递 归 专 题 实 践 报 告

——迷宫探路与非递归快排

Jammy Zeta

目录

Ⅰ.迷宫探路

Ⅱ.非递归快排

Ⅰ.迷宫探路

一. 问题描述

在一个迷宫地图中给定起点与终点，程序需要判断迷宫是否有解，若有解，则找出其中的一条路径。单一路径的结果往往会包含蛇形的路线，因此需要考虑最短路径的算法。

二. 问题分析

1. 人为构造迷宫在迷宫较大时会带来许多麻烦，故考虑利用 rand()函数来构造迷宫，另外该迷宫可以人为修改
2. 读入迷宫时需将迷宫转换为便于后续操作的形式，故可以用特定数字代表通路和障碍
3. 仅寻求一个可行解的思路是简单的，只要从起点开始往前走，走不通，则调换方向，若是另外三个方向（排除来时的方向）均走不通（均是障碍）则往回走一步再调换方向试探，前进的过程中同时记录路径，直至抵达终点，如若最终回到了起点，说明所有路径均无法抵达终点，可以断定迷宫无解。该方法亦称回溯法
4. 寻求最短路径时，问题就有一些棘手了。一开始的想法是自然也是最直接的：在 3 的基础上求出所有可行解然后逐一比较得出最短路径。但这个方法显然存在很多缺点，比如耗时长，会占用较大的存储空间等。接下来考虑如何简化算法。其实仔细分析问题后我们可以发现一条可行路径上的每一个位置都可能会存在多种抵达终点的路径，那我们是否可以在每一个位置上标记出该位置到终点的最少步数（最短路径的移动次数），这样下次再抵达该位置时就不必继续试探下去，而是沿着逐一递减的最少步数到达终点，事实证明这个思路是可行的
5. 但我们还需要解决一个问题，那就是如何去标记每个位置的最短路径。最初的想法是直接从起点开始在递归的过程中标记每个位置的最少步数，此时会发现即使处于终点的邻位（终点四个方位的通路）也很难让程序直接得出最短步数是 1，因为程序必须走遍所有路径才能得出最短步数。事实上我们可以换一个角度去思考，我们寻求的是到终点的最短路径，那为何不直接从终点向外发散去标记呢？因为终点前进一步所抵达的位置可以说明该位置到终点的最短步数必然是 1。那这就可以类比成一个病毒传染模型，终点是一个传染源，每一次被感染（被标记）的格子都会去感染（标记）四周还未感染（标记）的格子。一旦起点被标记即可结束标记，然后沿着逐一递减的标记数字即可找到最短路径。

三． 算法说明

1. 输入迷宫的长和宽以及迷宫中障碍所占比例可以在相应的文本中随机创建一个迷宫，并包含重置、修改的功能
2. 创建一个长与宽均比迷宫大 2 的二维数组，为防止数组越界将迷宫的四周均设为障碍，然后读入迷宫，按通路为 0、障碍为-2(求单一路径中)或-1(求最短路径中)的规则存入数组

//单一路径：

3. 读入起点与终点坐标，在数组中将终点标记为-1
4. 由于链表的结构特性用其来存储路径，前进即添加节点，回溯即删除节点
5. 路径搜索函数（PathSearch）如果遇到起点便返回-1，若遇到终点则返回 1
6. 为防止出现死循环，每次所达到的位置均标记为-3（之所以不标记为-2 是因为最后输出迷宫的时候需要将走过的通路和起初就存在的障碍区分开来），可以保证每个位置只走一次

//最短路径：

7. 读入起点与终点坐标，在数组中将终点标记为 1
8. 第一次将终点四周的通路标记为 2，第二次再将所有标记为 2 的位置四周的通路标记为 3，以此类推，直至标记到起点或迷宫通路已全部被标记
9. 为保证标记数字代表的是该位置到达终点的最短步数，不标记已标记的位置
10. 若是直接采取普通的递归不会产生“病毒感染扩散”的效果，因此这里采取用链表将这些标记同样数字位置四周所有的通路串联起来进行递归，即先是终点四周通路标记后被串联起来，再将这些位置四周的通路标记完后串联起来，以此类推，直到该链表为空或者起点被标记为止
11. 最后起点若还是 0 则表示终点无法抵达起点，若起点不为 0 则按照降序的顺序找到最短路径

单一路径源代码

最短路径源代码

四. 部分样例展示

One Path——单一路径
One of the shortest Paths——最短路径

Sample 1（10×10）

```
One Path:
 #    #    #    #    #    #    #    #    #    #    #    #
 #  [001]  #    -    #    #    #    -    #    -    -    #
 #  [002]  -    #    -    -    -    -    -    -    -    #
 #  [003]  -    #    -    -    #    -    -    -    -    #
 #  [004][005]  -    -    -    -    -    -    -    -    #
 #    #  [006][007]  #    #    -    #    -    -    -    #
 #    -    #  [008]  -    -    #    -    -    -    -    #
 #  [013][012][009]  #    -    -    -    -    -    #    #
 #  [014][011][010]  -    #    -    #    -    -    -    #
 #  [015]  #    #    #    -    -    -    #    -    -    #
 #  [016][017][018][019][020][021][022][023][024][025]  #
 #    #    #    #    #    #    #    #    #    #    #    #
Elapsed time:0.000000 s
```

P.s.上图中[009]-[014]的过程中出现蛇形走位

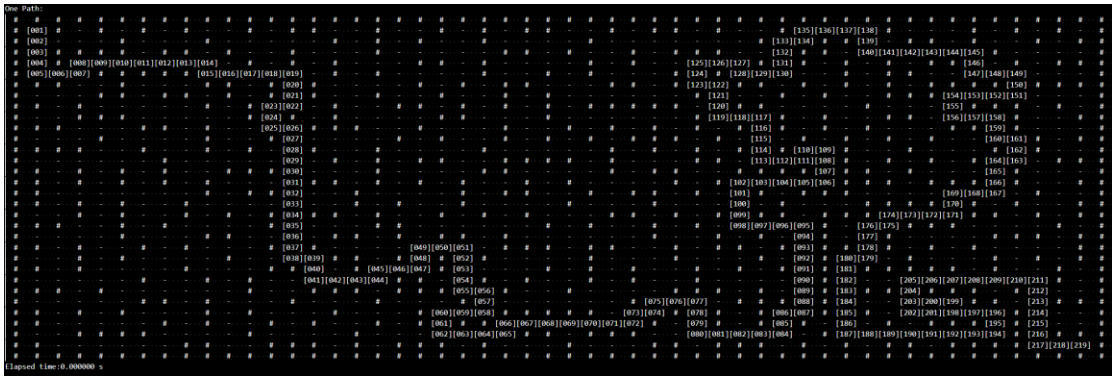
```
One of the shortest Paths:
 #    #    #    #    #    #    #    #    #    #    #    #
 #  [001]  #    -    #    #    #    -    #    -    -    #
 #  [002]  -    #    -    -    -    -    -    -    -    #
 #  [003]  -    #    -    -    #    -    -    -    -    #
 #  [004][005]  -    -    -    -    -    -    -    -    #
 #    #  [006][007]  #    #    -    #    -    -    -    #
 #    -    #  [008][009][010]  #    -    -    -    -    #
 #    -    -    -    #  [011][012]  -    -    -    #    #
 #    -    -    -    -    #  [013]  #    -    -    -    #
 #    -    #    #    #    -  [014]  -    #    -    -    #
 #    -    -    -    -    -  [015][016][017][018][019]  #
 #    #    #    #    #    #    #    #    #    #    #    #
Elapsed time:0.000000 s
```

Sample 3（50×50）

Elapsed time:0 s

Elapsed time:0 s

Sample 2（30×50）

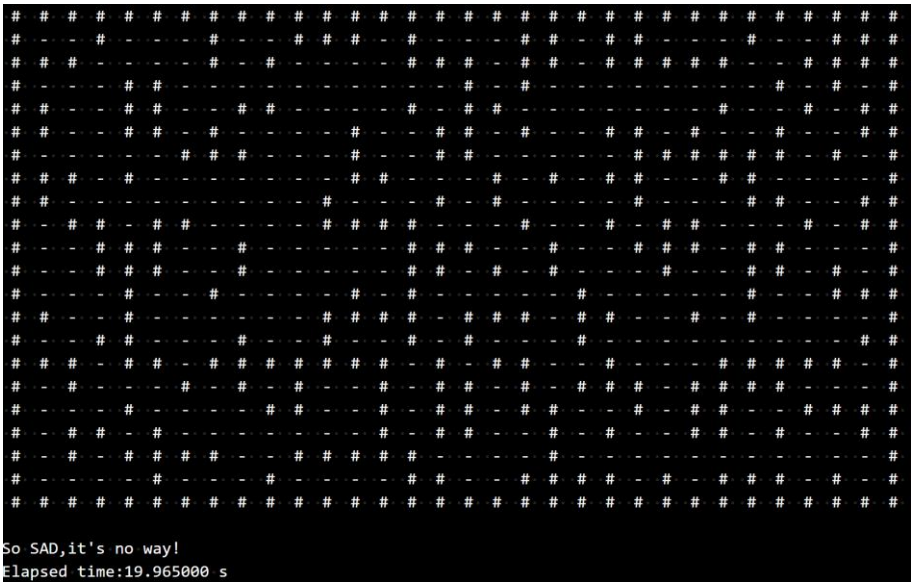


Elapsed time:0 s

Elapsed time:0.001 s

Sample 4（NULL）

——寻求最短路径的无解极端情况



Ⅱ.非递归快排

一. 问题描述

为了更好地了解递归程序内部的运行方式,我们可以尝试用非递归算法(栈)改写递归算法。

二. 问题分析

1. 非递归算法与递归算法的本质事实上是类似的,无非递归是程序自动分配栈,而非递归则是人为分配栈去存储相应的参数
2. 用栈实现递归函数需要具备初始化栈(`InitStack`)、入栈(`PushStack`)和出栈(`PopStack`)等基本操作
3. 栈的空间是人为分配的,因此需要考虑空间不足的情况,即需要具备扩充栈空间(`InflateStack`)的操作
4. 将每次递归所需的参数逐一压入栈,然后通过循环逐一出栈即可

三. 算法说明

1. 栈的结构由两个整型分别存储栈的大小(size)、栈顶元素的序号(top)和一个用于存储函数参数的整型指针 data 组成
2. 栈的初始化空间为 100 B,后续单次增加空间则为 50 B
3. 每次入栈时先比较 size 和 top 的大小来确定栈的空间是否已满,若已满则扩充栈空间
4. 扩充栈空间的函数是利用`void realloc(void *Memory,int Size)`函数来完成的
5. 递归型快排函数本是在函数的末尾开始下一轮的递归,因此非递归快排函数在函数的末尾将本应参加递归的函数参数压入栈中,而在函数的开头通过出栈读入栈顶的参数,外加循环即可完成非递归快速排序

源代码

四. 部分样例展示

Sample 1

```
Enter the size of array:
100
Enter 0 to make the Array invisible,or enter any other numbers.
1
the initial Array:
11304 29686 19195 24659 8198 13546 16563 10924 16048 24330 12865 32636 22552 3722 26325 30141 8607 29785 6265 14562 1951
3 9134 23378 25359 17868 24036 2594 18313 10714 20615 23570 16748 9568 31151 1967 23196 20990 9868 5938 24317 19543 1645
1 20996 12263 2491 23288 24449 27604 12548 15148 1789 15964 31421 17738 12548 27748 23690 2940 3558 28818 963 28099 1395
 15364 25443 71 21508 28481 22817 11244 30907 8047 11740 15653 22795 31687 30012 21773 15738 26439 4247 28852 2981 23206
 22801 31414 24965 8845 13576 6931 30194 19972 7343 14650 25189 24875 20984 25703 14850 8234
the random Array in order:
71 963 1395 1789 1967 2491 2594 2940 2981 3558 3722 4247 5938 6265 6931 7343 8047 8198 8234 8607 8845 9134 9568 9868 107
14 10924 11244 11304 11740 12263 12548 12548 12865 13546 13576 14562 14650 14850 15148 15364 15653 15738 15964 16048 164
51 16563 16748 17738 17868 18313 19195 19513 19543 19972 20615 20984 20990 20996 21508 21773 22552 22795 22801 22817 231
96 23206 23288 23378 23570 23690 24036 24317 24330 24449 24659 24875 24965 25189 25359 25443 25703 26325 26439 27604 277
48 28099 28481 28818 28852 29686 29785 30012 30141 30194 30907 31151 31414 31421 31687 32636
Elapsed time:0.000000 s
```

Sample 2

```
Enter the size of array:
10000
Enter 0 to make the Array invisible,or enter any other numbers.
0
Elapsed time:0.002000 s
```

Sample 3

```
Enter the size of array:
100000
Enter 0 to make the Array invisible,or enter any other numbers.
0
Elapsed time:0.019000 s
```

Sample 4

```
Enter the size of array:
1000000
Enter 0 to make the Array invisible,or enter any other numbers.
0
Elapsed time:0.228000 s
```

结语：非递归快排效率和递归型快排还是有一定差距的，不过非递归大部分时间应该都用于栈的处理上了。

附录 I

```
//寻求单一路径:
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
typedef struct PathTrack
{
    struct PathTrack *pre;
    int x, y;
    struct PathTrack *next;
} ptk;
void createMaze(int length, int width);
void basicSetting(int length, int width);
int PathSearch(ptk *start, int **maze);
ptk *addPath(ptk *start, int sgn);
ptk *withdrawPath(ptk *start);
int main()
{
    int length, width, flag;
    printf("Enter the length and width of maze:\n");
    scanf("%d%d", &length, &width);
    basicSetting(length, width);
    FILE *fp = fopen("maze1.txt", "a+");
    int **maze = (int **)malloc((length + 2) * sizeof(int *));
    for (int i = 0; i < length + 2; ++i)
        maze[i] = (int *)malloc((width + 2) * sizeof(int));
    for (int i = 0; i < length + 2; ++i)
    {
        for (int j = 0; j < width + 2; ++j)
        {
            char tmp;
            fscanf(fp, " %c ", &tmp);
            if (tmp == '#')
```

```
                maze[i][j] = -2;
            else if (tmp == '-')
                maze[i][j] = 0;
        }
    }
    printf("Now enter the coordinates of start:\n");
    int startx, starty;
    scanf("%d%d", &startx, &starty);
    maze[startx][starty] = 0;
    printf("Now enter the coordinates of destination:\n");
    int endx, endy;
    scanf("%d%d", &endx, &endy);
    maze[endx][endy] = -1;
    ptk *start = (ptk *)malloc(sizeof(ptk));
    start->x = startx;
    start->y = starty;
    start->pre = NULL;
    start->next = NULL;
    clock_t beginning = clock();
    flag = PathSearch(start, maze);
    clock_t ending = clock();
    if (flag == -1)
        fprintf(fp, "\nSo SAD,it's no way!\n");
    else
    {
        int cnt = 1;
        ptk *tmp = start;
        while (tmp)
        {
            maze[tmp->x][tmp->y] = cnt++;
            tmp = tmp->next;
        }
        fprintf(fp, "\nOne Path:\n");
        for (int i = 0; i < length + 2; ++i)
```

```
        {
            for (int j = 0; j < width + 2; ++j)
                if (maze[i][j] == -2)
                    fprintf(fp, "  #  ");
                else if (maze[i][j] == -3 || maze[i][j] == 0)
                    fprintf(fp, "  -  ");
                else
                    fprintf(fp, "[%03d]", maze[i][j]);
            fprintf(fp, "\n");
        }
    }
    fprintf(fp, "Elapsed time:%f s", (double)(ending - beginni
ng) / CLOCKS_PER_SEC);
    while (start)
    {
        ptk *cup = start->next;
        free(start);
        start = cup;
    }
    for (int i = 0; i < length + 2; ++i)
        free(maze[i]);
    free(maze);
    fclose(fp);
}
void createMaze(int length, int width)
{
    FILE *fp = fopen("maze1.txt", "w");
    printf("Enter the ratio of barrier to blank:\n");
    float ratio;
    scanf("%f", &ratio);
    for (int i = 0; i < length + 2; ++i)
    {
        for (int j = 0; j < width + 2; ++j)
        {
```

```
            if (i == 0 || i == length + 1 || j == 0 || j == wi
dth + 1)
                fprintf(fp, " # ");
            else
            {
                int flag = rand() % 100;
                if (flag < ratio * 100)
                    fprintf(fp, " # ");
                else
                    fprintf(fp, " - ");
            }
        }
        fprintf(fp, "\n");
    }
    fclose(fp);
}
void basicSetting(int length, int width)
{
    int flag;
    printf("Enter 1 if Maze is ready,or enter other key\n");
    scanf("%d", &flag);
    if (flag != 1)
    {
        srand(time(0));
        do
        {
            createMaze(length, width);
            printf("Created successfully!\n");
            printf("Enter 1 if you want to reset the maze,\
or enter 0.\n");
            scanf("%d", &flag);
        } while (flag);
        printf("Now you can amend the maze in maze.txt.\n");
        system("pause");
```

```
    }
}
int PathSearch(ptk *start, int **maze)
{
    int flag;
    if (start->pre == NULL && maze[start->x][start->y] == -3)
        return -1;
    else if (maze[start->x][start->y] == -1)
        return 1;
    maze[start->x][start->y] = -3;
    if (maze[start->x + 1][start->y] > -2)
        flag = PathSearch(addPath(start, 1), maze);
    else if (maze[start->x][start->y + 1] > -2)
        flag = PathSearch(addPath(start, 2), maze);
    else if (maze[start->x - 1][start->y] > -2)
        flag = PathSearch(addPath(start, 3), maze);
    else if (maze[start->x][start->y - 1] > -2)
        flag = PathSearch(addPath(start, 4), maze);
    else if (start->pre == NULL)
        return -1;
    else
        flag = PathSearch(withdrawPath(start), maze);
    return flag;
}
ptk *addPath(ptk *start, int sgn)
{
    ptk *new = (ptk *)malloc(sizeof(ptk));
    int x = start->x, y = start->y;
    switch (sgn)
    {
    case 1:
        x += 1;
        break;
    case 2:
```

```
            y += 1;
            break;
        case 3:
            x -= 1;
            break;
        case 4:
            y -= 1;
            break;
        }
        new->x = x;
        new->y = y;
        new->pre = start;
        new->next = NULL;
        start->next = new;
        return new;
}
ptk *withdrawPath(ptk *start)
{
        ptk *cup = start->pre;
        start->pre->next = NULL;
        free(start);
        return cup;
}
```

[算法说明]

```
//寻求最短路径:
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
typedef struct PathTrack
{
    int x, y;
    struct PathTrack *next;
} ptk;
void createMaze(int length, int width);
void basicSetting(int length, int width);
void markMaze(int **maze, ptk *dst, ptk *src);
void PathTrack(int **maze, ptk *start);
void freeLink(ptk *start);
void putMaze(int **maze, int length, int width);
int main()
{
    int length, width;
    printf("Enter the length and width of maze:\n");
    scanf("%d%d", &length, &width);
    basicSetting(length, width);
    FILE *fp = fopen("maze2.txt", "a+");
    int **maze = (int **)malloc((length + 2) * sizeof(int *));
    for (int i = 0; i < length + 2; ++i)
        maze[i] = (int *)malloc((width + 2) * sizeof(int));
    for (int i = 0; i < length + 2; ++i)
    {
        for (int j = 0; j < width + 2; ++j)
        {
            char tmp;
            fscanf(fp, " %c ", &tmp);
            if (tmp == '#')
                maze[i][j] = -1;
            else if (tmp == '-')
```

```
                maze[i][j] = 0;
        }
    }
    printf("Now enter the coordinates of start:\n");
    ptk *start = (ptk *)malloc(sizeof(ptk));
    scanf("%d%d", &start->x, &start->y);
    maze[start->x][start->y] = 0;
    start->next = NULL;
    printf("Now enter the coordinates of destination:\n");
    ptk *end = (ptk *)malloc(sizeof(ptk));
    scanf("%d%d", &end->x, &end->y);
    end->next = NULL;
    maze[end->x][end->y] = 1;
    clock_t beginning = clock();
    markMaze(maze, start, end);
    PathTrack(maze, start);
    clock_t ending = clock();
    if (maze[start->x][start->y] == 0)
        fprintf(fp, "\nSo SAD,it's no way!\n");
    else
    {
        fprintf(fp, "\nOne of the shortest Paths:\n");
        for (int i = 0; i < length + 2; ++i)
        {
            for (int j = 0; j < width + 2; ++j)
            {
                if (maze[i][j] == -1)
                    fprintf(fp, "  #  ");
                else if (maze[i][j] >= 0)
                    fprintf(fp, "  -  ");
                else
                    fprintf(fp, "[%03d]", -maze[i][j] - 1);
            }
            fprintf(fp, "\n");
```

```
        }
    }
    fprintf(fp, "Elapsed time:%f s", (double)(ending - beginni
ng) / CLOCKS_PER_SEC);
    fclose(fp);
    for (int i = 0; i < length + 2; ++i)
        free(maze[i]);
    free(maze);
    free(start);
    system("pause");
}
void createMaze(int length, int width)
{
    FILE *fp = fopen("maze2.txt", "w");
    printf("Enter the ratio of barrier to blank:\n");
    float ratio;
    scanf("%f", &ratio);
    for (int i = 0; i < length + 2; ++i)
    {
        for (int j = 0; j < width + 2; ++j)
        {
            if (i == 0 || i == length + 1 || j == 0 || j == wi
dth + 1)
                fprintf(fp, " # ");
            else
            {
                int flag = rand() % 100;
                if (flag < ratio * 100)
                    fprintf(fp, " # ");
                else
                    fprintf(fp, " - ");
            }
        }
        fprintf(fp, "\n");
```

```
    }
    fclose(fp);
}
void basicSetting(int length, int width)
{
    int flag;
    printf("Enter 1 if Maze is ready,or enter other key\n");
    scanf("%d", &flag);
    if (flag != 1)
    {
        srand(time(0));
        do
        {
            createMaze(length, width);
            printf("Created successfully!\n");
            printf("Enter 1 if you want to reset the maze,\
or enter 0.\n");
            scanf("%d", &flag);
        } while (flag);
        printf("Now you can amend the maze in maze.txt.\n");
        system("pause");
    }
}
void markMaze(int **maze, ptk *dst, ptk *src)
{
    if (src == NULL)
        return;
    ptk *tmp = src, *start = NULL, *new = NULL, *tail = NULL;
    while (tmp)
    {
        int step = maze[tmp->x][tmp->y];
        for (int i = 0; i < 4; ++i)
        {
            int x = tmp->x, y = tmp->y;
```

```
            switch (i)
            {
            case 0:
                ++x;
                break;
            case 1:
                ++y;
                break;
            case 2:
                --x;
                break;
            case 3:
                --y;
                break;
            }
            if (maze[x][y] == 0)
            {
                new = (ptk *)malloc(sizeof(ptk));
                new->x = x;
                new->y = y;
                new->next = NULL;
                if (tail != NULL)
                    tail = tail->next = new;
                else
                    tail = start = new;
                maze[x][y] = step + 1;
            }
        }
        tmp = tmp->next;
    }
    freeLink(src);
    if (maze[dst->x][dst->y] != 0)
        return;
    markMaze(maze, dst, start);
```

```
}
void PathTrack(int **maze, ptk *start)
{
    int x = start->x, y = start->y;
    int tracker = maze[x][y], step = maze[x][y];
    maze[x][y] = -2;
    while (--tracker)
    {
        if (maze[x + 1][y] == tracker)
            x += 1;
        else if (maze[x][y + 1] == tracker)
            y += 1;
        else if (maze[x - 1][y] == tracker)
            x -= 1;
        else if (maze[x][y - 1] == tracker)
            y -= 1;
        maze[x][y] = -(step - maze[x][y] + 2);
    }
}
void freeLink(ptk *start)
{
    while (start)
    {
        ptk *tmp = start->next;
        free(start);
        start = tmp;
    }
}
void putMaze(int **maze, int length, int width)
{
    for (int i = 0; i < length + 2; ++i)
    {
        for (int j = 0; j < width + 2; ++j)
            printf("%03d ", maze[i][j]);
```

```
        printf("\n");
    }
    system("pause");
}
```

算法说明

附录 II

```
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include <time.h>
#define STACK_INIT_SIZE 100
#define BLOCK_SIZE 50
typedef struct stack
{
    int size;
    int top;
    int *data;
} stack;
void InitStack(stack *stk);
void PushStack(stack *stk, int n);
int PopStack(stack *stk);
void InflateStack(stack *stk);
void newQSort(int arr[], int size, stack *stk);
void printArray(int array[], int size);
void swap(int *x, int *y);
int main()
{
    int size, flag;
    stack stk;
    InitStack(&stk);
    printf("Enter the size of array:\n");
    scanf("%d", &size);
    int *array = (int *)malloc(size * sizeof(int));
    srand(time(NULL));
    for (int i = 0; i < size; ++i)
        array[i] = rand();
    printf("Enter 0 to make the Array invisible,or enter any o
ther numbers.\n");
```

```
    scanf("%d", &flag);
    if (flag)
    {
        printf("the initial Array:\n");
        printArray(array, size);
    }
    clock_t start = clock();
    newQSort(array, size, &stk);
    clock_t end = clock();
    if (flag)
    {
        printf("the random Array in order:\n");
        printArray(array, size);
    }
    printf("Elapsed time:%f s\n", (double)(end - start) / CLOC
KS_PER_SEC);
    free(stk.data);
    free(array);
}
void InitStack(stack *stk)
{
    assert(stk);
    stk->top = 0;
    stk->size = STACK_INIT_SIZE;
    stk->data = (int *)malloc(STACK_INIT_SIZE * sizeof(int));
}
void PushStack(stack *stk, int n)
{
    if (stk->size == stk->top)
        InflateStack(stk);
    stk->data[stk->top++] = n;
}
int PopStack(stack *stk)
{
```

```
    assert(stk);
    if (stk->size - stk->top > 100)
        stk->data = (int *)realloc(stk->data, (stk->size -
= 2 * BLOCK_SIZE) * sizeof(int));
    return stk->data[(stk->top--) - 1];
}
void InflateStack(stack *stk)
{
    stk->data = (int *)realloc(stk->data, (stk->size += BLOCK_
SIZE) * sizeof(int));
    assert(stk->data);
}
void newQSort(int arr[], int size, stack *stk)
{
    int head, foot, first, last, mid, pivot;
    PushStack(stk, size - 1);
    PushStack(stk, 0);
    while (stk->top)
    {
        first = head = PopStack(stk);
        last = foot = PopStack(stk);
        if (head >= foot)
            continue;
        mid = rand() % (foot - head + 1) + head;
        pivot = arr[mid];
        while (mid < last && arr[last] >= pivot)
            --last;
        swap(&arr[mid], &arr[last]);
        while (first < last)
        {
            while (first < last && arr[first] <= pivot)
                ++first;
            swap(&arr[first], &arr[last]);
            while (first < last && arr[last] >= pivot)
```

```
                --last;
            swap(&arr[first], &arr[last]);
        }
        PushStack(stk, last - 1);
        PushStack(stk, head);
        PushStack(stk, foot);
        PushStack(stk, first + 1);
    }
}
void printArray(int array[], int size)
{
    for (int i = 0; i < size; ++i)
        printf("%d ", array[i]);
    printf("\n");
}
void swap(int *x, int *y)
{
    int cup = *x;
    *x = *y;
    *y = cup;
}
```

[算法说明]