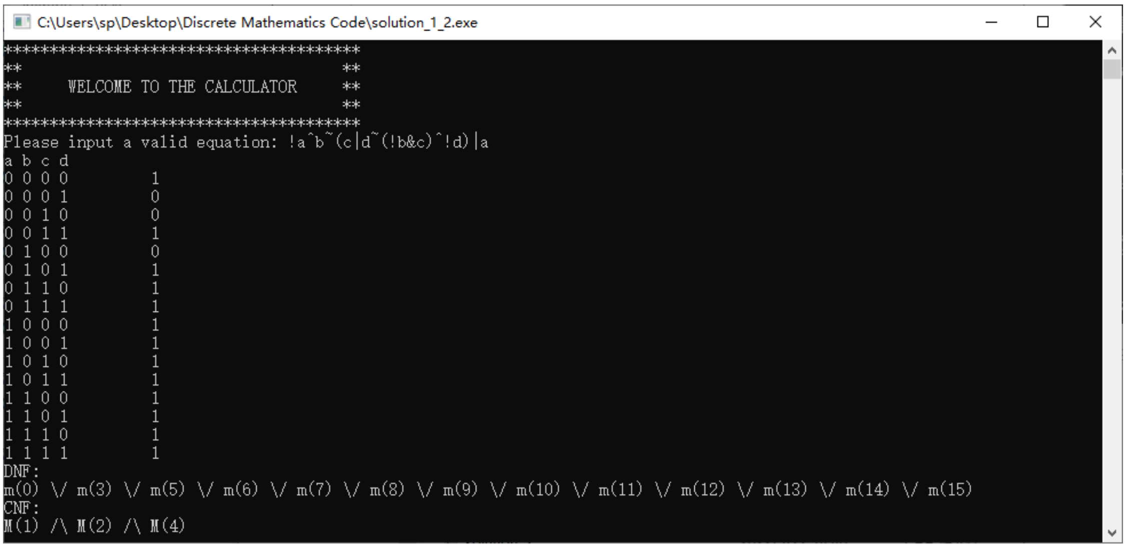

图 2: 实验一 B（1）程序运行结果

图 3: 实验一 B（2）程序运行结果

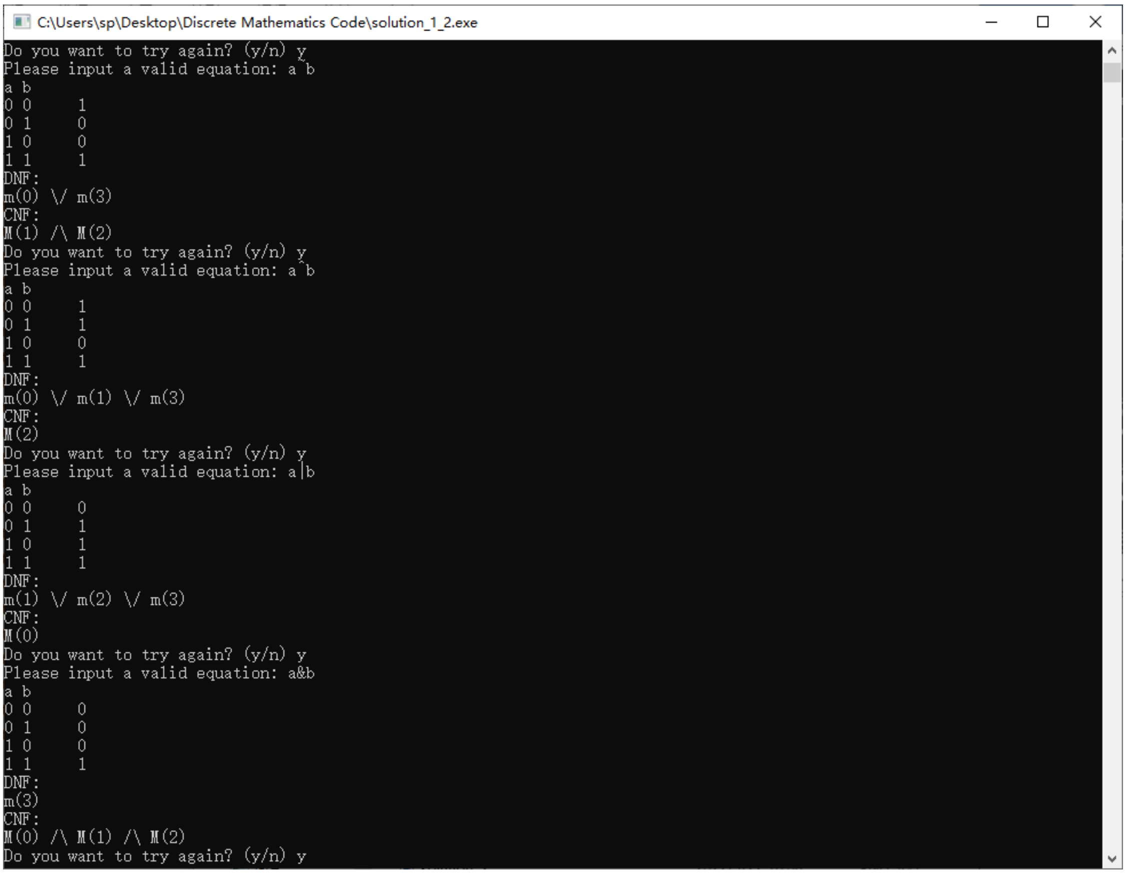

图 4: 实验一 B（3）程序运行结果

2 实验二

2.1 题目简介

根据下面命题，用命题逻辑推理方法确定谁是作案者，并给出推理过程，C 语言源代码及演示界面。

1. 营业员 A 或 B 偷了手表；
2. 若 A 作案，则作案不在营业时间；
3. 若 B 提供的证据正确，则货柜未上锁；
4. 若 B 提供的证据不正确，则作案发生在营业时间；
5. 货柜上了锁。

2.2 解题思路

定义命题变元：

- A：营业员 A 偷了手表
- B：营业员 B 偷了手表
- C：作案不在营业时间
- D：B 提供的证据正确
- E：货柜未上锁

则上述的五个命题可以符号化为：

$$(A \vee B) \wedge (A \rightarrow C) \wedge (D \rightarrow E) \wedge (\neg D \rightarrow \neg C) \wedge \neg E \tag{3}$$

等价为：

$$(A \vee B) \wedge (\neg A \vee C) \wedge (\neg D \vee E) \wedge (D \vee \neg C) \wedge \neg E \tag{4}$$

要寻找作案者，即寻找命题变元 A, B（当然，也包括 C, D, E）的值，使式 (4) 为真。对五个变元的 $2^5 = 32$ 种情况进行遍历，即可得到答案。

该题较为简单，无需用到特殊的数据结构或算法。程序运行结果如图 5所示。

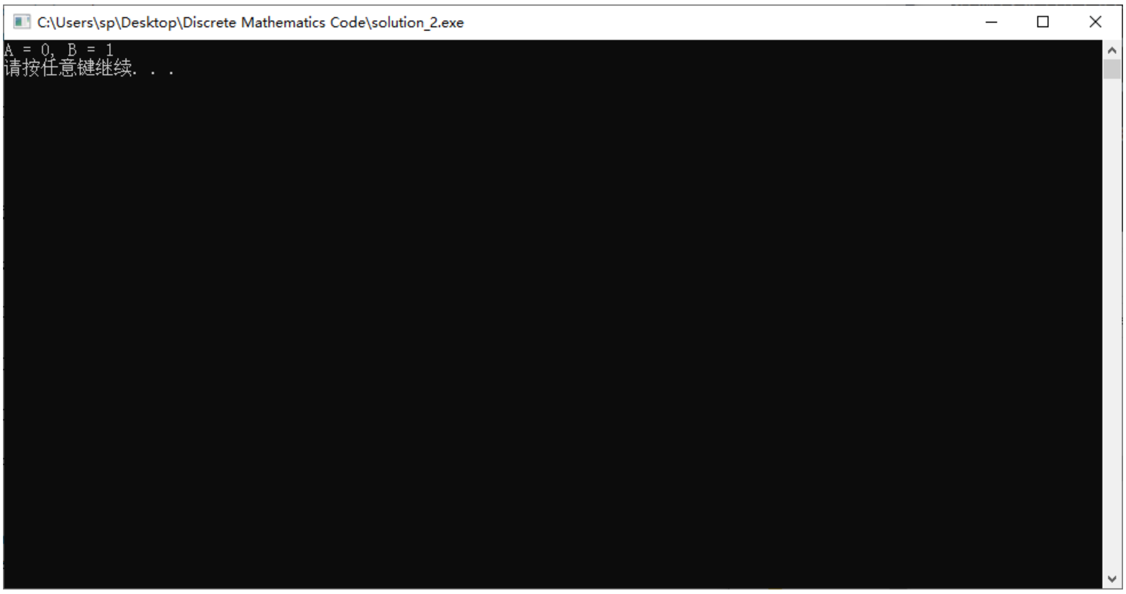

图 5: 实验二程序运行结果

3 实验三

3.1 题目简介

求关系的自反、对称和传递闭包。

3.2 解题思路

对于求自反闭包，只要将矩阵的主对角线全部置为 1 即可，时间复杂度为 $O(n)$。

对于求对称闭包，只要将关系矩阵并（∧）其的转置矩阵，时间复杂度为 $O(n^2)$。

对于求传递闭包，一种时间复杂度为 $O(n^4)$ 的算法是求

$$M_R^* = M_R \wedge M_R^{[2]} \wedge \cdots \wedge M_R^{[n]} \tag{5}$$

其中，$M_R^{[n]} = M_R^{[n-1]} \odot M_R$，$M_R$ 是关系 R 的 0-1 矩阵，R^* 是关系 R 的传递闭包。另一种时间复杂度为 $O(n^3)$ 的 Warshall 算法将在实验六中进行应用。在该实验中，我们仍先使用第一种算法。

3.3 核心问题

集合的二元关系使用了二维数组来进行存储，具体到 C++，则是使用了 STL 模板：

```
std::vector<std::vector<bool>>
```

最后程序运行结果如图 6所示。

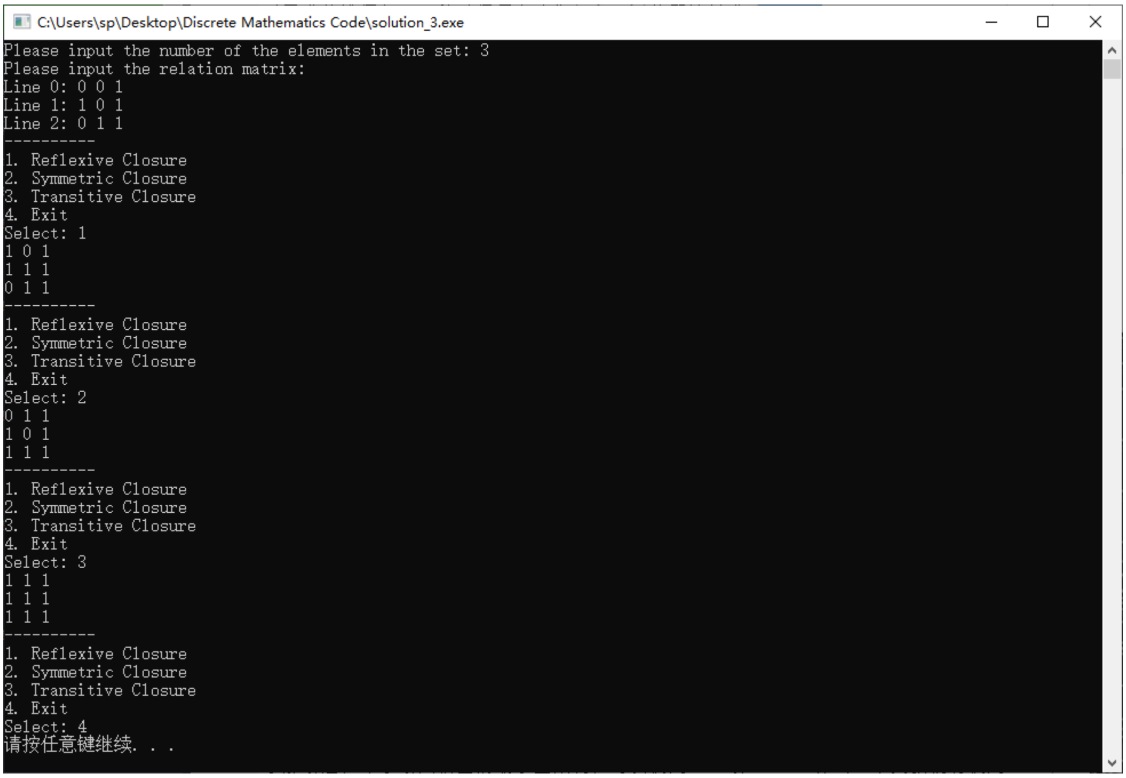

图 6: 实验三程序运行结果

4 实验四

4.1 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。

4.2 解题思路

求解该题，即求该赋权图的最小生成树，可以使用 Prim 或 Kruskal 算法。本文我们将使用 Kruskal 算法，它的伪代码如 Procedure 3所示，摘自《Introduction to Algorithms》[2] 第 23.2 节。算法的时间复杂度为 $O(m\log m)$，其中，m 为图中边的个数。

Procedure 3 Kruskal 算法

Input: graph G, weights of edges w
Output: minimum spanning tree A

```
A = ∅
for each vertex v ∈ G.V do
    makeSet(v)
end for
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight do
    if findSet(u) ≠ findSet(v) then
        A = A ⋃ {(u, v)}
        union(u, v)
    end if
end for
```

4.3 核心问题

首先，关于将所有的边按照边权进行排序，可以使用一般的快速排序算法。在 C++ 中，使用算法库中的 `std::sort` 即可。这一步的时间复杂度为 $O(m\log m)$，其中，m 为图中边的个数。

另外，关于如何判断某条边的加入是否会形成回路，在《Introduction to Algorithms》[2] 中，是通过判断该边的两顶点是否在同一集合中实现的。起初，所有的点都在不同的集合中，当某条边将不同集合的两点联结起来后，就将这两个集合合并。如果一条边 $e=(u,v)$ 的两顶点 u,v 在同一集合，也就是说它们在同一生成树中，那么根据生成树的性质[1]，他们之间有通路，如果添加了边 e，就形成了回路，因此需要跳过这样的边。我们在 C++ 中这样实现：定义一个长度为图的顶点数的数组 `arr1` 存放各个元素属于哪个集合，再定义一个这样的结构 `map1`：

```
std::map<int, std::vector<int>>
```

存放每个集合里有哪些元素。当需要检索点在哪个集合中时，只需要在 `arr1` 检索即可。当需要合并两个集合 A,B 时，一方面，在 `map1` 中查找 B 对应的元素，在 `arr1` 中将这些元素对应的集合都改为 A；另一方面，在 `map1` 中将 A,B 进行合并。这一步的时间复杂度为 $O(m)$，其中，m 为图中边的个数。

最后程序运行结果如图 7所示。

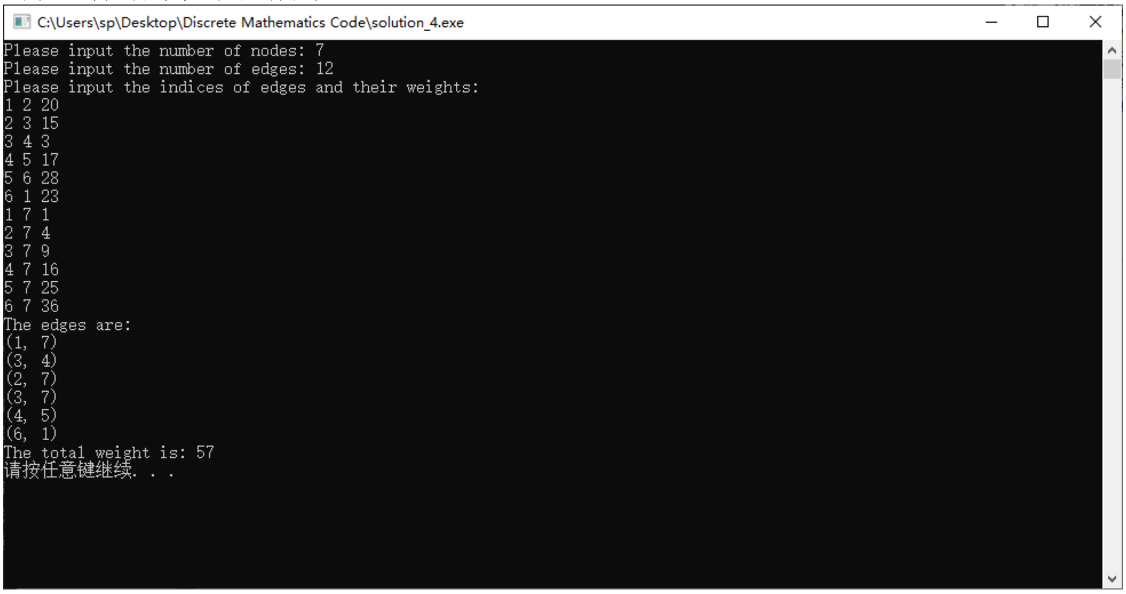

图 7: 实验四程序运行结果

5 实验五

5.1 题目简介

输入一组通信符号的使用频率，求各通信符号对应的前缀码。

5.2 解题思路

给定频率，求最优前缀码，即使用 Huffman 编码对这些符号进行编码。Huffman 编码的伪代码如 Procedure 4所示，摘自《Discrete mathematics and its applications》[1] 第 11.2.4 节。

Procedure 4 Huffman 编码

Input: symbols a_i with frequencies $w_i, i=1,\ldots,n$
Output: Huffman tree F

$F \leftarrow$ forest of n rooted trees, each consisting of the single vertex a_i and assigned weight w_i
while F is not a tree **do**
    Replace the rooted trees T and T' of least weights from F with $w(T) \geqslant w(T')$ with a tree having a new root that has T as its left subtree and T' as its right subtree.
    Label the new edge to T with 0 and the new edge to T' with 1.
    Assign $w(T)+w(T')$ as the weight of the new tree.
end while

5.3 核心问题

首先，关于树的存储，在 C++ 中，我们采用这样的结构体，左右节点默认为空：

```
struct node {
   int value;
   node* left = NULL;
   node* right = NULL;
};
```

其次，在伪代码的第 3行，需要从所有根节点中抽取两个权重最小的节点；而在伪代码的第 5行，又需要将新生成的根节点插入到原来的序列中。基于这样的需求，我们使用**最小堆**来维护这种序列。在 C++ 中，我们使用标准库的 `std::make_heap`，`std::pop_heap`，`std::push_heap` 等函数，各自的复杂度为 $O(n)$，$O(\log n)$，$O(\log n)$[2]，其中 n 为元素个数。

最后，需要对树进行遍历，从而得到各个字符的编码。在遍历到叶节点后，就输出遍历路径上各边的比特。在具体实现时，采用递归的结构即可，如下所示：

```
void preorderWalk(node* x, vector<bool> code)
{
   if (x->left) {
      auto code_left = code;
      code_left.push_back(0);
      preorderWalk(x->left, code_left);
   }
   if (x->right) {
      auto code_right = code;
      code_right.push_back(1);
      preorderWalk(x->right, code_right);
   }
   if (!x->left && !x->right)
   {
      cout << x->value << '\t';
      for (size_t i = 0; i < code.size(); i++)
      {
         cout << code.at(i);
      }
      cout << endl;
      return;
   }
}
```

程序运行结果如图 8所示。

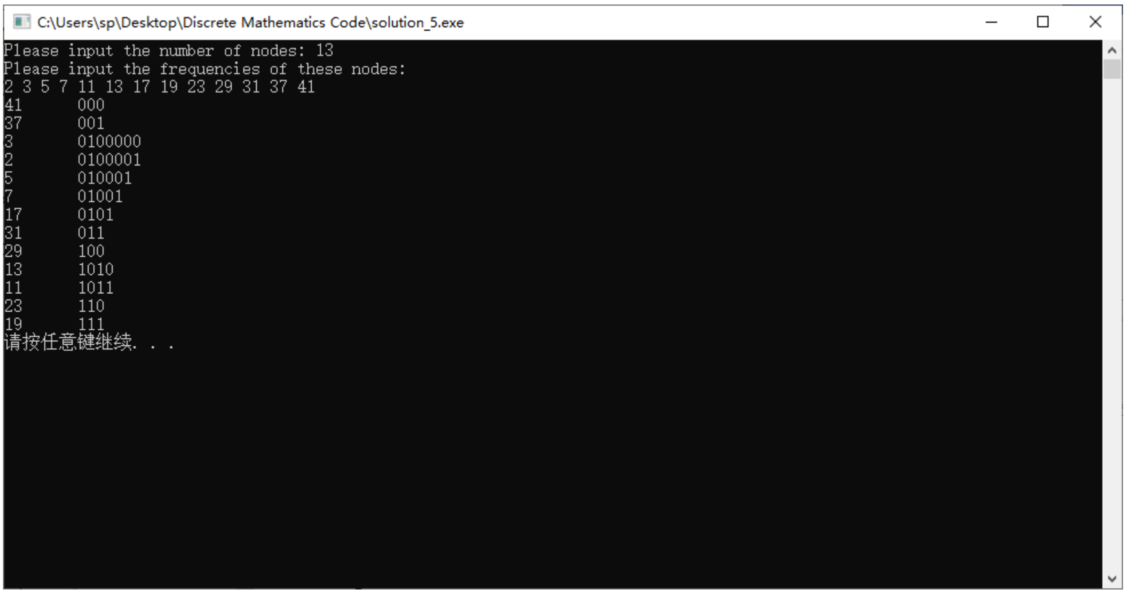

图 8: 实验五程序运行结果

6 实验六

6.1 题目简介

使用 Warshall 算法求二元关系的传递闭包。

6.2 解题思路

使用的算法即为 Warshall 算法，Warshall 算法的伪代码见《离散数学（第二版）》[3] 的第 7.5 节。Warshall 算法的时间复杂度为 $O(n^3)$，其中 n 为集合的元素个数。

6.3 核心问题

集合的二元关系使用了二维数组来进行存储，具体到 C++，则是使用了 STL 模板：

```
std::vector<std::vector<bool>>
```

在将二元关系转化为矩阵时，需要将集合元素的名称对应到矩阵的行（列）号，这里，我们使用 STL 模板提供的映射：

```
std::map<std::string, int>
```

当需要检索某名称对应的序号时，使用成员函数 `find` 即可。

在我们改进的程序中：

- 元素名称不限于只能为字符，可以为字符串；
- 若输入的元素名称检索不到，会抛出警告。

最后程序运行结果如图 9所示。

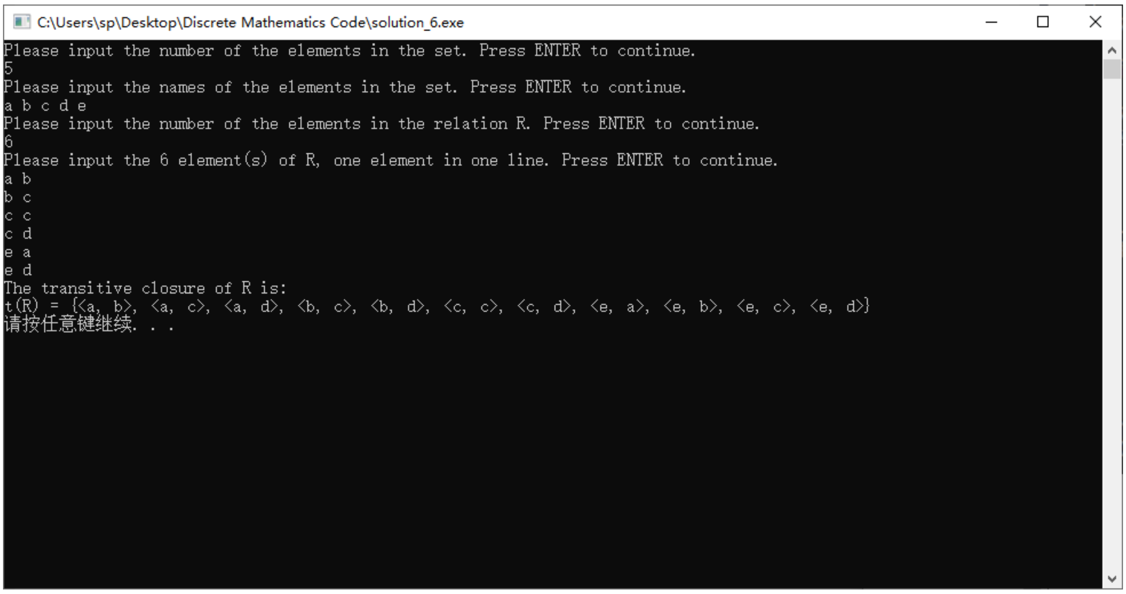

图 9: 实验六程序运行结果

参考文献

[1] ROSEN K H. Discrete mathematics and its applications[M]. New York: McGraw-Hill, 2019.

[2] CORMEN T H, LEISERSON C E, RIVEST R L, et al. Introduction to Algorithms, Third Edition[M]. 3rd. The MIT Press, 2009.

[3] 屈婉玲, 耿素云, 张立昂. 离散数学（第二版）[M]. 北京: 高等教育出版社, 2015.