

**学生姓名 杨凯楠**

**学 号** 8208201004

**专业班级 信息安全2002班**

**指导教师** 石峰

**学 院** 计算机学院

**完成时间** 2021年11月18日

算法设计与分析

实验报告(递归与分治)

1. **实验目的**

理解递归算法的思想和递归程序的执行过程，并能熟练编写递归程序。

掌握分治算法的思想，对给定的问题能设计出分治算法予以解决。

1. **实验内容**

1.找第k小值问题

2.最近点对问题

**三、具体设计**

1、**找k小值问题**

分析：找k小值的程序中，我采取的是类似快排的算法，此算法是一种平均情况下时间复杂度为O(n)的快速选择算法，用到的是快速排序中的第一步，将第一个数作为中枢，使大于它的所有数放到它右边，小于它的所有数放到它左边。之后比较该中枢的最后位置i与k的大小，若i比k小，说明第k小的元素在i的右半段，之后对i的右半段进行快速选择；若i比k大，说明第k小的元素在i的左半段，之后对i的左半段进行快速选择；若i正好等于k，则直接返回。

概要设计：

**1.选择中枢：**如下，为了提高快速选择的效率，最好尽可能的选择数值居中的数作为中枢。

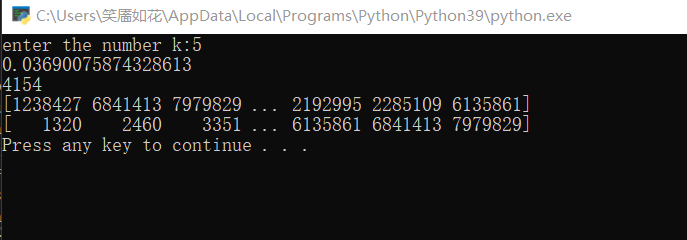
2.对区间[left, right-1]以中枢nums[right-1]**进行一次快速排序**，之后大于nums[right-1]的数全在它的右边，小于nums[right-1]的数全在它的左边。

3.比较i和k的大小：若求序列中的第k大的数，现在由步骤2已经知道一次选择排序后的中枢下标为i，说明前面i个数比pivot大，后面right-i个数比pivot大。如果k < i，说明第k大的数在前面i个数中；如果k > i，说明第k大的数在后面的right-i中；如果i==k，直接返回答案。对于前两种情况只需要对pivot的左半段或者右半段中寻找即可。

详细设计：

1. **import** numpy as np
2. **import** time
3. **import** copy
4. **def** creat(min,max,tot):
5. num=np.random.randint(min,max,tot)
6. **return** num
8. **def** get\_k(num,left,right,k):
9. **if** len(num)<k **or** k<0:
10. **return**
11. temp\_k=get\_index(num,left,right)
12. **if** k == temp\_k+1:
13. **return** num[temp\_k]
14. **if** k<temp\_k+1:
15. **return** get\_k(num,left,temp\_k-1,k)
16. **else**:
17. **return** get\_k(num,temp\_k+1,right,k)
19. **def** swap(num,a,b):
20. temp=num[a]
21. num[a]=num[b]
22. num[b]=temp
24. **def** get\_index(num,left,right):
25. i,j=left,right
26. **while** True:
27. **while** i<right **and** num[i]<num[left]:
28. i+=1
29. **while** left<j **and** num[left]<num[j]:
30. j-=1
31. **if** i>=j:
32. **break**
33. **else**:
34. swap(num,i,j)
35. swap(num,left,j)
36. **return** j
37. **def** main():
38. min,max,tot=1,10000000,10000
39. num=creat(min,max,tot)
40. nums=copy.deepcopy(num)
41. k=input("enter the number k:")
42. starttime=time.time()
43. numk=get\_k(num,0,tot-1,int(k))
44. endtime=time.time()
45. **print**(endtime-starttime)
46. **print**(numk)
47. **print**(nums)
48. **print**(num)
49. main()

**运行结果：**



调试分析：本次测试随机产生了一万个数，经过0.0369s寻找，找到第5小的数为4154。可见算法运行效率还是很高的。

2、**最近点对问题**

分析：

分解

对所有的点按照x坐标（或者y）从小到大排序（排序方法时间复杂度O(nlogn)。

根据下标进行分割，使得点集分为两个集合。

解决

递归的寻找两个集合中的最近点对。

取两个集合最近点对中的最小值min(dis\_left， dis\_right)。

合并

最近距离不一定存在于两个集合中，可能一个点在集合A，一个点在集合B，而这两点间距离小于dis。

难点在于合并。即一个点在集合A，一个在集合B中的情况，可以针对此情况，用之前分解的标准值，即按照x坐标（或者y）从小到大排序后的中间点的x坐标作为mid，划分一个 [mid−dis,mid+dis]区域，如果存在最小距离点对，必定存在这个区域中。

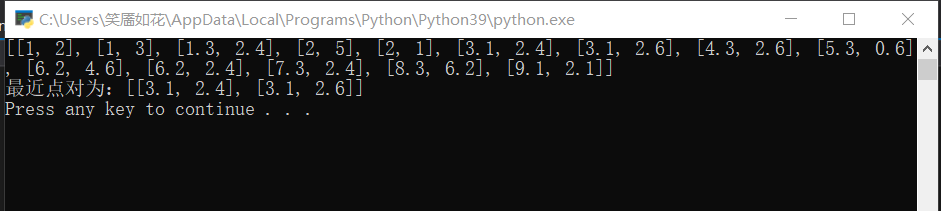
概要设计：一、分解 二、解决 三、合并

详细设计：

1. **from** math **import** sqrt
2. **import** numpy as np
3. **def** nearest\_points(s):
4. len1 = len(s)                                 #len1为点对的数量
5. #print("len=",len1)
6. left = s[0:int(len1/2)]
7. right = s[int(len1/2):]
8. mid\_x = (left[-1][0]+right[0][0])/2               #midx为中间坐标x值
9. **if** len(left) > 2:
10. lmin = nearest\_points(left)    #左侧部分最近点对
11. **else**:   lmin = left
12. **if** len(right) > 2:
13. rmin = nearest\_points(right)   #右侧部分最近点对
14. **else**:   rmin = right
16. **if** len(lmin) >1:
17. disl = get\_distance(lmin)
18. **else**: disl = float("inf")                      #将其设置为正无穷
19. **if** len(rmin) >1:
20. disr = get\_distance(rmin)
21. **else**: disr = float("inf")
23. d = min(disl, disr)   #最近点对距离
25. mid\_min=[]
26. **for** i **in** left:                             #以左部分的点为基准点，找右半部分
27. **if** mid\_x-i[0]<=d :                                          #如果左侧部分与中间线的距离<=d
28. **for** j **in** right:
29. **if** j[0]-i[0]<=d **and** abs(i[1]-j[1])<=d:       #如果右侧部分点在i点的(d,2d)之间
30. **if** get\_distance((i,j))<=d:
31. mid\_min.append([i,j])                      #ij两点的间距若小于d则加入队列
32. **if** mid\_min:
33. dic=[]                                            #定义一个内容为字典的列表
34. **for** i **in** mid\_min:
35. dic.append({get\_distance(i):i})
36. dic.sort(key=**lambda** x: x.keys())                #按字典的键排序
37. **return** list(dic[0].values())[0]
38. **elif** disl>disr:
39. **return** rmin
40. **else**:
41. **return** lmin

44. # 求点对的距离
45. **def** get\_distance(min):
46. **return** sqrt((min[0][0]-min[1][0])\*\*2 + (min[0][1]-min[1][1])\*\*2)
48. **def** main(s):
49. s.sort( key=**lambda** s: s[0])
50. **print**(s)
51. NP = nearest\_points(s)
52. **print**(NP)
53. s=[[1,2],[1,3],[2,5],[1.3,2.4],[6.2,4.6],[9.1,2.1],[2,1],[3.1,2.4],[5.3,0.6],[6.2,2.4],[8.3,6.2],[7.3,2.4],[4.3,2.6],[3.1,2.6]]
54. main(s)

实现：



测试分析：经过计算，最近点对确实为（3.1,2.4）与（3.1,2.6）之间。

1. **实验心得**

完成本次实验时我正好正在学Python，所以萌发了用Python写本实验的念头，然而在写了三个算法实验后，发现Python写的算法程序可读性并不强，并且逻辑也并不是很明确，在此以后将会有所改变。分治法是一种很奇妙的思想，它将原本大的问题分成性质相同的规模更小的问题，在求得更小规模的解后合并，逐步合并成最终解。经过这次实验，我初步体会到，算法不是一门课程，不是一项技术，**而是一种思想**。



**学生姓名 杨凯楠**

**学 号** 8208201004

**专业班级 信息安全2002班**

**指导教师** 石峰

**学 院** 计算机学院

**完成时间** 2021年11月21日

算法设计与分析

实验报告(回溯)

1. **实验目的：**熟练掌握回溯算法
2. **实验内容**

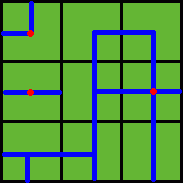
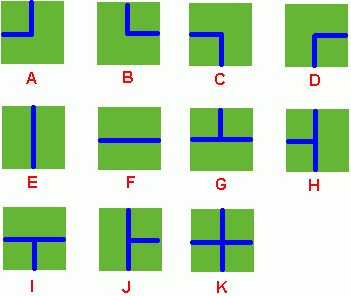
1.装载问题

有两艘船，载重量分别是c1、 c2，n个集装箱，重量是wi (i=1…n)，且所有集装箱的总重量不超过c1+c2。确定是否有可能将所有集装箱全部装入两艘船。

提示：求出不超过c1的最大值max，若总重量－max < c2则能装入到两艘船。

2.农场灌溉问题（ZOJ2412）

一农场由图所示的十一种小方块组成，蓝色线条为灌溉渠。若相邻两块的灌溉渠相连则只需一口水井灌溉。给出若干由字母表示的最大不超过50×50具体由(m，n)表示，的农场图，编程求出最小需要打的井数。每个测例的输出占一行。当M=N=-1时结束程序。



**Sample Input**

2 2

DK

HF

3 3

ADC

FJK

IHE

-1 -1

**Sample Output**

2

3

1. **具体设计**

**1.装载问题**

有两艘船，载重量分别是c1、 c2，n个集装箱，重量是wi (i=1…n)，且所有集装箱的总重量不超过c1+c2。确定是否有可能将所有集装箱全部装入两艘船。

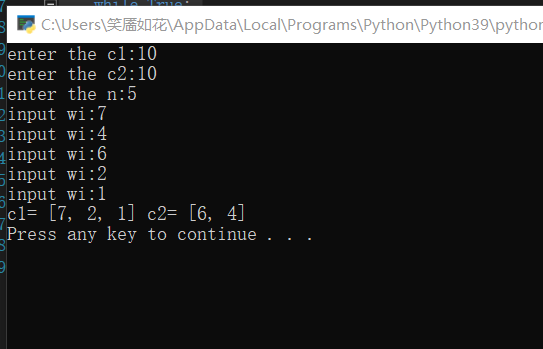
提示：求出不超过c1的最大值max，若总重量－max < c2则能装入到两艘船。

**分析**：本题较为简单，通过递归，采用深度优先遍历，边遍历边生成树在遍历过程中随时查看是否已经到达死节点。最终寻找出最优解。

**详细设计：**

1. **def** set():
2. c1=int(input("enter the c1:"))
3. c2=int(input("enter the c2:"))
4. n=int (input("enter the n:"))
5. a=[]
6. maxw=0
7. **for** i **in** range(n):
8. a.append(int(input("input wi:")))
9. maxw+=a[i]
10. **if** maxw>c1+c2:                     #当输入错误的数据返回FALSE，并在主函数中重新输入数据
11. **return** False,False,False,False
12. **return** c1,c2,n,a

15. **def** solution(c1,c2,w):
16. w.sort(reverse=True)
17. temp\_c=tot=0
18. a=[]
19. b=[]
20. **for** i **in** w:
21. temp\_c+=i
22. tot+=i
23. **if** temp\_c<=c1:
24. a.append(i)
25. **else**:
26. temp\_c-=i
27. b.append(i)
28. **if** tot-temp\_c<=c2:
29. **return** True,a,b
30. **else**:
31. **return** False,False,False
33. **def** main():
34. **while** True:
35. c1,c2,n,w=set()
36. **if** c1:
37. **break**
38. **print**("wrong data!")
39. flag,soc1,soc2=solution(c1,c2,w)
40. **if** flag:
41. **print**("c1=",soc1,"c2=",soc2)
42. **else**:
43. **print**("wrong")
44. main()

**实现：**

调试分析：由此可见在该输入条件下是可以装载进去的。有一个小技巧是，先对一只船做最大容量装载，然后用总重量减去最大容量判断第二只船能不能装下剩下的集装箱。

**2.农场灌溉问题（ZOJ2412）**

一农场由图所示的十一种小方块组成，蓝色线条为灌溉渠。若相邻两块的灌溉渠相连则只需一口水井灌溉。给出若干由字母表示的最大不超过50×50具体由(m，n)表示，的农场图，编程求出最小需要打的井数。每个测例的输出占一行。当M=N=-1时结束程序。

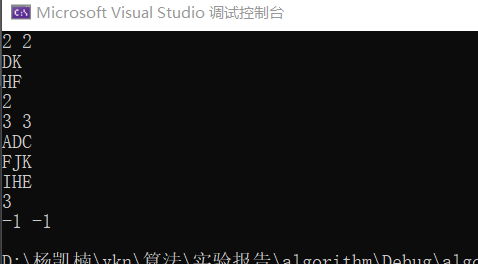
**分析：**本题实际上就是图的连通度的问题，采用深度优先遍历，遍历一遍就可得出连通度。在将农田抽象化时，采用了把农田看成一个点，连通的水渠看成无向边的方法。初始化农田时，只对单块农田进行初始化，当他上下左右伸展出水渠时就定义为二维数组的1，否则为0。两点之间存在无向边的条件是两点相邻的地方都伸展出了水渠。

**详细设计**：

1. #include<stdio.h>
2. #include<iostream>
3. #include<vector>
4. using namespace std;
5. void set\_canal(int a[][4]) {
6. a[0][0] = a[0][1] = 1;
7. a[1][1] = a[1][2] = 1;
8. a[2][0] = a[2][3] = 1;
9. a[3][2] = a[3][3] = 1;
10. a[4][1] = a[4][3] = 1;
11. a[5][0] = a[5][2] = 1;
12. a[6][0] = a[6][1] = a[6][2] = 1;
13. a[7][0] = a[7][1] = a[7][3] = 1;
14. a[8][0] = a[8][2] = a[8][3] = 1;
15. a[9][1] = a[9][2] = a[9][3] = 1;
16. a[10][0] = a[10][1] = a[10][2] = a[10][3] = 1;
17. }
18. void set\_canal\_now(vector<vector<int> >& a,int row,int col) {
19. char b;
20. scanf\_s("%c", &b, 1);
21. **for** (int i = 0; i < row; i++) {
22. **for** (int j = 0; j < col; j++) {
23. scanf\_s("%c", &b, 1);
24. a[i][j] = b - 65;
25. }
26. scanf\_s("%c",&b,1);
27. }
29. }
30. //a里面存的是字母对应的数字
31. void count\_recursion(vector<vector<int> >& a, int row, int b[][4], int col, vector<vector<int> >& flag,int i,int j) {
32. **if** (flag[i][j] == 0&&(i>=0&&i<row)&&(0<=j&&j<col)) {
33. flag[i][j] = 1;
34. **if**(i<row-1)
35. **if**(b[a[i][j]][3]==1&&b[a[i+1][j]][1]==1)
36. count\_recursion(a, row, b, col, flag, i+1, j);
37. **if**(i>0)
38. **if**(b[a[i][j]][1]==1&&b[a[i-1][j]][3]==1)
39. count\_recursion(a, row, b, col, flag, i -1, j);
40. **if**(j>0)
41. **if** (b[a[i][j]][0] == 1 && b[a[i][j-1]][2] == 1)
42. count\_recursion(a, row, b, col, flag, i , j-1);
43. **if**(j<col-1)
44. **if** (b[a[i][j]][2] == 1 && b[a[i][j+1]][0] == 1)
45. count\_recursion(a, row, b, col, flag, i , j+1);
46. }

49. }
50. int countl(vector<vector<int> >& a, int row, int col, int b[][4]) {
51. int i, j,count=0;
52. vector<vector<int> > flag(row, vector<int>(col));
53. **for** (i = 0; i < row; i++)
54. **for** (j = 0; j < col; j++)
55. flag[i][j] = 0;
56. **for** (i = 0; i < row; i++) {
57. **for** (j = 0; j < col; j++) {
58. **if** (flag[i][j] == 0) {
59. count++;
60. count\_recursion(a, row, b, col, flag, i, j);
61. }
62. }
63. }
64. **return** count;
65. }
66. int main(void) {
67. int a[11][4] = { 0 };
68. set\_canal(a);
69. int row, col;
70. cin >> row >> col;
71. **while** (row != -1 && col != -1) {
72. vector<vector<int> > b(row, vector<int>(col));
73. set\_canal\_now(b, row, col);
74. int count\_jing = countl(b, row, col, a);
75. cout << count\_jing << endl;
76. cin >> row >> col;
77. }
78. **return** 0;
79. }

**实现**：



**调试分析**：测试结果显示没有问题。在调试过程中，我用容器vector库创建了一个二维空间来存储农田信息，每个空间单元为一个农田种类。

1. **实验心得**

本次试验充分体现了用算法解决实际问题的特点，当拿到一个具体问题是，需要迅速将其抽象化，判断适用什么数据结构，采用什么算法，可以将其等效为哪一种已经学过的问题。可以说本次实验非常具有实践意义。经过本次实验，我掌握了回溯的算法，深刻体会到了算法的妙处，开始进一步尝试用算法问题解决生活中的问题，感觉收获颇丰而又愈发感觉自身知识的欠缺，今后将继续努力。



**学生姓名 杨凯楠**

**学 号** 8208201004

**专业班级 信息安全2002班**

**指导教师** 石峰

**学 院** 计算机学院

**完成时间** 2021年11月18日

算法设计与分析

实验报告(搜索)

1. **实验目的**：熟练掌握搜索算法
2. **实验内容**

1.电子老鼠闯迷宫

如下图12×12方格图，找出一条自入口（2，9）到出口（11，8）的最短路径。

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |

测试数据如下：

12 10 7 1 8

2. 分酒问题

有一酒瓶装有8斤酒，没有量器，只有分别装5斤和3斤的空酒瓶。设计一程序将8斤酒分成两个4斤，并以最少的步骤给出答案。

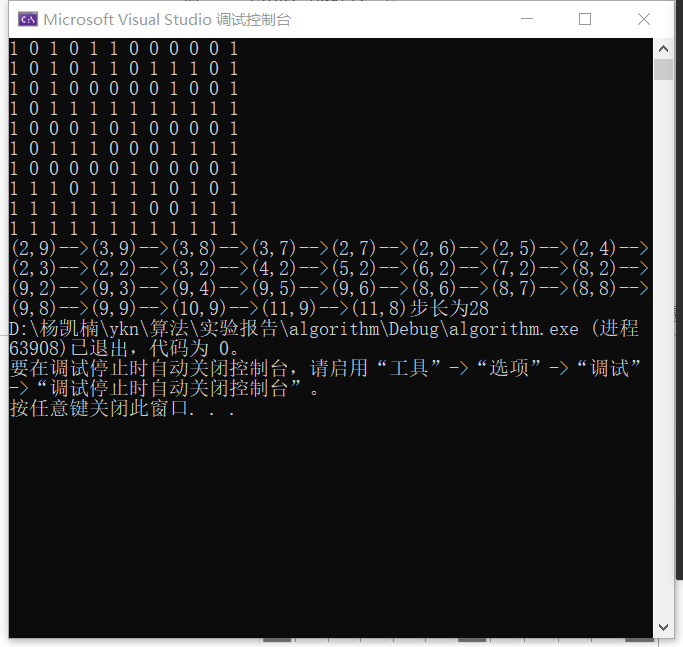
1. **具体设计**
2. **电子老鼠闯迷宫**

**分析**：本题的解题思路是将其等效为广度优先遍历，a[][]表示方格，save[][]表示【当前已经遍历过的点的序号】【横纵坐标】，path[]【内为当前遍历序号】，存储的内容为上一序号。如果当前的位置可以走的话就尾加1，把当前位置的父结点加入队列；当现在的父结点已经没有子结点的话，就把头加1，把上一个已经没有子结点的父结点退出队列。  
只要是走过的路线都要把它赋值为1（障碍），防止有重复的路线，而且先到某一个路线的一定是最快最优的。

**详细设计：**

1. #include<iostream>
2. #include<cstdio>
3. **using** **namespace** std;
4. **const** **int** mov[4][2] = { {0,1 }, {0, -1},{ -1, 0},{ 1, 0} };
5. **int** size = 12;
6. **int** a[12][12];//方格状况
7. **int** save[12 \* 12][2];//[当前已经遍历过的点的序号][横纵坐标】
8. **int** path[12 \* 12];//【内为当前遍历序号】存储的内容为上一序号
9. **int** a1, b1;//出发点坐标
10. **int** a2, b2;//终点坐标
11. **int** countlen = 0;
12. **bool** judge(**int** i, **int** j) {
13. **if** (a[i][j] == 1)
14. **return** **false**;
15. **return** **true**;
16. }
17. **void** coutmy(**int** a) {
18. **if** (a == 1)
19. **return**;
20. countlen++;
21. coutmy(path[a]);
22. cout << "-->" <<"("<< save[a][0]+1 << "," << save[a][1]+1<<")";
23. }
24. **void** solution() {
25. **int** sign1=0, sign2=1;//sign1为空处遍历过的序号，sign2为正确路径序号path[]
26. path[sign2] = sign1;//第一个遍历的前级肯定为出发点
27. save[1][0] = a1; save[1][1] = b1;
28. a[a1][b1] = 1;//出发点肯定走完了设为1
29. **do** {
30. sign1++;
31. **for** (**int** i = 0; i < 4; i++) {
32. **if** (judge(save[sign1][0] + mov[i][0], save[sign1][1] + mov[i][1])) {
33. sign2++;
34. path[sign2] = sign1;
35. save[sign2][0] = save[sign1][0] + mov[i][0];
36. save[sign2][1] = save[sign1][1] + mov[i][1];
37. a[save[sign2][0]][save[sign2][1]] = 1;
38. }
39. **if** (save[sign2][0] == a2-1 && save[sign2][1] == b2-1) {
40. **int** count = 0;
41. cout << "(" << a1 << "," << b1 << ")";
42. coutmy(sign2);
43. cout << "步长为" << countlen;
44. **return**;
45. }
46. }
47. } **while** (sign1 < sign2);
49. }
50. **int** main(**void**) {
51. **int** i, j;
53. cin >> a1 >> b1 >> a2 >> b2;
55. **for** (i = 0; i < 12; i++)
56. **for** (j = 0; j < 12; j++)
57. cin >> a[i][j];
58. solution();
59. }

**运行结果**：



**调试分析**：本题的一个难点是路径的保存，这也耗费了我不少时间，路径的保存大致思路是建立一个数组，让数组记录前一个位置的坐标。在到达终点输出路径时用递归的方法从头到尾输出。

**2. 分酒问题**

有一酒瓶装有8斤酒，没有量器，只有分别装5斤和3斤的空酒瓶。设计一程序将8斤酒分成两个4斤，并以最少的步骤给出答案。

**分析**：分酒问题采用广度优先遍历，将每一种可能的倒酒方式都全部考虑他下一步倒酒的所有情形，直到最先达到目标情况。

概要设计：

1. **int** captain[] = { 8,5,3 };//容器标准容量
2. **int** des[3];
3. **int** cou = 0;
4. **int** step = 0;
5. **struct** state {
6. **int** now[3];//现在各杯中酒量;
7. **int** xulie;//在数组中位置
8. **int** qiandao;//实现这一步骤的前一步骤，前导
9. };
10. queue<state> q;//广度优先搜索队列
11. vector<state> p;//顺序数组
12. //判断是否可以进一步进行倒酒动作,i向j里面倒酒。
13. **bool** check(**int** now[], **int** i, **int** j) ；
15. //输出函数
16. **void** out(**int** a)；
18. //广度优先遍历，可以求得最优解
19. **void** bfs(state s)；

**详细设计**：

1. **bool** check(**int** now[], **int** i, **int** j) {
2. **if** (now[i] == 0 || now[j] == captain[j])
3. **return** **false**;
4. **return** **true**;
5. }
6. //输出函数
7. **void** out(**int** a) {
8. step += 1;
9. **if** (p[a].xulie == 0) {
10. cout << p[a].now[0] << " " << p[a].now[1] << " " << p[a].now[2] << endl;
11. **return**;
12. }
13. out(p[a].qiandao);
14. cout << p[a].now[0] << " " << p[a].now[1] << " " << p[a].now[2] << endl;
15. }
16. //广度优先遍历，可以求得最优解
17. **void** bfs(state s) {
18. state tmp, tempr;
19. s.xulie = 0;
20. s.qiandao = 0;
21. p.push\_back(s);
22. q.push(s);
23. **while** (!q.empty()) {
24. **bool** flag = **true**;
25. tmp = q.front();
26. **for** (**int** i = 0; i < 3; i++) {
27. **if** (tmp.now[i] != des[i])
28. flag = **false**;
29. }
30. **if** (flag){
31. cout << "倒酒步骤为：" << endl;
32. out(tmp.xulie);
33. cout << "共用" << step-1 << "步" << endl;
34. **return**;
35. }
36. **for** (**int** i = 0; i < 3; i++) {
37. **if** (tmp.now[i] > 0) {
38. **for** (**int** j = 0; j < 3; j++) {
39. **if** (i != j && check(tmp.now, i, j)) {
40. **int** tmpi = tmp.now[i], tmpj = tmp.now[j];
41. tempr = tmp;
42. //daojiu
43. tmpj += tmp.now[i];
44. tmpi -= tmp.now[i];
45. //panduan
46. **if** (tmpi == 0 && tmpj == captain[j]) {//ikong,
47. tempr.now[i] = tmpi;
48. tempr.now[j] = tmpj;
49. cou++;
50. tempr.xulie = cou;
51. tempr.qiandao =  tmp.xulie;
52. p.push\_back(tempr);
53. q.push(tempr);
54. }
55. **else** **if** (tmpi == 0 && tmpj < captain[j]) {
56. tempr.now[i] = tmpi;
57. tempr.now[j] = tmpj;
58. cou++;
59. tempr.xulie = cou;
60. tempr.qiandao = tmp.xulie;
61. p.push\_back(tempr);
62. q.push(tempr);
63. }
64. //ibukong
65. tmpi = tmp.now[i], tmpj = tmp.now[j];
66. tmpi -= (captain[j] - tmpj);
67. tmpj = captain[j];
68. **if** (tmpi > 0 && tmpi < captain[i]) {
69. tempr.now[i] = tmpi;
70. tempr.now[j] = tmpj;
71. cou++;
72. tempr.xulie = cou;
73. tempr.qiandao = tmp.xulie;
74. p.push\_back(tempr);
75. q.push(tempr);
76. }
77. }
78. }
79. }
80. }
81. q.pop();
82. }
83. }



88. **int** main() {
89. state s;
90. **for** (**int** i = 0; i < 3; i++) {
91. cin >> des[i];
92. }
93. s.now[0] = 8;
94. s.now[1] = 0;
95. s.now[2] = 0;
96. s.xulie = 0;
97. s.qiandao = 0;
98. bfs(s);
99. **return** 0;
100. }

**运行结果**：



。

**调试分析**：

同样，本道题在记录倒酒路径的问题上耗费了我不少功夫，但是总的而言还是上一道题所说过得算法思维。在倒酒的时候需要考虑i空和i不为空的情况。

1. **心得体会**

经过本次实验，我学习掌握了BFS以及DFS算法，也通过不断摸索，学习到了利用数组存储遍历路径的方法。感悟颇多，最大的感受是数学知识与数学分析方法在算法中的重要性，当一道题没有头绪时，用笔认真在纸上分析一下往往能使自己的逻辑变得清晰明了，然后纸上的来说终觉浅，绝知此事要躬行，我将它用代码的形式写出来，获益颇多。