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

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

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)

```cpp
// MeetingArrange.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束
//

#include <iostream>
#define MAXSIZE 100
using namespace std;

class meeting
{                       //活动类
public:
    int i;              //活动序号
    int s;              //活动开始时间
    int e;              //活动终止时间
};

int main()
{
    int k;
    cin >> k;           //读取活动数
    int a, b;
    meeting m[MAXSIZE];
    for (int i = 1; i < k+1; i++)
    {                   //录入时间
        cin >> a >> b;
        m[i].i = i;
        m[i].s = a;
        m[i].e = b;
    }
                        //建立相容矩阵
    int** g = new int*[k+1];
    for (int i = 0; i < k+1; i++)
        g[i] = new int[k+1];

    for (int i = 1; i <= k; i++)
    {                   //上三角矩阵，不相容为1，相容为0
        for (int j = 1; j <= i; j++)
            g[i][j] = 0;
        for (int j = i + 1; j <= k; j++)
            {
                if (m[i].e >= m[j].s)
                    g[i][j] = 1;
                else
                    g[i][j] = 0;
            }
    }
```

O(n) O(n)

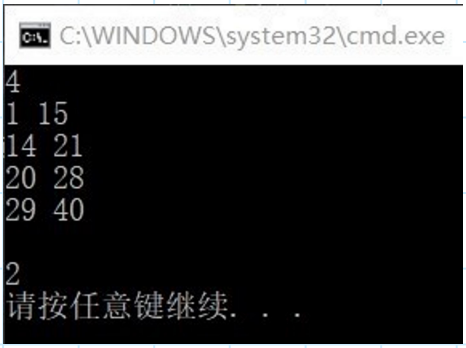
```
                    g[i][j] = 0;
                }
            }

            int* x = new int[k + 1];        //染色数组
            for (int i = 1; i <= k; i++)
                x[i] = 1;       //同染一种颜色
            for (int i = 1; i <= k; i++)
            {
                int n = 0;      //每次寻找不相容活动，从第一种颜色开始染
                for (int j = 1; j <= k; j++)
                {
                    if (g[i][j] == 1)
                    {
                        x[j] += n++;
                        if (x[i] == x[j])   //染到相同的颜色时增加一种颜色
                            x[j]++;
                    }
                }
            }

            int max = 0;
            for (int i = 1; i <= k; i++)
                if (x[i] > max)         //寻找颜色最大数
                    max = x[i];
            cout << "\n" << max << endl;

        }
```

O(n)

O(n)

O(n²)

运行结果如下所示



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。购物时可以使用的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*

coinschange (全局范围)

```
// coinschange.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//

#include <iostream>
#define MAXSIZE 100
#define MIN(a,b)    ( (a) <(b) ? (a) : (b) )
using namespace std;

bool enough(int c[],int coins[],int need)
{                                                   //判断所有的钱总和是否能付款
    double sum = 0;
    int count = 0;
    for (int i = 0; i < 6; i++)
        sum += c[i] * coins[i];
```

coinschange.cpp*

coinschange (全局范围)

```cpp
// coinschange.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//

#include <iostream>
#define MAXSIZE 100
#define MIN(a,b)    ( (a) <(b) ? (a) : (b) )
using namespace std;

bool enough(int c[],int coins[],int need)
{                                           //判断所有的钱总和是否能付款
    double sum = 0;
    int count = 0;
    for (int i = 0; i < 6; i++)
        sum += c[i] * coins[i];
    if (sum < need)
    {
        cout << "not enough money" << endl;
        return false;
    }

    return true;
}

int pay(int c[],int coins[],int need)
{                                           //顾客凑钱，没有补款
    int i = 5;                              //贪心，从最大的钱开始凑
    int n = 0;
    while (need)
    {                                       //凑够钱后退出
        while (c[i] > need) i--;
        if (coins[i])
        {                   //如果硬币足够，需付款减少，该硬币额减一，使用币数加一
            need -= c[i];
            coins[i]--;
            n++;
        }
        else
        {
            i--;
        }
    }
    return n;
}

bool complete(int c[],int change)
{                                       //判断补得钱是否是现有的币值，是返回true
    int i = 0;
    while (c[i] < change)   i++;
    if (c[i] == change) return true;
    else return false;
}

int change(int c[], int coins[], int need)
{                                       //商家找钱的方法
    int i = 0, j = 0;
    int change = 0;
    int n = 0;
    int shop[6] = { MAXSIZE,MAXSIZE,MAXSIZE,MAXSIZE,MAXSIZE,MAXSIZE };
    while ( c[i] < need )   i++;        //出比需付款大的币值
    while (!coins[i])   i++;            //同时该币值要有硬币
    if (i > 6)
        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          {                                   //凑钱直到商家能一次性补
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          }
82          n += pay(c, shop, change);
83
84          return n;
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