算法设计与分析第五次作业

2019202243 郭炜

题目描述：

求解会议安排问题

陈老师是一个比赛队的主教练，有一天他想给团队成员开会，应该为这次会议安排教室，但教室非常缺乏，所以教室管理员必须接受订单和拒绝订单以优化教室的利用率。如果接受一个订单，该订单的开始时间和结東时间成为一个活动。注意，每个时间段只能安排一个订单（即假设只有一个教室）。请找出一个最大化的总活动时间的方法。你的任务是这样的：读入订单，计算所有活动（接受的订单）占用时间的最大值。

输入描述：标准的输入将包含多个测试用例。对于每个测试用例，第1行是一个整数

n（n≤10000），接着的n行中每一行包括两个整数p和k（1≤p≤k≤300 000），其中p是

一个订单的开始时间，k是结束时间。

输出描述：对于每个测试用例，输出所有活动占用时间的最大值。

输入样例：

4

1 2

3 5

1 4

4 5

样例输出：

4

Code：

Pro.h

#pragma once

class pro

{

public:

    pro(int, int);

    ~pro();

    int getStart() const;

    int getEnd() const;

    int getTime() const;

private:

    int start;

    int end;

};

Pro.cpp

#include <vector>

#include "Pro.h"

pro::pro(int *start*, int *end*)

{

    this->start = start;

    this->end = end;

}

pro::~pro()

{

}

int pro::getStart() const

{

    return this->start;

}

int pro::getEnd() const

{

    return this->end;

}

int pro::getTime() const

{

    return this->getEnd() - this->getStart();

}

Source.cpp

#include <iostream>

#include <vector>

#include <algorithm>

#include "Pro.h"

using namespace std;

int count = 0;

// 按照结束时间降序，若结束时间相同，按照开始时间降序

bool mySort(const pro *p1*, const pro *p2*)

{

    if (p1.getEnd() != p2.getEnd())

    {

        return p1.getEnd() > p2.getEnd();

    }

    else

    {

        return p1.getStart() > p2.getStart();

    }

}

// 非回溯法

void func(const vector<pro>& *pros*)

{

    if (pros.size() <= 0)

    {

        return;

    }

    int start = pros[0].getEnd();

    for (auto p : pros)

    {

        if (p.getEnd() <= start)

        {

            ::count += p.getEnd() - p.getStart();

            start = p.getStart();

        }

    }

}

void test1()

{

*/\**

*4*

*1 2*

*3 5*

*1 4*

*4 5*

*\*/*

    vector<pro> pros;

    size\_t n;

    int s, e;

    cin >> n;

    for (size\_t i = 0; i < n; i++)

    {

        cin >> s >> e;

        pros.push\_back(pro(s, e));

    }

    cout << "--- sort pros ---" << endl;

    sort(pros.begin(), pros.end(), mySort);

    for (auto p : pros)

    {

        cout << p.getStart() << " " << p.getEnd() << endl;

    }

    func(pros);

    cout << "--- res ---" << endl;

    cout << ::count << endl;

}

// 回溯法

int temp = 0, ans = 0;

bool flag = false;

void dfs(vector<pro>& *pros*, int *i*)

{

// 标志位 表示是否找到一条完整路径

    ::flag = false;

    for (size\_t j = 1; j < pros.size(); j++)

    {

        if (pros.at(j).getStart() >= pros.at(i).getEnd())

        {

            int u = pros.at(j).getTime();

            temp += u;

            dfs(pros, j);

            temp -= u;

            ::flag = true;

        }

    }

*//*判断是否遍历结束

    if (::flag == false)

    {

        if (temp > ans)

        {

            ans = temp;

        }

    }

}

void test2()

{

*/\**

*4*

*1 2*

*3 5*

*1 4*

*4 5*

*\*/*

    vector<pro> pros;

    size\_t n;

    int s, e;

    cin >> n;

    pros.push\_back(pro(0, 0));

    for (size\_t i = 1; i <= n; i++)

    {

        cin >> s >> e;

        pros.push\_back(pro(s, e));

    }

    dfs(pros, 0);

    cout << "--- res ---" << endl;

    cout << ::ans << endl;

}

int main()

{

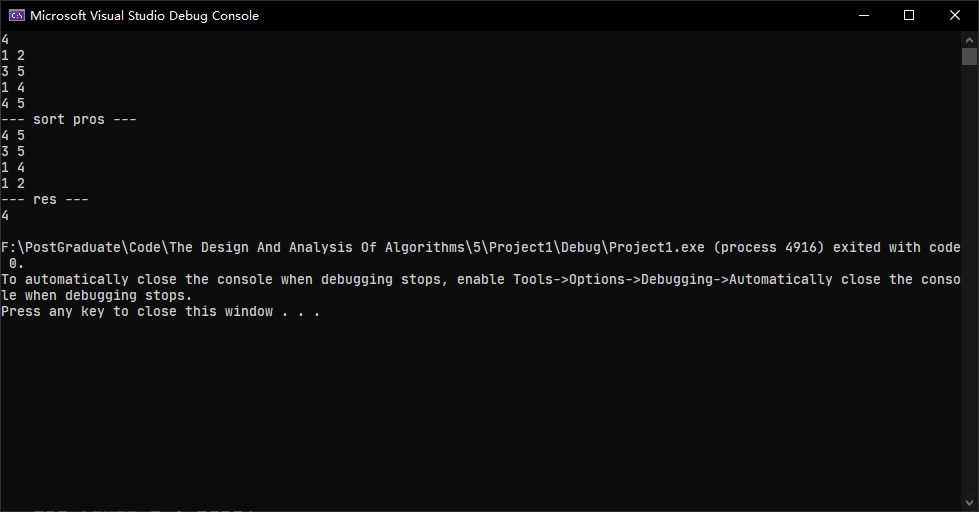
    test2();

    return 0;

}

Result:

非回溯法结果



回溯法结果

