

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：求给定序列的最小值及所有最小值的下标	学号：20151910042	上机实践日期：2018-07-07
上机实践编号：1	组号：	

一、实验目的

完成该实验，为后期的更进一步的实验做准备。

二、实验内容

给定两组数 $\mathbf{a} = (a_1, a_2, \dots, a_n)$ 和 $\mathbf{b} = (b_1, b_2, \dots, b_n)$ ，求

1. 一组数 $\mathbf{c} = (c_1, c_2, \dots, c_n)$ ，其中

$$c_i = \begin{cases} \frac{a_i}{b_i}, & b_i \neq 0 \\ \text{NaN}, & b_i = 0 \end{cases}, \quad i = 1, 2, \dots, n.$$

2. 求最小值及所有最小值的下标，其中最小值为

$$\min \left\{ \frac{a_i}{b_i} \mid b_i > 0, \quad i = 1, 2, \dots, n. \right\}$$

三、实验平台

Windows 10 Pro 1703;

Microsoft Visual Studio 2017 Enterprise。

四、算法设计¹

Algorithm: DIV, find the minimal value and all its(their) indexes

Input: $\mathbf{a} = (a_1, a_2, \dots, a_n)$ and $\mathbf{b} = (b_1, b_2, \dots, b_n)$.

Output: list \mathbf{c} , minimal value pivot and their indexes.

Begin

Step 1: DEFINE INT / 0 = NaN

Step 2: $\mathbf{c} = \mathbf{a} ./ \mathbf{b}$

Step 3: \mathbf{C} is a set who contains all the elements in \mathbf{c} whose $b_i > 0$

Step 4: sort \mathbf{C} incrementally, let pivot the first element of the sorted \mathbf{C}

Step 5: find all elements in \mathbf{c} whose $b_i > 0$ and equal to pivot then put their indexes into **Index**.

End

¹ 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。

五、程序代码

1.1 程序描述

这个解释程序的使用方法是这样的：在 shell 中通过调用本可执行程序 div，输入两个字符串参数，然后程序自动输出 c 与最小值及其所有位置。如下所示：（这里隐藏了 PowerShell 的工作目录，仅用 PS > 作为提示符）

```
PS > .\div.exe "( -3.14,20 ,-256, 0 ,6,5,12121,4588, 89)" "(3.14, -1, 256,3.2222,2,0,5633.2,168,78)"
argument 1 is
(-3.14, 20.00, -256.00, 0.00, 6.00, 5.00, 12121.00, 4588.00, 89.00)

argument 2 is
(3.14, -1.00, 256.00, 3.22, 2.00, 0.00, 5633.20, 168.00, 78.00)

The answer C = (-1.00 , -20.00 , -1.00 , 0.00 , 3.00 , NaN , 2.15 , 27.31 , 1.14 )
Minimal Value is -1.00 , position is (1.00, 3.00)

PS >
```

因为并没有 shell 接口，所以基本上是自己写一个 shell 来做这个与机器的交互。首先是清洗，把两个字符串进行 clean 重整，去除可能的空格之后，第一步是跳过第一个圆括号，同时把最后的圆括号变为逗号。这样一来就好多了，一个 double 数值跟着一个逗号。（这里都是对一个字符串来说的，毕竟解释得了一个就能解释两个。）

第二步就是分割，把这个字符串当作一块“长条豆腐”，每次从头部切一部分下来，直到切光。头部已经是处理好的了，所以一直切到遇到的第一个逗号，这个过程把逗号之前的字符，即可能出现的负号与小数点进行分类处理：负号直接跳过，最后乘-1；单个的数字与小数点直接归入队列，与此同时，队列的头号元素，跟随着一个从1开始的索引，该索引按照增序排到队列的末尾——遇到的第一个分号。如此之后，可以通过遍历一次，找到小数点所在位置对应的索引，然后利用对称的坐标变换公式，把其他数字符号与小数点的距离转化为10的指数，然后通过 pow 函数算出具体的数值，完成字符到数值的转化。

在整个过程中要注意保护头指针与 work 指针的归位。一个数字一旦算出，就交给动态数组保存。整个字符串的切割，一直做到\0。这个过程一直中，一直保持着保存操作。当解释程序返回一个浮点数就要存入，返回 NULL 就结束归入。当遇到\0之后，也就得到了一个存有输入信息的双精度数组。

拿到了两个动态数组之后，就可以做除法、排序与查找了。

1.2 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: Source.cpp
```

```

6  * 文件标识: 见配置管理计划书
7  * 摘 要: Prim 算法
8  *
9  * 当前版本: 1.0
10 * 作 者: 刘鹏
11 * 创建日期: 2018 年 3 月 14 日
12 * 完成日期: 2018 年 6 月 25 日
13 *
14 * 取代版本:
15 * 原作者 : 刘鹏
16 * 完成日期:
17 */
18
19 /*
20 * A function like division which can execute with some conditions.
21 */
22
23 #include<stdio.h>
24 #include<stdlib.h>
25 #include<string.h>
26 #include<math.h>
27
28 // The following type is a container for creating a stack.
29 typedef struct char_LinkedList {
30     char_LinkedList *head;
31     char elements;      // partition must be integer less than 10
32     int times;          // container
33     char_LinkedList *next;
34 }char_LinkedList;
35
36 typedef struct Dynamic_Array {
37     double *A;          // low-level array
38     int capacity;       // the capacity
39     int n;              // used room
40 }Dynamic_Array;
41
42 typedef struct Div {
43     double up;
44     double down;
45     double value;
46     char state[10];     // NaN or Negative or Normal
47 }Div;
48
49 typedef struct Div_Dynamic_Array {
50     Div *A;             // 底层结构体数组的头指针, 不能动!
51     int capacity;       // 底层结构体数组的容量
52     int n;              // 底层数组的占用量
53 }Div_Dynamic_Array;
54

```

```

55 void Div_Resize(Div_Dynamic_Array *D) {
56     int i = 0;
57     Div *tmp = (Div *)calloc(2 * D->capacity, sizeof(Div));
58     if (tmp == NULL) {
59         printf("Cannot get memory, crash!\n");
60         return;
61     }
62     for (i = 0; i < D->capacity; i++) {
63         (tmp + i)->up = (D->A + i)->up;
64         (tmp + i)->down = (D->A + i)->down;
65         (tmp + i)->value = (D->A + i)->value;
66         strcpy((tmp + i)->state, (D->A + i)->state);
67     }
68     free(D->A);
69     D->A = tmp;
70     tmp = NULL;
71
72     D->capacity *= 2;
73 }
74
75 void Div_Append(Div_Dynamic_Array *D, Div e) {
76     if (D->n == D->capacity) {
77         Div_Resize(D);
78     }
79     (D->A + D->n)->up = e.up;
80     (D->A + D->n)->down = e.down;
81     (D->A + D->n)->value = e.value;
82     strcpy((D->A + D->n)->state, e.state);
83     D->n += 1;
84     //int i;
85     //for (i = 0; i <= D->n; i++) {
86     //    printf("%s\t", (D->A + i)->state);
87     //}
88     //printf("\n");
89 }
90
91 void Div_print(Div_Dynamic_Array *d) {
92     int i;
93     printf("The answer C = (");
94     for (i = 0; i < d->n; i++) {
95         if (!strcmp((d->A + i)->state, "NaN")) {
96             printf("%s ", "NaN");
97         }
98         else {
99             double value = (d->A + i)->value;
100             printf("%2.2f ", value);
101         }
102         if (i == d->n - 1) {
103             printf("");

```

```
104     }
105     else {
106         printf(", ");
107     }
108 }
109 printf("\n");
110 }
111
112 void Div_onArray(Dynamic_Array *a, Dynamic_Array *b, Div_Dynamic_Array *ans) {
113     if (a->n != b->n) {
114         printf("length should be the same.");
115         return;
116     }
117
118     int i;
119     for (i = 0; i < a->n; i++) {
120         if (*(b->A + i) == 0) {
121             Div tmp;
122             tmp.up = NULL;
123             tmp.down = NULL;
124             tmp.value = NULL;
125             char c[] = "NaN";
126             strcpy(tmp.state, c);
127             Div_Append(ans, tmp);
128         }
129         else {
130             if (*(b->A + i) < 0.) {
131                 Div tmp;
132                 tmp.up = *(a->A + i);
133                 tmp.down = *(b->A + i);
134                 tmp.value = tmp.up / tmp.down;
135                 char c[] = "Negative";
136                 strcpy(tmp.state, c);
137                 Div_Append(ans, tmp);
138             }
139             else {
140                 Div tmp;
141                 tmp.up = *(a->A + i);
142                 tmp.down = *(b->A + i);
143                 tmp.value = tmp.up / tmp.down;
144                 char c[] = "Normal";
145                 strcpy(tmp.state, c);
146                 Div_Append(ans, tmp);
147             }
148         }
149     }
150 }
151
152 // output a Double array
```

```

153 void print(int n, Dynamic_Array *d) {
154     printf(/* "argument %d is \n*/ "(");
155     int i;
156     for (i = 0; i < d->n - 1; i++) {
157         printf("%2.2f, ", *(d->A + i));
158     }
159     printf("%2.2f", *(d->A + i));
160     printf(")\n\n");
161 }
162
163 // Output a double-array with integer format
164 void print_int(int n, Dynamic_Array *d) {
165     printf(/* "argument %d is \n*/ "(");
166     int i;
167     for (i = 0; i < d->n - 1; i++) {
168         printf("%2.0f, ", *(d->A + i));
169     }
170     printf("%2.0f", *(d->A + i));
171     printf(")\n\n");
172 }
173
174 void Resize(Dynamic_Array *D) {
175     int i = 0;
176     double *tmp = (double *)calloc(2 * D->capacity, sizeof(double));
177     if (tmp == NULL) {
178         printf("Cannot get memory, crash!\n");
179         return;
180     }
181     for (i = 0; i < D->capacity; i++) {
182         *(tmp + i) = *(D->A + i);
183     }
184     D->A = tmp;
185     D->capacity *= 2;
186 }
187
188 void Append(Dynamic_Array *D, double e) {
189     if (D->n == D->capacity) {
190         Resize(D);
191     }
192     *(D->A + D->n) = e;
193     D->n += 1;
194 }
195
196 Dynamic_Array *Quick_sort(Dynamic_Array *a) {
197
198     Dynamic_Array *less = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
199     less->A = (double *)calloc(1, sizeof(double));
200     if (!less) {
201         printf("Can't get memory!");

```

```

202     return NULL;
203 }
204 less->capacity = 1;
205 less->n = 0;
206
207 Dynamic_Array *more = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
208 more->A = (double *)calloc(1, sizeof(double));
209 if (!more) {
210     printf("Can't get memory!");
211     return NULL;
212 }
213 more->capacity = 1;
214 more->n = 0;
215
216 Dynamic_Array *eq = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
217 eq->A = (double *)calloc(1, sizeof(double));
218 if (!eq) {
219     printf("Can't get memory!");
220     return NULL;
221 }
222 eq->capacity = 1;
223 eq->n = 0;
224
225 int i;
226 if (a->n <= 1) {
227     return a;
228 }
229 else {
230     for (i = 0; i < a->n; i++) {
231         double pivot = *(a->A);
232         if (*(a->A + i) > pivot) {
233             Append(more, *(a->A + i));
234         }
235         else {
236             if (*(a->A + i) < pivot) {
237                 Append(less, *(a->A + i));
238             }
239             else {
240                 Append(eq, *(a->A + i));
241             }
242         }
243     }
244 }
245 less = Quick_sort(less);
246 more = Quick_sort(more);
247 for (i = 0; i < eq->n; i++) {
248     Append(less, *(eq->A + i));
249 }
250 for (i = 0; i < more->n; i++) {

```

```

251     Append(less, *(more->A + i));
252 }
253 return less;
254 }
255
256 void find(Div_Dynamic_Array *a) {
257
258     Dynamic_Array *c = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
259     Dynamic_Array *d = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
260     c->A = (double *)calloc(a->n, sizeof(double));
261     if (c == NULL || d == NULL || c->A == NULL) {
262         printf("Can't get memory!\n");
263         return;
264     }
265     c->capacity = a->n;
266     c->n = 0;
267
268     int i = 0;
269     for (i = 0; i < a->n; i++) {
270
271         // denominator is legal
272         if (!strcmp((a->A + i)->state, "Normal")) {
273             Append(c, (a->A + i)->value);
274         }
275     }
276     d = Quick_sort(c);
277
278     double pivot = *(d->A + 0);
279     Dynamic_Array *tmp = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
280     tmp->A = (double *)calloc(1, sizeof(double));
281     if (tmp == NULL || tmp->A == NULL) {
282         printf("Can't get memory!\n");
283         return;
284     }
285     tmp->capacity = 1;
286     tmp->n = 0;
287     for (i = 0; i < a->n; i++) {
288         if (!strcmp((a->A + i)->state, "Normal") && (a->A + i)->value == pivot) {
289             Append(tmp, ++i);
290         }
291     }
292     if (tmp->n == 0) {
293         printf("Sorry, no minimal value.\n");
294         return;
295     }
296     printf("Minimal Value is %2.2f , position is ", pivot);
297     print_int(tmp->n, tmp);
298 }
299

```



```
300 // Get rid of the useless blank characters.
301 char *clean(char *string) {
302     char *head = string;
303     int count_space = 0;
304     while (*string == ' ' && *string != '\0') {
305         count_space += 1;
306         string += 1;
307     }
308     string = head;
309
310     int len = 1;
311     while (*string != '\0') {
312         len += 1;
313         string++;
314     }
315     string = head;
316
317     char *ans = (char *)calloc(len - count_space, sizeof(char));
318     if (ans == NULL) {
319         printf("Can't get memory!\n");
320         return NULL;
321     }
322     char *ans_head = ans;
323
324     while (*string != '\0') {
325         if (*string != ' ') {
326             *ans = *string;
327             ans++;
328         }
329         string++;
330     }
331     *ans = *string;
332     ans = ans_head;
333     string = head;
334
335     ans = ans + 1;
336     char *tmp;
337     for (tmp = ans; *tmp != '\0'; tmp++) {
338         if (*(tmp + 1) == '\0') {
339             *tmp = ',';
340         }
341     }
342     return ans;
343 }
344
345 char *cut(char *string) {
346     while (*string != ',') {
347         if (*string == '\0') {
348             return '\0';
```

```

349     }
350     string++;
351 }
352 return ++string;
353 }
354
355 // Put an new element into the stack
356 double get_Number(char *string) {
357     if (*string == '\0') {
358         return NULL;
359     }
360     double ans = 0.;
361     if (*string == '\0') {
362         return NULL;
363     }
364     if (*string != '-') {
365         char_LinkedList *work = (char_LinkedList *)malloc(sizeof(char_LinkedList));
366         if (work == NULL) {
367             printf("Can't get memory!\n");
368             return 0;
369         }
370         // container
371
372         char_LinkedList *head = work;
373         int i = 1;
374         while (*string != ',') {
375             work->elements = *string;
376             work->times = i;
377             work->next = (char_LinkedList *)malloc(sizeof(char_LinkedList)); // malloc
378             if (work->next == NULL) {
379                 printf("Can't get memory!\n");
380                 return 0.;
381             }
382             work = work->next; // move
383             work->elements = NULL;
384             work->times = NULL;
385             string++;
386             i++;
387         }
388
389         work->elements = *string;
390         work->times = NULL;
391
392         string++;
393
394         work = head;
395         int dot = 1;
396         int comma = 1;
397         int dot_index = NULL;

```

```
398     while (work->elements != ',') {
399         if (work->elements == '.') {
400             dot_index = dot;
401             break;
402         }
403         work = work->next;
404         dot++;
405     }
406
407     if (dot_index == NULL) {
408         dot_index = i;
409     }
410
411     work = head;
412
413     while (work->times != NULL) {
414         work->times = -1 * (work->times - dot_index);
415         work = work->next;
416     }
417
418     work = head;
419
420     while (work->elements != ',') {
421         if (work->elements == '.') {
422             work = work->next;
423             continue;
424         }
425         if (work->times > 0) {
426             ans += pow(10, work->times - 1) * double(int(work->elements) - int('0'));
427             work = work->next;
428         }
429         else {
430             ans += pow(10, work->times) * double(int(work->elements) - int('0'));
431             work = work->next;
432         }
433     }
434 }
435 else {
436     string = string + 1;
437     ans = -1 * get_Number(string);
438 }
439 return ans;
440 }
441
442 int main(int argc, char *argv[]) {
443     if (argc != 3) {
444         printf("This function needs and only needs 2 arguments.\n");
445         return 0;
446     }
```

```

447
448     char *string_1 = *(argv + 1);
449     char *string_2 = *(argv + 2);
450
451     //char string_1_tmp[] = "( -3.14,20 ,-256, 0 ,6,5,12121,4588, 89)";
452     //char *string_1 = string_1_tmp;
453
454     //char string_2_tmp[] = "(3.14, -1, 256,3.2222,2,0,5633.2,168,78)";
455     //char *string_2 = string_2_tmp;
456
457     string_1 = clean(string_1);
458     string_2 = clean(string_2);
459
460     Dynamic_Array c_1, c_2;
461     c_1.A = (double *)malloc(sizeof(double));
462     if (c_1.A == NULL) {
463         printf("Can't get memory!\n");
464         return 0;
465     }
466     c_1.capacity = 1;
467     c_1.n = 0;
468
469     c_2.A = (double *)malloc(sizeof(double));
470     if (c_2.A == NULL) {
471         printf("Can't get memory!\n");
472         return 0;
473     }
474     c_2.capacity = 1;
475     c_2.n = 0;
476
477     while (string_1 != '\0') {
478         Append(&c_1, get_Number(string_1));
479         string_1 = cut(string_1);
480     }
481
482     while (string_2 != '\0') {
483         Append(&c_2, get_Number(string_2));
484         string_2 = cut(string_2);
485     }
486     c_1.n -= 1;
487     c_2.n -= 1;
488
489     Div_Dynamic_Array ans;
490     ans.A = (Div *)malloc(sizeof(Div));
491     if (ans.A == NULL) {
492         printf("Can't get memory!\n");
493         return 0;
494     }
495     ans.capacity = 1;

```

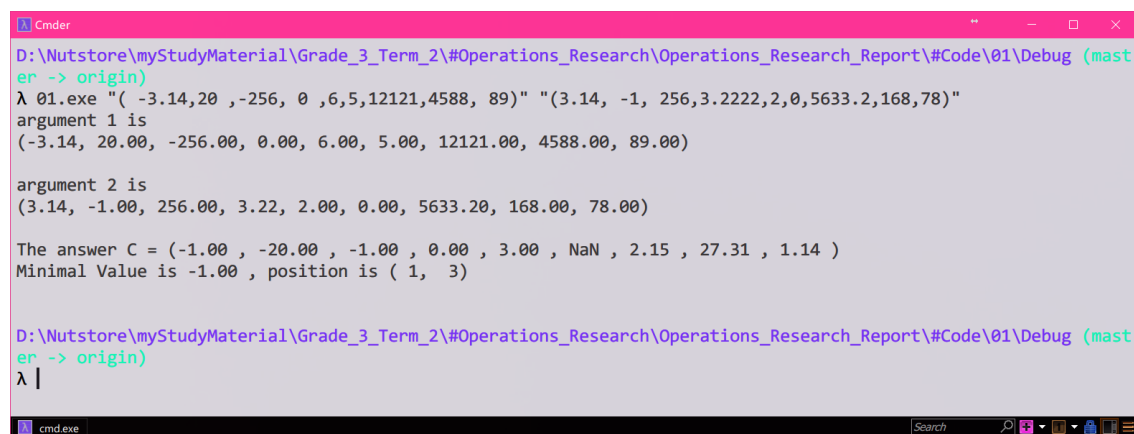
```

496     ans.n = 0;
497
498     printf("argument 1 is\n");
499     print(1, &c_1);
500     printf("argument 2 is\n");
501     print(2, &c_2);
502     Div_onArray(&c_1, &c_2, &ans);
503     Div_print(&ans);
504     find(&ans);
505
506     //system("pause");
507     return 0;
508 }

```

程序代码 1

六、 运行结果



```

D:\Nutstore\myStudyMaterial\Grade_3_Term_2\#Operations_Research\Operations_Research_Report\#Code\01\Debug (master -> origin)
λ 01.exe "( -3.14,20 , -256, 0 ,6,5,12121,4588, 89)" "(3.14, -1, 256,3.2222,2,0,5633.2,168,78)"
argument 1 is
(-3.14, 20.00, -256.00, 0.00, 6.00, 5.00, 12121.00, 4588.00, 89.00)

argument 2 is
(3.14, -1.00, 256.00, 3.22, 2.00, 0.00, 5633.20, 168.00, 78.00)

The answer C = (-1.00 , -20.00 , -1.00 , 0.00 , 3.00 , NaN , 2.15 , 27.31 , 1.14 )
Minimal Value is -1.00 , position is ( 1, 3)

D:\Nutstore\myStudyMaterial\Grade_3_Term_2\#Operations_Research\Operations_Research_Report\#Code\01\Debug (master -> origin)
λ |

```

运行结果 1（经过了反相处理）

1.3 代码分析

优势在于可以 shell 调用，不再需要修改源代码；其次，数组是动态的，所以可以大容量输入。

七、 实验体会

Shell 的解释程序是最难的，这里用了一个原创的方式，来解释输入的字符串。

指针的操作比较复杂，需要时刻牢记 malloc 与 free 的对应^[1]，并且要对堆中申请到的地址进行排查，看是否申请成功。在进行调试的时候，时常遇到内存的读取冲突问题，查找了微软的官方 Visual C++ 编译器的手册，方才明白这里的局部变量必须要初始化才可以使用，这与 GNU 的 MinGW 编译器稍有区别。

八、 参考文献

- [1] 林锐. 高质量 C++/C 编程指南 [M]. 1.0 ed., 2001.