

《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

1	实验一	2
1.1	问题 A	2
1.1.1	题目简介	2
1.1.2	解题思路	2
1.2	问题 B&C	2
1.2.1	题目简介	2
1.2.2	解题思路	2
1.2.3	核心问题	3
2	实验二	5
2.1	题目简介	5
2.2	解题思路	5
3	实验三	6
3.1	题目简介	6
3.2	解题思路	6
3.3	核心问题	6
4	实验四	7
4.1	题目简介	7
4.2	解题思路	7
4.3	核心问题	7
5	实验五	8
5.1	题目简介	8
5.2	解题思路	8
5.3	核心问题	9
6	实验六	10
6.1	题目简介	10
6.2	解题思路	10
6.3	核心问题	10

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。

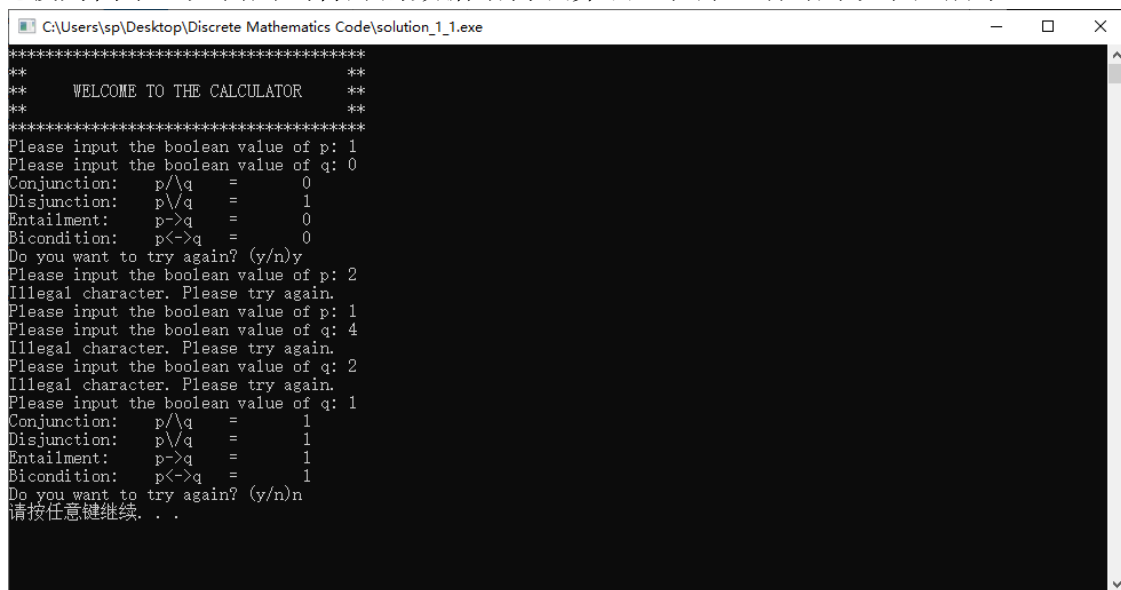
1.1.2 解题思路

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

$$p \rightarrow q \Leftrightarrow \neg p \vee q \quad (1)$$

$$p \leftrightarrow q \Leftrightarrow (\neg p \vee q) \wedge (\neg q \vee p) \quad (2)$$

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。



```
C:\Users\sp\Desktop\Discrete Mathematics Code\solution_1.exe
*****
**                                     **
**      WELCOME TO THE CALCULATOR      **
**                                     **
*****
Please input the boolean value of p: 1
Please input the boolean value of q: 0
Conjunction:  p/\q  =    0
Disjunction:  p\q   =    1
Entailment:   p->q   =    0
Bicondition:  p<->q  =    0
Do you want to try again? (y/n)y
Please input the boolean value of p: 2
Illegal character. Please try again.
Please input the boolean value of p: 1
Please input the boolean value of q: 4
Illegal character. Please try again.
Please input the boolean value of q: 2
Illegal character. Please try again.
Please input the boolean value of q: 1
Conjunction:  p/\q  =    1
Disjunction:  p\q   =    1
Entailment:   p->q   =    1
Bicondition:  p<->q  =    1
Do you want to try again? (y/n)n
请按任意键继续. . .
```

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 $k = 0, 1, \dots, 2^n - 1$ 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

```

1:  $ST \leftarrow$  empty stack
2:  $PS \leftarrow$  empty string
3: for each character  $c \in IS$  do
4:   if  $c$  is a letter then                                ▷ 如果遇到字母, 则直接添加到后缀表达式中
5:      $PS \leftarrow PS + c$ 
6:   else if  $c = '('$  then                                    ▷ 如果遇到左括号, 则将其入栈
7:     Push( $ST, c$ )
8:   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
9:     while not IsEmpty( $ST$ ) and Top( $ST$ )  $\neq '('$  do
10:       $PS \leftarrow PS + \text{Top}(ST)$ 
11:      Pop( $ST$ )
12:    end while
13:    if Top( $ST$ )  $= '('$  then
14:      Pop( $ST$ )
15:    end if
16:  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
17:                                   ▷ 此处为忽略左括号, 我们定义左括号的优先级最小
18:    while not IsEmpty( $ST$ ) and Priority( $c$ )  $\leq$  Priority(Top( $ST$ )) do
19:       $PS \leftarrow PS + \text{Top}(ST)$ 
20:      Pop( $ST$ )
21:    end while
22:  end if
23: end for
24: while not IsEmpty( $ST$ ) do                                ▷ 将栈中剩余字母按顺序添加到后缀表达式中
25:    $PS \leftarrow PS + \text{Top}(ST)$ 
26:   Pop( $ST$ )
27: 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 $0, 1, \dots, n-1$, 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 and 主合取范式。

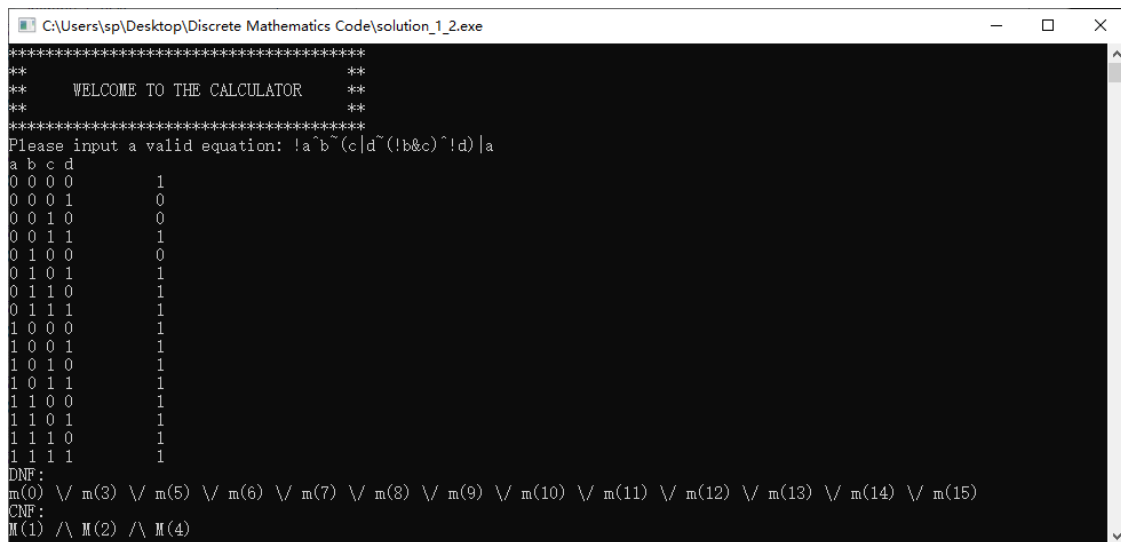
Procedure 2 求解后缀表达式

Input: postfix notation PS , solution S

Output: value v

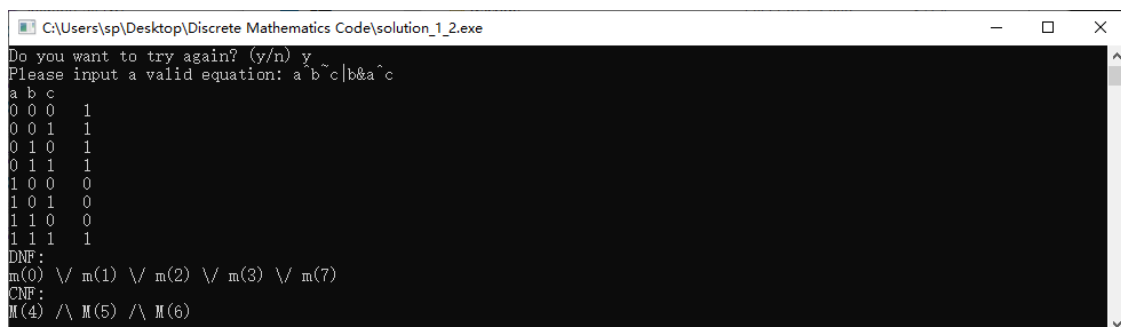
```
1:  $ST \leftarrow$  empty stack
2: for each character  $c \in PS$  do
3:   if  $c$  is a letter then                                ▷ 如果遇到字母，则将其的值入栈
4:     Push( $ST, S[c]$ )
5:   else
6:     if  $c = '!$ ' then                                       ▷ 如果遇到否定运算符，则计算栈顶的值的否定后，将结果入栈
7:        $x \leftarrow$  not Top( $ST$ )
8:       Pop( $ST$ )
9:       Push( $ST, x$ )
10:    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
11:       $y \leftarrow$  Top( $ST$ )
12:      Pop( $ST$ )
13:       $x \leftarrow$  Top( $ST$ )
14:      Pop( $ST$ )
15:      Push(Calculate( $x, c, y$ ))
16:    end if
17:  end if
18: end for
19:  $v \leftarrow$  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。



```
C:\Users\sp\Desktop\Discrete Mathematics Code\solution_1_2.exe
*****
**                               **
**   WELCOME TO THE CALCULATOR   **
**                               **
*****
Please input a valid equation: !a~b~(c|d~(!b&c)!d)|a
a b c d
0 0 0 0      1
0 0 0 1      0
0 0 1 0      0
0 0 1 1      1
0 1 0 0      0
0 1 0 1      1
0 1 1 0      1
0 1 1 1      1
1 0 0 0      1
1 0 0 1      1
1 0 1 0      1
1 0 1 1      1
1 1 0 0      1
1 1 0 1      1
1 1 1 0      1
1 1 1 1      1
DNF:
m(0) ∨ m(3) ∨ m(5) ∨ m(6) ∨ m(7) ∨ m(8) ∨ m(9) ∨ m(10) ∨ m(11) ∨ m(12) ∨ m(13) ∨ m(14) ∨ m(15)
CNF:
M(1) ∧ M(2) ∧ M(4)
```

图 2: 实验一 B (1) 程序运行结果



```
C:\Users\sp\Desktop\Discrete Mathematics Code\solution_1_2.exe
Do you want to try again? (y/n) y
Please input a valid equation: a b~c|b&a^c
a b c
0 0 0      1
0 0 1      1
0 1 0      1
0 1 1      1
1 0 0      0
1 0 1      0
1 1 0      0
1 1 1      1
DNF:
m(0) ∨ m(1) ∨ m(2) ∨ m(3) ∨ m(7)
CNF:
M(4) ∧ M(5) ∧ M(6)
```

图 3: 实验一 B (2) 程序运行结果

```

C:\Users\sp\Desktop\Discrete Mathematics Code\solution_1_2.exe
Do you want to try again? (y/n) y
Please input a valid equation: a b
a b
0 0 1
0 1 0
1 0 0
1 1 1
DNF:
m(0) ∨ m(3)
CNF:
M(1) ∧ M(2)
Do you want to try again? (y/n) y
Please input a valid equation: a b
a b
0 0 1
0 1 1
1 0 0
1 1 1
DNF:
m(0) ∨ m(1) ∨ m(3)
CNF:
M(2)
Do you want to try again? (y/n) y
Please input a valid equation: a|b
a b
0 0 0
0 1 1
1 0 1
1 1 1
DNF:
m(1) ∨ m(2) ∨ m(3)
CNF:
M(0)
Do you want to try again? (y/n) y
Please input a valid equation: a&b
a b
0 0 0
0 1 0
1 0 0
1 1 1
DNF:
m(3)
CNF:
M(0) ∧ M(1) ∧ M(2)
Do you want to try again? (y/n) y

```

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：

$$(A \vee B) \wedge (\neg A \vee C) \wedge (\neg D \vee E) \wedge (D \vee \neg C) \wedge \neg E \quad (4)$$

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。

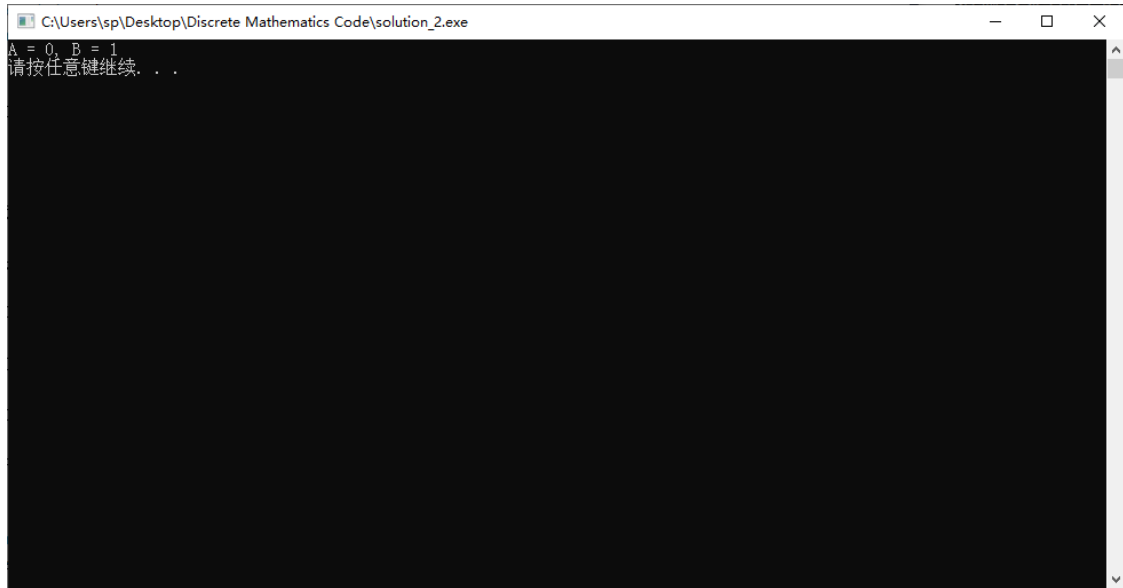


图 5: 实验二程序运行结果

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 (\wedge) 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

$$M_R^* = M_R \wedge M_R^{[2]} \wedge \cdots \wedge M_R^{[n]} \quad (5)$$

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。

```

C:\Users\sp\Desktop\Discrete Mathematics Code\solution_3.exe
Please input the number of the elements in the set: 3
Please input the relation matrix:
Line 0: 0 0 1
Line 1: 1 0 1
Line 2: 0 1 1
-----
1. Reflexive Closure
2. Symmetric Closure
3. Transitive Closure
4. Exit
Select: 1
1 0 1
1 1 1
0 1 1
-----
1. Reflexive Closure
2. Symmetric Closure
3. Transitive Closure
4. Exit
Select: 2
0 1 1
1 0 1
1 1 1
-----
1. Reflexive Closure
2. Symmetric Closure
3. Transitive Closure
4. Exit
Select: 3
1 1 1
1 1 1
1 1 1
-----
1. Reflexive Closure
2. Symmetric Closure
3. Transitive Closure
4. Exit
Select: 4
请按任意键继续. . .

```

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

```

1:  $A = \emptyset$ 
2: for each vertex  $v \in G.V$  do
3:   makeSet( $v$ )
4: end for
5: sort the edges of  $G.E$  into nondecreasing order by weight  $w$ 
6: for each edge  $(u, v) \in G.E$ , taken in nondecreasing order by weight do
7:   if findSet( $u$ )  $\neq$  findSet( $v$ ) then
8:      $A = A \cup \{(u, v)\}$ 
9:     union( $u, v$ )
10:  end if
11: end for

```

4.3 核心问题

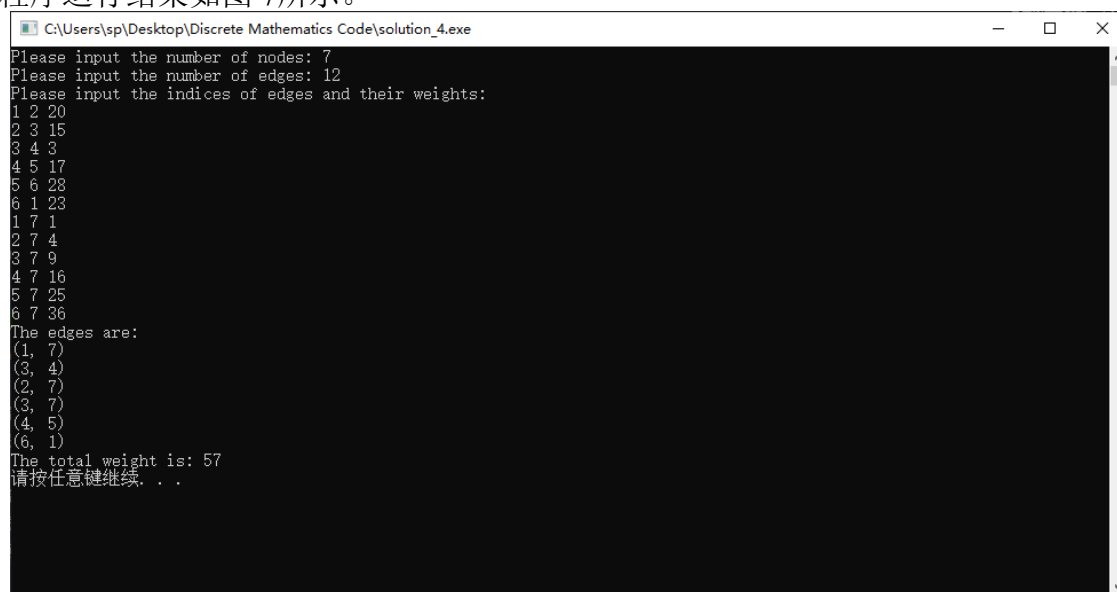
首先，关于将所有的边按照边权进行排序，可以使用一般的快速排序算法。在 C++ 中，使用算法库中的 `std::sort` 即可。这一步的时间复杂度为 $O(m \log m)$ ，其中， m 为图中边的个数。

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。



```

C:\Users\sp\Desktop\Discrete Mathematics Code\solution_4.exe
Please input the number of nodes: 7
Please input the number of edges: 12
Please input the indices of edges and their weights:
1 2 20
2 3 15
3 4 3
4 5 17
5 6 28
6 1 23
1 7 1
2 7 4
3 7 9
4 7 16
5 7 25
6 7 36
The edges are:
(1, 7)
(3, 4)
(2, 7)
(3, 7)
(4, 5)
(6, 1)
The total weight is: 57
请按任意键继续. . .

```

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 $w_i, i = 1, \dots, n$

Output: Huffman tree F

- 1: $F \leftarrow$ forest of n rooted trees, each consisting of the single vertex a_i and assigned weight w_i
 - 2: **while** F is not a tree **do**
 - 3: Replace the rooted trees T and T' of least weights from F with $w(T) \geq w(T')$ with a tree having a new root that has T as its left subtree and T' as its right subtree.
 - 4: Label the new edge to T with 0 and the new edge to T' with 1.
 - 5: Assign $w(T) + w(T')$ as the weight of the new tree.
 - 6: **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。

```
C:\Users\sp\Desktop\Discrete Mathematics Code\solution_5.exe
Please input the number of nodes: 13
Please input the frequencies of these nodes:
2 3 5 7 11 13 17 19 23 29 31 37 41
41      000
37      001
3        0100000
2        0100001
5        010001
7        01001
17       0101
31       011
29       100
13       1010
11       1011
23       110
19       111
请按任意键继续. . .
```

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。

```
C:\Users\sp\Desktop\Discrete Mathematics Code\solution_6.exe
Please input the number of the elements in the set. Press ENTER to continue.
5
Please input the names of the elements in the set. Press ENTER to continue.
a b c d e
Please input the number of the elements in the relation R. Press ENTER to continue.
6
Please input the 6 element(s) of R, one element in one line. Press ENTER to continue.
a b
b c
c c
c d
e a
e d
The transitive closure of R is:
t(R) = {<a, b>, <a, c>, <a, d>, <b, c>, <b, d>, <c, c>, <c, d>, <e, a>, <e, b>, <e, c>, <e, d>}
请按任意键继续. . .
```

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.