

算法第四次作业

2020年5月13日 14:48

朱一帆-智科-222018321212058

4-1 会场安排问题

假设要在足够多的会场里安排一批活动，并希望使用尽可能少的会场。设计一个有效的贪心算法进行安排。（这个问题实际上是著名的图着色问题。若将每个活动作为图的一个顶点，不相容的活动用边连接。使相邻顶点着有不同颜色的最小着色数，相当于要找最小会场数。）

设计思路：先将总的活动数读取，再读取每个活动的开始时间和结束时间。根据每个活动的时间建立一个矩阵 $g[i][j]$ ，含义如下

$$g[i][j] = \begin{cases} 0, & \text{当两个活动相容} \\ 1, & \text{当两个活动不相容} \end{cases}$$

根据建立起来的矩阵进行染色。首先将所有点染成一个颜色，然后扫描不相容点，并给不相容点从第一种颜色开始染不相同的颜色，从而能达到最少的染色数。

```
MeetingArrange (全局范围)
1 // MeetingArrange.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束
2 //
3
4 #include <iostream>
5 #define MAXSIZE 100
6 using namespace std;
7
8 class meeting //活动类
9 {
10 public:
11     int i; //活动序号
12     int s; //活动开始时间
13     int e; //活动终止时间
14 };
15
16 int main()
17 {
18     int k;
19     cin >> k; //读取活动数
20     int a, b;
21     meeting m[MAXSIZE];
22     for (int i = 1; i < k+1; i++)
23     { //录入时间
24         cin >> a >> b;
25         m[i].i = i;
26         m[i].s = a;
27         m[i].e = b;
28     }
29     //建立相容矩阵
30     int** g = new int*[k+1];
31     for (int i = 0; i < k+1; i++)
32         g[i] = new int[k+1];
33
34     for (int i = 1; i <= k; i++)
35     { //上三角矩阵，不相容为1，相容为0
36         for (int j = 1; j <= i; j++)
37             g[i][j] = 0;
38         for (int j = i + 1; j <= k; j++)
39         {
40             if (m[i].e >= m[j].s)
41                 g[i][j] = 1;
42             else
43                 g[i][j] = 0;
44         }
45     }
46 }
```

Handwritten red annotations: A red box highlights the nested loops for building the compatibility matrix (lines 34-45). To the right of the box, the text $O(n)$ is written twice, with a horizontal line connecting them, indicating the time complexity of this section.

```

43         g[i][j] = 0;
44     }
45 }
46
47 int* x = new int[k + 1]; //染色数组
48 for (int i = 1; i <= k; i++)
49     x[i] = 1; //同染一种颜色
50 for (int i = 1; i <= k; i++)
51 {
52     int n = 0; //每次寻找不相容活动，从第一种颜色开始染
53     for (int j = 1; j <= k; j++)
54     {
55         if (g[i][j] == 1)
56         {
57             x[j] += n++;
58             if (x[i] == x[j]) //染到相同的颜色时增加一种颜色
59                 x[j]++;
60         }
61     }
62 }
63
64 int max = 0;
65 for (int i = 1; i <= k; i++)
66     if (x[i] > max) //寻找颜色最大数
67         max = x[i];
68 cout << "\n" << max << endl;
69
70 }
71

```

Handwritten annotations:

- $O(n)$ is written in red next to the inner loop (lines 53-59).
- $O(n)$ is written in red next to the outer loop (lines 50-59).
- $O(n^2)$ is written in red next to the entire loop structure (lines 50-62).

运行结果如下所示

```

C:\WINDOWS\system32\cmd.exe
5
1 23
12 28
25 35
27 80
36 50
3
请按任意键继续. . .

```

```

C:\WINDOWS\system32\cmd.exe
4
1 15
14 21
20 28
29 40
2
请按任意键继续. . .

```

算法实现题 硬币找钱问题

问题描述：设有6种不同面值的硬币，各硬币的面值分别为5分、1角、2角、5角、1元和2元。现要用这些面值的硬币来购物和找钱。购物时可以使用各种面值的硬币个数存于数组Coins[1:6]中，商店里各面值的硬币有足够多。在一次购物中希望使用最少硬币个数。

设计思路：先使用贪心算法让顾客出小于需付款的最大金额，并使币值不断减小，最后凑得需付的钱，这是让顾客单方面付款没有找钱的方法。第二种是让顾客出大于需付款的钱，并不断凑使得能补的钱刚好是现有的币值。最后比较两种方法，选取使用硬币数最少的方法。

```

coinschange.cpp*  X
coinschange (全局范围)
1 // coinschange.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
2 //
3
4 #include <iostream>
5 #define MAXSIZE 100
6 #define MIN(a,b) ((a) < (b) ? (a) : (b))
7 using namespace std;
8
9 bool enough(int c[], int coins[], int need) //判断所有的钱总和是否能付款
10 {
11     double sum = 0;
12     int count = 0;
13     for (int i = 0; i < 6; i++)
14         sum += c[i] * coins[i];

```

coinschange.cpp* X

coinschange

(全局范围)

```
1 // coinschange.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
2 //
3
4 #include <iostream>
5 #define MAXSIZE 100
6 #define MIN(a,b) ((a) < (b) ? (a) : (b))
7 using namespace std;
8
9 bool enough(int c[],int coins[],int need)
10 {
11     //判断所有的钱总和是否能付款
12     double sum = 0;
13     int count = 0;
14     for (int i = 0; i < 6; i++)
15         sum += c[i] * coins[i];
16     if (sum < need)
17     {
18         cout << "not enough money" << endl;
19         return false;
20     }
21     return true;
22 }
23
24 int pay(int c[],int coins[],int need)
25 {
26     //顾客凑钱，没有补款
27     //贪心，从最大的钱开始凑
28     while (need)
29     {
30         //凑够钱后退出
31         while (c[i] > need) i--;
```

```
32         if (coins[i])
33             //如果硬币足够，需付款减少，该硬币额减一，使用币数加一
34             {
35                 need -= c[i];
36                 coins[i]--;
37                 n++;
38             }
39         else
40             {
41                 i--;
42             }
43     }
44     return n;
45 }
46 bool complete(int c[],int change)
47 {
48     //判断补得钱是否是现有的币值，是返回true
49     int i = 0;
50     while (c[i] < change) i++;
51     if (c[i] == change) return true;
52     else return false;
53 }
54 int change(int c[], int coins[], int need)
55 {
56     //商家找钱的方法
57     int i = 0, j = 0;
58     int change = 0;
59     int n = 0;
60     int shop[6] = { MAXSIZE, MAXSIZE, MAXSIZE, MAXSIZE, MAXSIZE, MAXSIZE };
61     while (c[i] < need) i++; //出比需付款大的币值
62     while (!coins[i]) i++; //同时该币值要有硬币
63     if (i > 6)
64         return -1; //最大币值不够付款，不能使用该方法
```

```

63     change = c[i] - need;
64     coins[i]--;
65     n += 1;
66     while (!complete(c, change))
67     {
68         if (coins[j])
69         {
70             change += c[j];
71             n += 1;
72             coins[j]--;
73         }
74         else j++;
75         if (!complete(c, change))
76         {
77             change -= c[j++];
78             n -= 1;
79         }
80     }
81     n += pay(c, shop, change);
82
83     return n;
84 }
85
86
87 int main()
88 {
89     int c[6] = { 5, 10, 20, 50, 100, 200 }; //币值以分为单位
90     int coins[6];
91     for (int i = 0; i < 6; i++)
92         cin >> coins[i];
93     double need;

```

```

94     cin >> need;
95     int cost = (int)(need * 100);
96     if (enough(c, coins, cost))
97     {
98         int a = pay(c, coins, cost);
99         int b = change(c, coins, cost);
100        if (b < 0) cout << a;
101        else
102            cout << MIN(a, b) << endl;
103    }
104 }

```

运行结果如下

C:\WINDOWS\system32\cmd.exe

```

2 4 2 2 1 0 0.95
2
请按任意键继续. . .

```

C:\WINDOWS\system32\cmd.exe

```

2 4 2 0 1 0 0.55
3
请按任意键继续. . .

```