

# 云南大学数学与统计学院

## 《运筹学通论实验》上机实践报告

课程名称：运筹学实验	年级：2015 级	上机实践成绩：
指导教师：李建平	姓名：刘鹏	专业：信息与计算科学
上机实践名称：两阶段法求线性规划问题	学号：20151910042	上机实践日期：2018-07-07
上机实践编号：3	组号：	

### 一、实验目的

通过对两阶段法进行编程实现，让自己对单纯形算法理解得更加透彻；  
通过对 MATLAB 的 `linprog` 程序进行调用，学习使用 MATLAB 的优化功能。

### 二、实验内容

写出两阶段法<sup>[1]</sup>的算法；  
用 C 语言<sup>[2]</sup>编程实现两阶段算法。

### 三、实验平台

Microsoft Windows 10 Pro Workstation 1803;  
MathWorks MATLAB R2018a;  
Microsoft Visual Studio 2017 Enterprise.

### 四、算法设计<sup>1</sup>

单纯形算法是一个迭代方法，在每次迭代中，我们的目标是重新整理线性规划，使得基本解有一个更大的目标值。我们选择一个在目标函数中系数为正的的非基本变量 $x_e$ ，而且尽可能增加 $x_e$ 的值而不违反任何约束。变量 $x_e$ 称为基本变量，并且某个其他变量 $x_l$ 变成非基本变量。

**Algorithm**      **SIMPLEX**, Simplex Method for solving LP problems can start with 0.

**Input**            (1) 系数矩阵 $A = (a_{ij})_{m \times n}$ ,  $A = (A_1, A_2, \dots, A_m)$ ,  $A_i$ 是系数矩阵的第 $i$ 列;  
                  (2) 价值向量 $c = (c_{ij})_{n \times 1} = (c_1, c_2, \dots, c_n)$ ;  
                  (3) 常数向量 $b = (b_{ij})_{m \times 1} = (b_1, b_2, \dots, b_m) \geq 0$ ;  
                  It means to find  $\text{MAX}(cx)$ , s.t.  $Ax = b$ , and  $x \geq 0$ .

**Output**            如果有最优解，输出最优解 $x$ ，如果没有，输出 No Solution  
**Begin**

---

<sup>1</sup> 此处的伪代码中，矩阵运算符的意义均与 MATLAB 语言一致，如矩阵的左除、右除和点除等。

**Step 1**  $A' = (-c^T, A^T)^T = (a'_{ij})_{(m+1) \times n};$

**Step 2**  $b' = (0, b^T)^T = ((b'_{ij})_{(m+1) \times 1});$

**Step 3**  $A'' = (A', b') = (a''_{ij})_{(m+1) \times (n+1)};$

**Step 4** 记 $A''$ 的第一行为 $A_0$ ;

**Step 5** 从 $-A_0$ 中价值元素中的找寻最大的正数，命之为Pivot，记之在此行中的坐标为 $C$ ，GOTO **Step 6**; 如果经过标记之后回到这里而且找不到正数，说明循环结束，输出 $z = A''(1, n+1)$ ,  $x = A''(:, n+1)^T$ , GOTO **End**.

如果在所有价值元素中找不到正数，说明这可能是通过变化 MIN 类型的价值向量得来的，如果原先的约束都是小于等于，那么毫无疑问 $0$ 就是最优解，GOTO **End**; 如果原先的约束有大于等于约束与等于约束或者其中之一，那么很显然 $0$ 并不行，因为它要求松弛变量与剩余变量的最终解都为 $0$ ，这是荒谬的，也就是说 $0$ 并不行，还需要进行如下处理：在矩阵的第一行中，消去所有的非零变量，即通过基本行变换将那些负数变为 $0$ ，这样更新过的元素中就有正数了（因为变零的操作是加法，如果所有变换都产生不了正数，那说明只有 $0$ 这个不符合题意的解能单纯地满足约束方程），表现为 $A''$ 的第一行有负数， $\text{MIN}(A_0) < 0$ ，从 $-A_0$ 中价值元素中的找寻最大的正数，命之为Pivot，记之在此行中的坐标为 $C$ ，设置本次迭代不在经过此步骤，否则会有死循环，GOTO **Step 6**;

**Step 6**

利用条件除法作集合 $S = \{b_i/a_{i,C} \mid a_{i,C} > 0, i = 2, 3, \dots, m+1\}$ ,  $t = \text{MAX}(S)$ ，最大值不止一个就选其中一个，记 $t$ 在 $A''$ 中的坐标为 $(R, C)$ ，GOTO **Step 7**

**Step 7**

$A''(R, :) = A''(R, :)/A''(R, C);$

for  $i$  through 1 to  $m+1$

if  $i \neq R$ , then let  $A''(i, :) = A''(i, :) - A''(i, C)/A''(R, C)$

Go to **Step 5**;

**End**

下面是调用了上面的单纯形算法的两阶段算法。

**Algorithm** **DUAL-SIMPLEX**, 重点解决初值不能从 $0$ 开始的 LP 问题

**Input**

- (1) 系数矩阵 $A = (a_{ij})_{m \times n}$ ,  $A = (A_1, A_2, \dots, A_m)$ ,  $A_i$ 是系数矩阵的第 $i$ 列;
- (2) 价值向量 $c_1 = (c_{ij})_{n \times 1} = (c_1, c_2, \dots, c_n)$ ;
- (3) 价值向量 $c_2 = (c_{ij})_{n \times 1} = (c_1, c_2, \dots, c_n)$ ;
- (4) 常数向量 $b = (b_{ii})_{m \times 1} = (b_1, b_2, \dots, b_m) \geq 0$ ;

- Output**           如果有最优解，输出最优解 $\mathbf{x}$ ，如果没有，输出 No Solution
- Begin**
- Step 1**           **SIMPLEX**( $\mathbf{A}$ ,  $\mathbf{c}_1$ ,  $\mathbf{b}$ ), 输出 $z = \mathbf{A}''(1, n+1)$ , 此时如果 $z = 0$ , GOTO **Step 2**, 若不为零, 则输出无解, GOTO **End**;
- Step 2**           重新初始化单纯形表以为第二阶段做准备: 把 $\mathbf{A}''$ 中人工变量所在的行全部删除或者全部变为零, 把第一行用 $(-\mathbf{c}_2, 0)$ 代替; GOTO **Step 3**;
- Step 3**           将 $\mathbf{A}''$ 传给 **SIMPLEX** 算法的 **Step 4**, 完成计算。
- End**

## 五、 程序代码

### 5.1 程序描述

为了完成此项目，我创建了很多新的类，有读取类，动态数组类，矩阵类等，因为是针对单纯形算法进行量身打造，所以拓展性一般，在后来的几个新的项目里，我又在稍加修改的基础上复用了这几个类，并且针对其中几个可以通用的类进行了优化，比如加深抽象层次使之可以作为泛型容器。回过头来再看原先的代码并不足够优秀。但是时间有限，提交在即，这里就不再进行代码优化升级或者某些地方的重构了，毕竟这个程序做了大量的测试，并没有致命的 bug，特此说明。

因为程序很长，文件很多，这里就仅仅列出核心文件 Simplex.h 中的代码，其余的代码在附录中给出。本来头文件里不应该写实现，但是这里的程序不需要封装，所以实现与定义放在一起呈现。

### 5.2 程序代码

```

1  /*
2  * Copyright (c) 2018, Liu Peng, School of Mathematics and Statistics, YNU
3  * Apache License.
4  *
5  * 文件名称: Simplex.h
6  * 文件标识: 见配置管理计划书
7  * 摘 要: 对标准输入的单纯形问题进行求解
8  *
9  * 当前版本: 1.0
10 * 作 者: 刘鹏
11 * 创建日期: 2018 年 5 月 4 日
12 * 完成日期: 2018 年 5 月
13 *
14 * 取代版本:
15 * 原作者 : 刘鹏
16 * 完成日期:
17 */
18
19 #pragma once

```

```

20
21 #include "Matrix_Operation.h"
22 #include "Divide.h"
23
24 // Generally speaking, this data structure is not a table.
25 // whatever, it works.
26 typedef struct Simplex_Tableau {
27     Matrix *Matrix;
28     Dynamic_Array *Objective_Vector;
29     Dynamic_Array *b;
30 } Simplex_Tableau;
31
32 // Initialize the table of simplex method.
33 // This is a simple implementation, only can solve problems like "Ax = b"
34 // with all the slack variables has been added.
35 Simplex_Tableau *Simplex_Tableau_init(char *c, char *A, char *b) {
36     Simplex_Tableau *ans = (Simplex_Tableau *)calloc(1, sizeof(Simplex_Tableau));
37
38     Matrix *m = get_Matrix(A);
39     ans->Objective_Vector = get_Dynamic_Array(c);
40     ans->b = get_Dynamic_Array(b);
41
42     int i = 0;
43     int j = 0;
44     Dynamic_Array *tmp = Dynamic_Array_init();
45
46     // STEP 1: Append the zero'th row.
47     for (i = 1; i <= ans->Objective_Vector->n; i++) {
48         Dynamic_Array_append(tmp, -1 * Dynamic_Array_get_Element(ans->Objective_Vector, i));
49     }
50     Dynamic_Array_append(tmp, 0);
51
52     // STEP 2: Append the Coefficient Matrix.
53     for (i = 1; i <= m->n_row; i++) {
54         for (j = 1; j <= m->n_column; j++) {
55             Dynamic_Array_append(tmp, Matrix_get_Element(m, i, j));
56         }
57         Dynamic_Array_append(tmp, Dynamic_Array_get_Element(ans->b, i));
58     }
59
60     ans->Matrix = Matrix_init(m->n_row + 1, m->n_column + 1);
61     ans->Matrix->low_level_array = tmp->A;
62
63     return ans;
64 }
65
66 // c2 is needed in the second phase.
67 // The Objective Vector need to be changed. reshape the matrix.

```

```

68 Simplex_Tableau *Simplex_Tableau_re_init(Simplex_Tableau *S, char *c2) {
69     int i = 1;
70     for (; i <= S->Objective_Vector->n; i++) {
71         double tmp = Dynamic_Array_get_Element(S->Objective_Vector, i);
72         if (tmp > 1e-15 || tmp < -1e-15) {
73             Matrix_column_to_zero(S->Matrix, i);
74         }
75     }
76
77     Dynamic_Array *New_Objective_Vector = get_Dynamic_Array(c2);
78     S->Objective_Vector = New_Objective_Vector;
79
80     for (i = 0; i < New_Objective_Vector->n; i++) {
81         // Cover the old value
82         *(S->Matrix->low_level_array + i) = Dynamic_Array_get_Element(New_Objective_Vector, i +
1);
83     }
84     Matrix_num_mul_vector(-1, S->Matrix, 1);
85
86     return S;
87 }
88
89 // Iterations for simplex method.
90 void Simplex(Simplex_Tableau *S) {
91
92     // Pre-print the original Matrix.
93     Matrix_print(S->Matrix);
94
95     // Checking the Problem type.
96     int i = 1;
97     int count_minus = 0;
98     for (; i <= S->Objective_Vector->n; i++) {
99         double tmp = Dynamic_Array_get_Element(S->Objective_Vector, i);
100         if (tmp < 1e-15 || tmp == 0) {
101             count_minus += 1;
102         }
103     }
104     if (count_minus == S->Objective_Vector->n) {
105         printf("This Linear Programming MAYBE a <MIN> type\n");
106         printf("Simplex Matrix will be reshaped.\n");
107         for (i = 1; i <= S->Objective_Vector->n; i++) {
108             if (Dynamic_Array_get_Element(S->Objective_Vector, i) != 0) {
109                 Matrix_pivot_Element_transInto_zero(S->Matrix, 1, i);
110                 Matrix_print(S->Matrix);
111             }
112         }
113     }
114 }

```

```

115     int iter_deepth = 1;
116
117     Dynamic_Array *object = Matrix_row_to_Vector(S->Matrix, 1, -1);
118     int N_pivot_column = Dynamic_Array_find_Maximal(object);
119     double Max = Dynamic_Array_get_Element(object, N_pivot_column);
120
121     Dynamic_Array *pivot_column = Matrix_column_to_Vector(S->Matrix, N_pivot_column);
122     Dynamic_Array *last_column = Matrix_column_to_Vector(S->Matrix, S->Matrix->n_column);
123
124     Div_Dynamic_Array *tmp = Div_Dynamic_Array_init(last_column, pivot_column);
125     int N_pivot_row = Div_Dynamic_Array_find_Minimal(tmp);
126
127     Matrix_pivot_Element_Trans(S->Matrix, N_pivot_row, N_pivot_column);
128
129     printf("Iter depth: %d\n", iter_deepth++);
130     Matrix_print(S->Matrix);
131
132     object = Matrix_row_to_Vector(S->Matrix, 1, -1);
133     N_pivot_column = Dynamic_Array_find_Maximal(object);
134     Max = Dynamic_Array_get_Element(object, N_pivot_column);
135
136     while (Max > 0 && iter_deepth <=10000) {
137
138         pivot_column = Matrix_column_to_Vector(S->Matrix, N_pivot_column);
139         last_column = Matrix_column_to_Vector(S->Matrix, S->Matrix->n_column);
140
141         tmp = Div_Dynamic_Array_init(last_column, pivot_column);
142         N_pivot_row = Div_Dynamic_Array_find_Minimal(tmp);
143
144         Matrix_pivot_Element_Trans(S->Matrix, N_pivot_row, N_pivot_column);
145         printf("Iter depth: %d\n", iter_deepth);
146         Matrix_print(S->Matrix);
147         object = Matrix_row_to_Vector(S->Matrix, 1, -1);
148         N_pivot_column = Dynamic_Array_find_Maximal(object);
149         Max = Dynamic_Array_get_Element(object, N_pivot_column);
150
151         iter_deepth += 1;
152     }
153 }
154
155 // For finding a initial solution, dual simplex method is needed.
156 // The input may be a little bit complex.
157 void dual_Simplex(Simplex_Tableau *S, char *c2) {
158
159     // First Phase
160     S->Objective_Vector = Matrix_row_to_Vector(S->Matrix, 1, -1);
161     Simplex(S);
162     if (Matrix_get_Element(S->Matrix, 1, S->Matrix->n_column) > 1e-14) {

```

```
163     printf("ANSWER of PHRASE ONE: %.4f\n", Matrix_get_Element(S->Matrix, 1, S->Ma-
trix->n_column));
164     printf("No Feasible Solution.\n");
165     return;
166 }
167 // else
168
169 printf("First Phase completed.\n\n");
170 Simplex_Tableau_re_init(S, c2);
171 printf("NEW Objective Function is ");
172 Dynamic_Array_print(S->Objective_Vector);
173 printf("\n");
174 Simplex(S);
175 }
```

程序代码 1

六、 运行结果

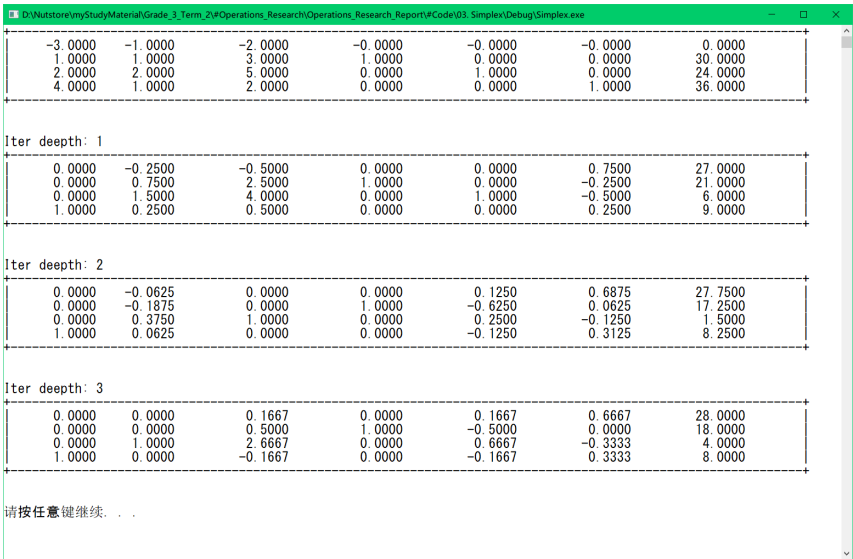
6.1 例 1

这个例子来自于算法导论<sup>[3]</sup>的 29.3 节  
最大化  
满足约束

$$\max 3x_1 + x_2 + 2x_3$$

$$\begin{aligned} x_1 + x_2 + 3x_3 &\leq 30 \\ 2x_1 + 2x_2 + 5x_3 &\leq 24 \\ 4x_1 + x_2 + 2x_3 &\leq 36 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

6.1.1 运行结果



The screenshot shows a window titled "Simplex.exe" displaying the Simplex algorithm's progress. It includes the initial tableau, three iterations (Iter depth: 1, 2, 3), and the final optimal solution. The final tableau shows the maximum value of 28.0000.

Iteration	Row 1	Row 2	Row 3	Row 4	Row 5	Row 6	Row 7
Initial	-3.0000	-1.0000	-2.0000	-0.0000	-0.0000	-0.0000	0.0000
	1.0000	1.0000	3.0000	1.0000	0.0000	0.0000	30.0000
	2.0000	2.0000	5.0000	0.0000	1.0000	0.0000	24.0000
	4.0000	1.0000	2.0000	0.0000	0.0000	1.0000	36.0000
Iter depth: 1	0.0000	-0.2500	-0.5000	0.0000	0.0000	0.7500	27.0000
	0.0000	0.7500	2.5000	1.0000	0.0000	-0.2500	21.0000
	0.0000	1.5000	4.0000	0.0000	1.0000	-0.5000	6.0000
	1.0000	0.2500	0.5000	0.0000	0.0000	0.2500	9.0000
Iter depth: 2	0.0000	-0.0625	0.0000	0.0000	0.1250	0.6875	27.7500
	0.0000	-0.1875	0.0000	1.0000	-0.6250	0.0625	17.2500
	0.0000	0.3750	1.0000	0.0000	0.2500	-0.1250	1.5000
	1.0000	0.0625	0.0000	0.0000	-0.1250	0.3125	8.2500
Iter depth: 3	0.0000	0.0000	0.1667	0.0000	0.1667	0.6667	28.0000
	0.0000	0.0000	0.5000	1.0000	-0.5000	0.0000	18.0000
	0.0000	1.0000	2.6667	0.0000	0.6667	-0.3333	4.0000
	1.0000	0.0000	-0.1667	0.0000	-0.1667	0.3333	8.0000

运行结果 1

### 6.1.2 代码分析

例子是算法导论<sup>[3]</sup>给出的一个例子，并没有采用单纯形两阶段法，因为都是小于等于的约束，所以初值很好取。

### 6.2 例 2

这个例子来自运筹学导论<sup>[1]</sup>两阶段的章节。

最小化

$$0.4x_1 + 0.5x_2$$

满足

$$0.3x_1 + 0.1x_2 \leq 2.7$$

$$0.5x_1 + 0.5x_2 \leq 6$$

$$0.6x_1 + 0.4x_2 \geq 6$$

$$x_1, x_2 \geq 0$$

因为存在非小于等于约束，所以先要把这些约束转化为等式约束

第一阶段问题

$$\min Z = \overline{x_4} + \overline{x_6}$$

s.t.

$$0.3x_1 + 0.1x_2 + x_3 = 2.7$$

$$0.5x_1 + 0.5x_2 + \overline{x_4} = 6$$

$$0.6x_1 + 0.4x_2 - x_5 + \overline{x_6} = 6$$

第二阶段问题

$$\min Z = 0.4x_1 + 0.5x_2$$

s.t.

$$0.3x_1 + 0.1x_2 + x_3 = 2.7$$

$$0.5x_1 + 0.5x_2 = 6$$

$$0.6x_1 + 0.4x_2 - x_5 = 6$$



6.2.1 运行结果

```
D:\Nutstore\myStudyMaterial\Grade_3_Term_2\Operations_Research\Operations_Research_Report\#Code\03. Simplex\Debug\Simplex.exe
+-----+
| -0.0000  -0.0000  -0.0000  1.0000  -0.0000  1.0000  0.0000 |
| 0.3000   0.1000   1.0000   0.0000   0.0000   0.0000  2.7000 |
| 0.5000   0.5000   0.0000   1.0000   0.0000   0.0000  6.0000 |
| 0.6000   0.4000   0.0000   0.0000  -1.0000   1.0000  6.0000 |
+-----+

This Linear Programming MAYBE a <MIN> type
Simplex Matrix will be reshaped.
+-----+
| -0.5000  -0.5000  -0.0000  0.0000  -0.0000  1.0000  -6.0000 |
| 0.3000   0.1000   1.0000   0.0000   0.0000   0.0000  2.7000 |
| 0.5000   0.5000   0.0000   1.0000   0.0000   0.0000  6.0000 |
| 0.6000   0.4000   0.0000   0.0000  -1.0000   1.0000  6.0000 |
+-----+

+-----+
| -1.1000  -0.9000  -0.0000  0.0000   1.0000   0.0000  -12.0000 |
| 0.3000   0.1000   1.0000   0.0000   0.0000   0.0000  2.7000 |
| 0.5000   0.5000   0.0000   1.0000   0.0000   0.0000  6.0000 |
| 0.6000   0.4000   0.0000   0.0000  -1.0000   1.0000  6.0000 |
+-----+

Iter depth: 1
+-----+
| 0.0000  -0.5333   3.6667   0.0000   1.0000   0.0000  -2.1000 |
| 1.0000   0.3333   3.3333   0.0000   0.0000   0.0000  9.0000 |
| 0.0000   0.3333  -1.6667   1.0000   0.0000   0.0000  1.5000 |
| 0.0000   0.2000  -2.0000   0.0000  -1.0000   1.0000  0.6000 |
+-----+

Iter depth: 2
+-----+
| 0.0000   0.0000  -1.6667   0.0000  -1.6667   2.6667  -0.5000 |
| 1.0000   0.0000   6.6667   0.0000   1.6667  -1.6667   8.0000 |
| 0.0000   0.0000   1.6667   1.0000   1.6667  -1.6667   0.5000 |
| 0.0000   1.0000  -10.0000   0.0000  -5.0000   5.0000   3.0000 |
+-----+

Iter depth: 3
+-----+
| 0.0000   0.0000   0.0000   1.0000   0.0000   1.0000   0.0000 |
| 1.0000   0.0000   0.0000  -4.0000  -5.0000   5.0000   6.0000 |
| 0.0000   0.0000   1.0000   0.6000   1.0000  -1.0000   0.3000 |
| 0.0000   1.0000   0.0000   6.0000   5.0000  -5.0000   6.0000 |
+-----+

First Phase completed.
NEW Objective Function is  -0.40      -0.50      0.00      0.00      0.00      0.00

+-----+
| 0.4000   0.5000  -0.0000  -0.0000  -0.0000  -0.0000  -0.0000 |
| 1.0000   0.0000   0.0000   0.0000  -5.0000   0.0000   6.0000 |
| 0.0000   0.0000   1.0000   0.0000   1.0000   0.0000   0.3000 |
| 0.0000   1.0000   0.0000   0.0000   5.0000   0.0000   6.0000 |
+-----+

This Linear Programming MAYBE a <MIN> type
Simplex Matrix will be reshaped.
+-----+
| 0.0000   0.5000  -0.0000  -0.0000   2.0000  -0.0000  -2.4000 |
| 1.0000   0.0000   0.0000   0.0000  -5.0000   0.0000   6.0000 |
| 0.0000   0.0000   1.0000   0.0000   1.0000   0.0000   0.3000 |
| 0.0000   1.0000   0.0000   0.0000   5.0000   0.0000   6.0000 |
+-----+

+-----+
| 0.0000   0.0000  -0.0000  -0.0000  -0.5000  -0.0000  -5.4000 |
| 1.0000   0.0000   0.0000   0.0000  -5.0000   0.0000   6.0000 |
| 0.0000   0.0000   1.0000   0.0000   1.0000   0.0000   0.3000 |
| 0.0000   1.0000   0.0000   0.0000   5.0000   0.0000   6.0000 |
+-----+

Iter depth: 1
+-----+
| 0.0000   0.0000   0.5000   0.0000   0.0000   0.0000  -5.2500 |
| 1.0000   0.0000   5.0000   0.0000   0.0000   0.0000   7.5000 |
| 0.0000   0.0000   1.0000   0.0000   1.0000   0.0000   0.3000 |
| 0.0000   1.0000  -5.0000   0.0000   0.0000   0.0000   4.5000 |
+-----+

请按任意键继续. . .
```

运行结果 2

### 6.2.2 代码分析

通过程序，得出了正确的答案。

## 七、 实验体会

这个单纯形算法并不是真正基于矩阵的算法。比较遗憾。

## 八、 参考文献

- [1] HILLIER F S, LIEBERMAN G J. 运筹学导论 [M]. 9th ed. 北京: 清华大学出版社, 2010.
- [2] 林锐. 高质量 C++/C 编程指南 [M]. 1.0 ed., 2001.
- [3] CORMEN T H, LEISERSON C E, RIVEST R L, et al. 算法导论 [M]. 3rd ed. 北京: 机械工业出版社, 2013.