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：** 2018-03-21
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$\boldsymbol{a}=(a_1,a_2,\cdots,a_n)$和$\boldsymbol{b}=(b_1,b_2,\cdots,b_n)$，求

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

$$c_i=\begin{cases}\dfrac{a_i}{b_i}, & b_i\neq 0\\ \mathrm{NaN}, & b_i=0\end{cases},\qquad i=1,2,\cdots,n.$$

2. 求最小值及所有最小值的下标，其中最小值为

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

三、实验平台

Windows 10 Pro 1703；

Microsoft© Visual Studio 2017 Enterprise。

四、算法设计

Algorithm: find the minimal value and all the indexes

Input: two list **a** and **b** and their length are n.

Output: list c, whose value is the division of a by b at the same position; minimal value and their positions.

Begin

Step 1:

```
for i = 0 through n
        if b[i] != 0
                c[i] = a[i] / b[i]
        else
                c[i] = NaN
```

output **c**

Step 2: **tmp_c** is a set contains all the elements in **c** whose b[i] > 0;
sort **tmp_c** incrementally, set tmp = **tmp_c**[0]

Step 3: for i = 0 through n
if c[i] == a[i] and b[i] > 0
output i

End

五、程序代码

5.1 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。

5.2 程序代码

```
// filename: Source.c

/* -*- coding: utf-8 -*-

Created on Wed Mar 14 19 : 10 : 28 2018

@author: LiuPeng

@version: 1.0

last edit: 208-03-24 17:36

*/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>

// The following type is a container for creating a stack.
typedef struct char_LinkedList {
    char_LinkedList *head;
    char elements;          // partition must be integer less than 10
    int times;              // 这是一个容器，放置一个数组，用指针作为头
    char_LinkedList *next;
}char_LinkedList;

typedef struct Dynamic_Array {
    double *A;              // 底层数组
    int capacity;           // 底层数组的容量
    int n;                 // 底层数组的占用量
}Dynamic_Array;

typedef struct Div {
    double up;
    double down;
    double value;
    char state[10];         // NaN or Negative，长度不定
                           // 这个 state 必须是 malloc 而来的，坚决不能直接用
}Div;

```

```
42  typedef struct Div_Dynamic_Array {
43      Div *A;            // 底层结构体数组的头指针，不能动!
44      int capacity;       // 底层结构体数组的容量
45      int n;             // 底层数组的占用量
46  }Div_Dynamic_Array;
47
48  void Div_Resize(Div_Dynamic_Array *D) {
49      int i = 0;
50      Div *tmp = (Div *)calloc(2 * D->capacity, sizeof(Div));
51      if (tmp == NULL) {
52          printf("Cannot get memory, crash!\n");
53          return;
54      }
55      for (i = 0; i < D->capacity; i++) {
56          (tmp + i)->up = (D->A + i)->up;
57          (tmp + i)->down = (D->A + i)->down;
58          (tmp + i)->value = (D->A + i)->value;
59          strcpy((tmp + i)->state, (D->A + i)->state);           //不能简单复制，否则会内存出错
60      }
61      free(D->A);
62      D->A = tmp;
63      tmp = NULL;         // 避免野指针
64
65      D->capacity *= 2;
66  }
67
68  void Div_Append(Div_Dynamic_Array *D, Div e) {
69      if (D->n == D->capacity) {
70          Div_Resize(D);
71      }
72      (D->A + D->n)->up = e.up;
73      (D->A + D->n)->down = e.down;
74      (D->A + D->n)->value = e.value;
75      strcpy((D->A + D->n)->state, e.state);
76      D->n += 1;
77      //int i;
78      //for (i = 0; i <= D->n; i++) {
79      //  printf("%s\t", (D->A + i)->state);
80      //}
81      //printf("\n");
82  }
83
84  void Div_print(Div_Dynamic_Array *d) {
85      int i;
86      printf("The answer C = (");
87      for (i = 0; i < d->n; i++) {
88          if (!strcmp((d->A + i)->state, "NaN")) {
89              printf("%s ", "NaN");
90          }
```

```
91          else {
92              double value = (d->A + i)->value;
93              printf("%2.2f ", value);
94          }
95          if (i == d->n - 1) {
96              printf("");
97          }
98          else {
99              printf(", ");
100         }
101     }
102     printf(")\n");
103 }
104
105 void Div_onArray(Dynamic_Array *a, Dynamic_Array *b, Div_Dynamic_Array *ans) {
106     if (a->n != b->n) {
107         printf("length should be the same.");
108         return;
109     }
110
111     int i;
112     for (i = 0; i < a->n; i++) {
113         if (*(b->A + i) == 0) {
114             Div tmp;
115             tmp.up = NULL;
116             tmp.down = NULL;
117             tmp.value = NULL;
118             char c[] = "NaN";
119             strcpy(tmp.state, c);
120             Div_Append(ans, tmp);
121         }
122         else {
123             if (*(b->A + i) < 0.) {
124                 Div tmp;
125                 tmp.up = *(a->A + i);
126                 tmp.down = *(b->A + i);
127                 tmp.value = tmp.up / tmp.down;
128                 char c[] = "Negative";
129                 strcpy(tmp.state, c);
130                 Div_Append(ans, tmp);
131             }
132             else {
133                 Div tmp;
134                 tmp.up = *(a->A + i);
135                 tmp.down = *(b->A + i);
136                 tmp.value = tmp.up / tmp.down;
137                 char c[] = "Normal";
138                 strcpy(tmp.state, c);
139                 Div_Append(ans, tmp);
```

```
140             }
141         }
142     }
143 }
144
145 void print(int n, Dynamic_Array *d) {   // 输出一个动态的双精度数组
146     printf(/* "argument %d is \n*/"(");
147     int i;
148     for (i = 0; i < d->n - 1; i++) {
149         printf("%2.2f, ", *(d->A + i));
150     }
151     printf("%2.2f", *(d->A + i));
152     printf(")\n\n");
153 }
154
155 void print_int(int n, Dynamic_Array *d) {   // 输出一个动态的双精度数组
156     printf(/* "argument %d is \n*/"(");
157     int i;
158     for (i = 0; i < d->n - 1; i++) {
159         printf("%2.0f, ", *(d->A + i));
160     }
161     printf("%2.0f", *(d->A + i));
162     printf(")\n\n");
163 }
164
165 void Resize(Dynamic_Array *D) {
166     int i = 0;
167     double *tmp = (double *)calloc(2 * D->capacity, sizeof(double));
168     if (tmp == NULL) {
169         printf("Cannot get memory, crash!\n");
170         return;
171     }
172     for (i = 0; i < D->capacity; i++) {
173         *(tmp + i) = *(D->A + i);
174     }
175     D->A = tmp;
176     D->capacity *= 2;
177 }
178
179 void Append(Dynamic_Array *D, double e) {
180     if (D->n == D->capacity) {
181         Resize(D);
182     }
183     *(D->A + D->n) = e;
184     D->n += 1;
185 }
186
187 Dynamic_Array *Quick_sort(Dynamic_Array *a) {
188
```

```
189     Dynamic_Array *less = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
190     less->A = (double *)calloc(1, sizeof(double));
191     if (!less) {
192         printf("Can't get memory!");
193         return NULL;
194     }
195     less->capacity = 1;
196     less->n = 0;
197 
198     Dynamic_Array *more = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
199     more->A = (double *)calloc(1, sizeof(double));
200     if (!more) {
201         printf("Can't get memory!");
202         return NULL;
203     }
204     more->capacity = 1;
205     more->n = 0;
206 
207     Dynamic_Array *eq = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
208     eq->A = (double *)calloc(1, sizeof(double));
209     if (!eq) {
210         printf("Can't get memory!");
211         return NULL;
212     }
213     eq->capacity = 1;
214     eq->n = 0;
215 
216     int i;
217     if (a->n <= 1) {
218         return a;
219     }
220     else {
221         /*double pivot = 1 / 3. * (*(a->A) + ;*/
222 
223         for (i = 0; i < a->n; i++) {
224             double pivot = *(a->A);
225             if (*(a->A + i) > pivot) {
226                 Append(more, *(a->A + i));
227             }
228             else {
229                 if (*(a->A + i) < pivot) {
230                     Append(less, *(a->A + i));
231                 }
232                 else {
233                     Append(eq, *(a->A + i));
234                 }
235             }
236         }
237     }
```

```
238      less = Quick_sort(less);
239      more = Quick_sort(more);
240      for (i = 0; i < eq->n; i++) {
241          Append(less, *(eq->A + i));
242      }
243      for (i = 0; i < more->n; i++) {
244          Append(less, *(more->A + i));
245      }
246      return less;
247  }
248
249  void find(Div_Dynamic_Array *a) {
250
251      Dynamic_Array *c = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
252      Dynamic_Array *d = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
253      c->A = (double *)calloc(a->n, sizeof(double));
254      if (c == NULL || d == NULL || c->A == NULL) {
255          printf("Can't get memory!\n");
256          return;
257      }
258      c->capacity = a->n;
259      c->n = 0;
260
261      int i = 0;
262      for (i = 0; i < a->n; i++) {
263          if (!strcmp((a->A + i)->state, "Normal")) {     // 分母合法的就 append
264              Append(c, (a->A + i)->value);
265          }
266      }
267      d = Quick_sort(c);      // 排序一下
268                             //print(d->n, d);
269
270      double pivot = *(d->A + 0);
271      Dynamic_Array *tmp = (Dynamic_Array *)calloc(1, sizeof(Dynamic_Array));
272      tmp->A = (double *)calloc(1, sizeof(double));
273      if (tmp == NULL || tmp->A == NULL) {
274          printf("Can't get memory!\n");
275          return;
276      }
277      tmp->capacity = 1;
278      tmp->n = 0;
279      for (i = 0; i < a->n; i++) {
280          if (!strcmp((a->A + i)->state, "Normal") && (a->A + i)->value == pivot) {
281              Append(tmp, ++i);
282          }
283      }
284      if (tmp->n == 0) {
285          printf("Sorry, no minimal value.\n");
286          return;
```

```
287     }
288     printf("Minimal Value is %2.2f , position is ", pivot);
289     print_int(tmp->n, tmp);
290 }
291 
292 char *clean(char *string) {     // 已经后期优化，减去了字符串中所有的空格
293     char *head = string;
294     int count_space = 0;
295     while (*string == ' ' && *string != '\0') {
296         count_space += 1;
297         string += 1;
298     }
299     string = head;
300 
301     int len = 1;    // 有'\0'，所以要＋1
302     while (*string != '\0') {
303         len += 1;
304         string++;
305     }
306     string = head;
307 
308     char *ans = (char *)calloc(len - count_space, sizeof(char));
309     if (ans == NULL) {
310         printf("Can't get memory!\n");
311         return NULL;
312     }
313     char *ans_head = ans;
314 
315     while (*string != '\0') {
316         if (*string != ' ') {
317             *ans = *string;
318             ans++;
319         }
320         string++;
321     }
322     *ans = *string;
323     ans = ans_head;
324     string = head;
325 
326     ans = ans + 1;
327     char *tmp;
328     for (tmp = ans; *tmp != '\0'; tmp++) {
329         if (*(tmp + 1) == '\0') {
330             *tmp = ',';
331         }
332     }
333     return ans;
334 }
335 
```

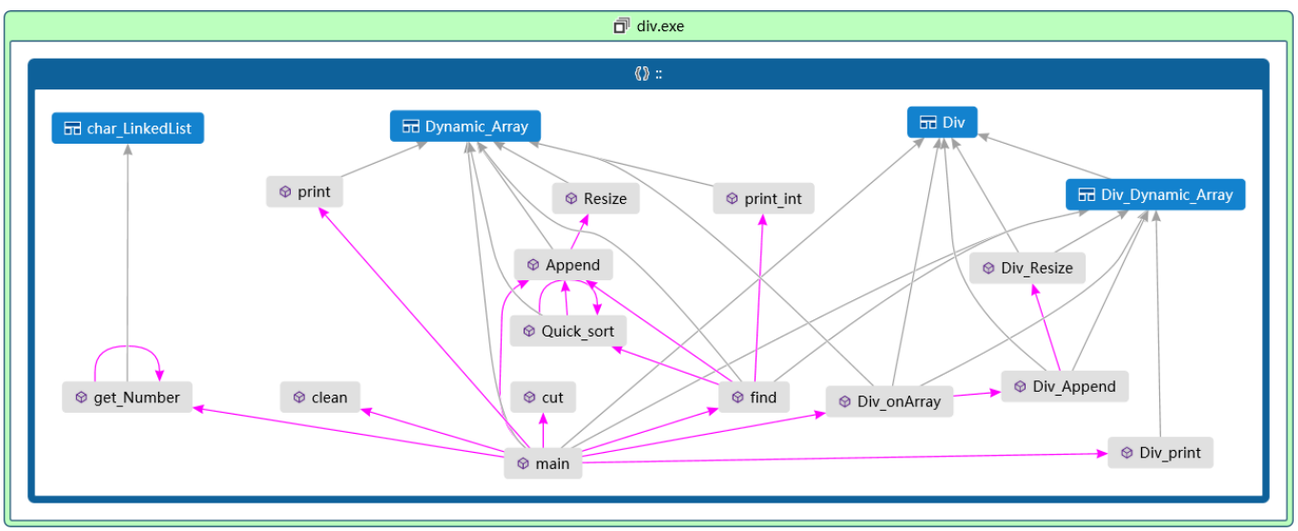
```
336 char *cut(char *string) {
337     while (*string != ',') {
338         if (*string == '\0') {
339             return '\0';
340         }
341         string++;
342     }
343     return ++string;
344 }
345 
346 // Put an new element into the stack
347 double get_Number(char *string) {
348     // 传递一个完整的 clean 过的字符串进来，按需切割头部，剩下的头作为新的头。
349     if (*string == '\0') {
350         return NULL;
351     }
352     double ans = 0.;
353     if (*string == '\0') {
354         return NULL;
355     }
356     if (*string != '-') {
357         char_LinkedList *work = (char_LinkedList *)malloc(sizeof(char_LinkedList));
358         if (work == NULL) {
359             printf("Can't get memory!\n");
360             return 0;
361         }
362         // container
363 
364         char_LinkedList *head = work;
365         int i = 1;
366         while (*string != ',') {
367             work->elements = *string;
368             work->times = i;
369             work->next = (char_LinkedList *)malloc(sizeof(char_LinkedList));    // 申请
370             if (work->next == NULL) {
371                 printf("Can't get memory!\n");
372                 return 0.;
373             }
374             work = work->next;                                                  // 移动
375             work->elements = NULL;
376             work->times = NULL;
377             string++;
378             i++;    // i 在后面还有用
379         }
380 
381         work->elements = *string;       // 逗号也要加上
382         work->times = NULL;            // 逗号的指数不能为有意义的
383 
384         string++;
```

```
385
386         work = head;
387         int dot = 1;
388         int comma = 1;      // 逗号的用处
389         int dot_index = NULL;
390         while (work->elements != ',') {
391             if (work->elements == '.') {
392                 dot_index = dot;
393                 break;
394             }
395             work = work->next;
396             dot++;
397         }
398
399         if (dot_index == NULL) {
400             dot_index = i;
401         }
402
403         work = head;
404
405         while (work->times != NULL) {
406             work->times = -1 * (work->times - dot_index);
407             work = work->next;
408         }
409
410         work = head;
411
412         while (work->elements != ',') {
413             if (work->elements == '.') {
414                 work = work->next;
415                 continue;
416             }
417             if (work->times > 0) {
418                 ans += pow(10, work->times - 1) * double(int(work->elements) - int('0'));
419                 work = work->next;
420             }
421             else {
422                 ans += pow(10, work->times) * double(int(work->elements) - int('0'));
423                 work = work->next;
424             }
425         }
426     }
427     else {
428         string = string + 1;
429         ans = -1 * get_Number(string);
430     }
431     return ans;
432 }
433
```

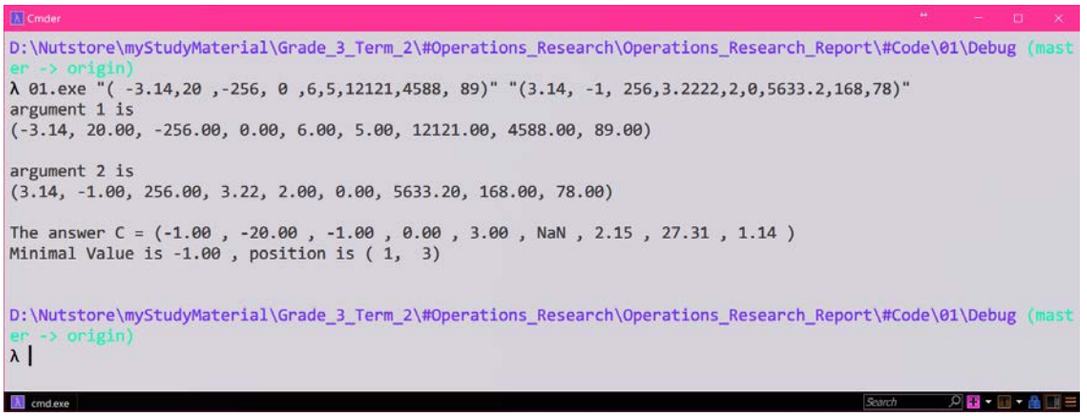
```
434  int main(int argc, char *argv[]) {
435      if (argc != 3) {
436          printf("This function needs and only needs 2 arguments.\n");
437          return 0;
438      }
439
440      char *string_1 = *(argv + 1);
441      char *string_2 = *(argv + 2);
442
443      //char string_1_tmp[] = "( -3.14,20 ,-256, 0 ,6,5,12121,4588, 89)";
444      //char *string_1 = string_1_tmp;
445
446      //char string_2_tmp[] = "(3.14, -1, 256,3.2222,2,0,5633.2,168,78)";
447      //char *string_2 = string_2_tmp;
448
449      string_1 = clean(string_1);
450      string_2 = clean(string_2);
451
452      Dynamic_Array c_1, c_2;
453      c_1.A = (double *)malloc(sizeof(double));
454      if (c_1.A == NULL) {
455          printf("Can't get memory!\n");
456          return 0;
457      }
458      c_1.capacity = 1;
459      c_1.n = 0;
460
461      c_2.A = (double *)malloc(sizeof(double));
462      if (c_2.A == NULL) {
463          printf("Can't get memory!\n");
464          return 0;
465      }
466      c_2.capacity = 1;
467      c_2.n = 0;
468
469      while (string_1 != '\0') {
470          Append(&c_1, get_Number(string_1));
471          string_1 = cut(string_1);
472      }
473
474      while (string_2 != '\0') {
475          Append(&c_2, get_Number(string_2));
476          string_2 = cut(string_2);
477      }
478      c_1.n -= 1;     // 这也是无奈之举啊，谁让 0.0 ==NULL 呢
479      c_2.n -= 1;
480
481      Div_Dynamic_Array ans;
482      ans.A = (Div *)malloc(sizeof(Div));
```

```
483     if (ans.A == NULL) {
484         printf("Can't get memory!\n");
485         return 0;
486     }
487     ans.capacity = 1;
488     ans.n = 0;
489 
490     printf("argument 1 is\n");
491     print(1, &c_1);
492     printf("argument 2 is\n");
493     print(2, &c_2);
494     Div_onArray(&c_1, &c_2, &ans);
495     Div_print(&ans);
496     find(&ans);
497 
498     //system("pause");
499     return 0;
500 }
```

程序代码 1



六、运行结果



运行结果 1 （经过了反相处理）

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。

劣势在于没有采用并行计算，在进行大规模计算的时候，只能调用一个 CPU 核心，效率较低。

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.