**北京邮电大学软件学院**

**2020-2021学年第一学期实验报告**

**课程名称：** 算法分析与设计

**项目名称： 实验五 动态规划法**

**项目完成人：**

**姓名：\_\_王衔飞\_\_学号：\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_\_\_\_学号：\_\_\_\_\_\_\_\_**

**指导教师：**\_\_\_\_\_\_\_\_\_\_\_李朝晖\_\_\_\_\_\_\_\_\_\_\_

**日 期： 2020年 11 月 15 日**

1. **实验目的**

1、 深刻理解并掌握动态规划法的设计思想；

2、 提高应用动态规划法设计算法的技能； 运用分治法编程解决基本题，学有余力者继续解决选做题。

1. **实验环境**

Xcode 12.2 + macOS 11.0.1 + gcc9

语言：C++ 17

1. **实验结果**

基本题 1：0—1 背包问题（动态规划法） 给定 n 种物品和一背包。物品 i 的重量是 wi，其价值为 vi，背包的容量 为 C。问应如何选择装入背包的物品，使得装入背包中物品的总价值最大?

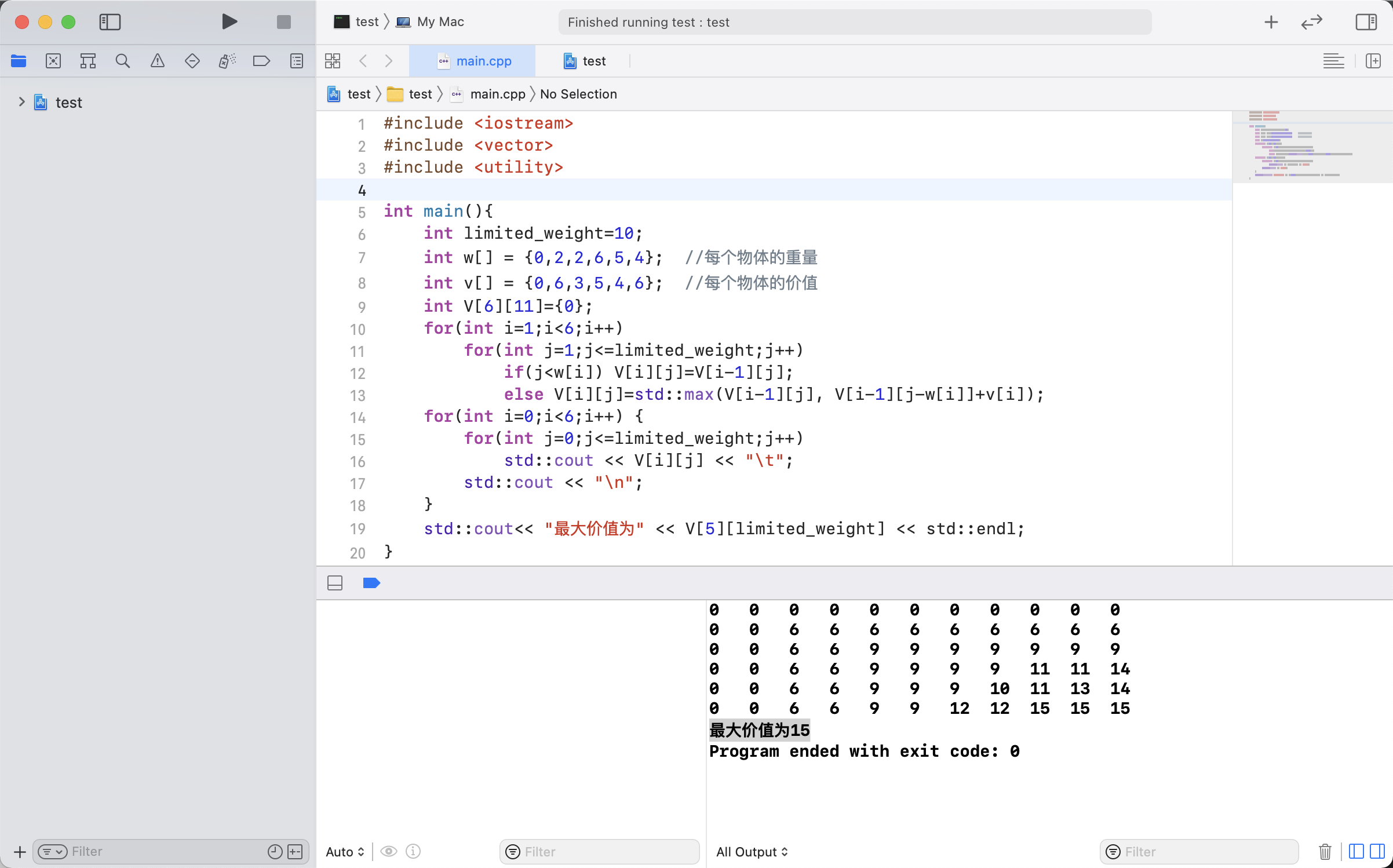
测试数据：

**int** limited\_weight=10;

**int** w[] = {0,2,2,6,5,4}; //每个物体的重量

**int** v[] = {0,6,3,5,4,6}; //每个物体的价值

输出：最大价值为15



基本题 2：带权的区间调度问题（动态规划法） 存在单一资源 R，有 n 个需求{1, 2, …, n}，每个需求指定一个开始时间 b i 与一个结束时间 ei ，在时间区间[bi , ei ]内该需求想要占用资源 R，资源 R 一 旦被占用则无法被其他需求利用。每个需求 i 带有一个权值 vi ，代表该需求 被满足之后能够带来的价值或者贡献。如果两个需求的时间区间不重叠，那 么它们是相容的。带权值的区间调度问题即，对上面所描述的情境，求出一 组相容子集 S 使得 S 中的区间权值之和最大。

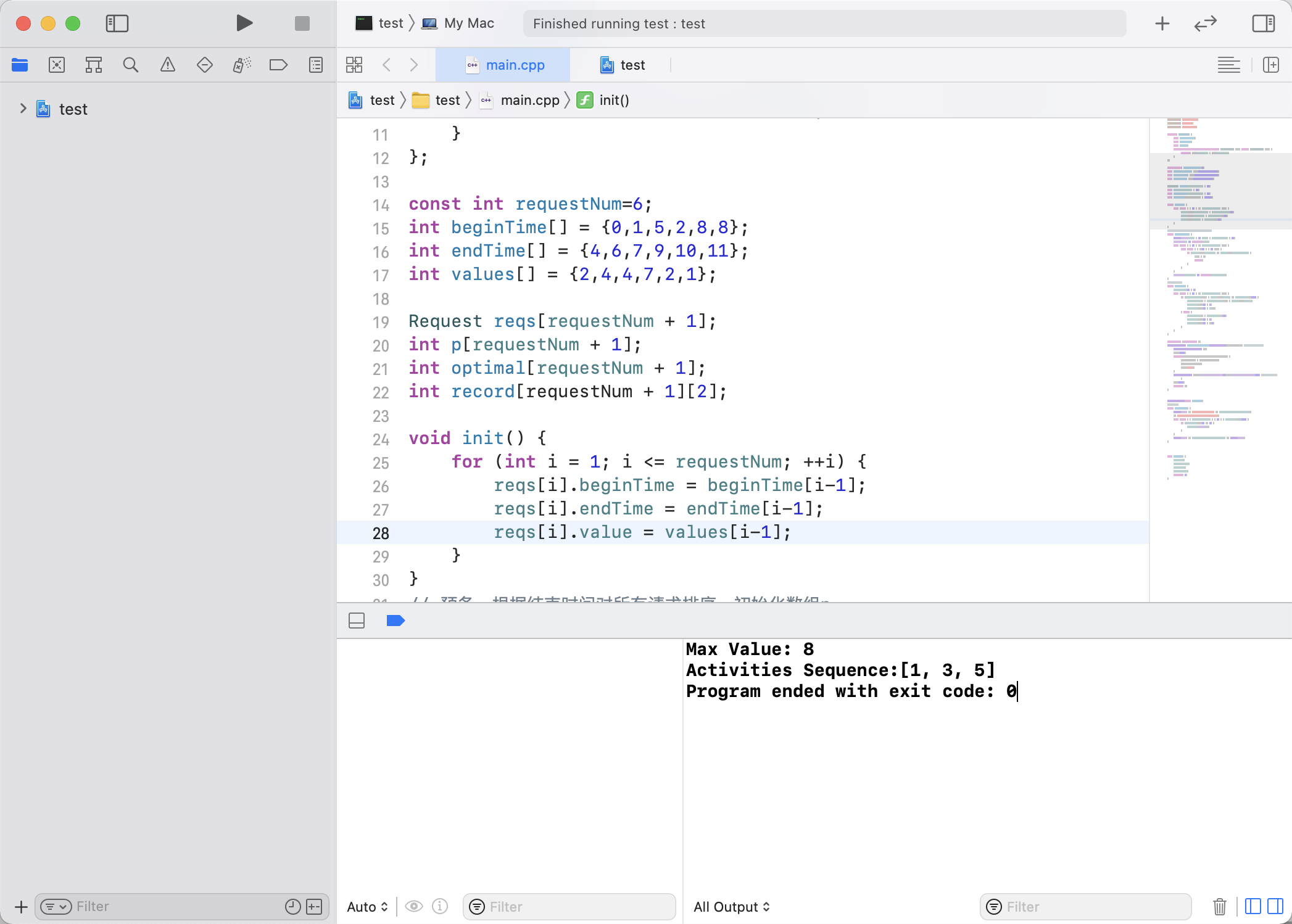
测试数据：

**int** beginTime[] = {0,1,5,2,8,8};

**int** endTime[] = {4,6,7,9,10,11};

**int** values[] = {2,4,4,7,2,1};

输出结果：Max Value: 8 Activities Sequence:[1, 3, 5]



1. **附录**

（附上实验文档，尽量有：问题分析、设计方案、算法原理、流程、设计图、程序、仿真结果、运行结果、心得等，具体内容根据实验要求来定。源代码请附在这里。源代码排版请特别注意，用5号字体，行间距为单倍行距。注意节省空间，不要浪费纸张。）

**0—1 背包问题（动态规划法）：**

根据动态规划函数，用一个(n+1)×(C+1)的二维表 V，V[i][j]表示把前 i 个物品装入容量为 j 的背包中获得的最大价值。按下述方法来划分阶段： 第一阶段，只装入前 1 个物品，确定在各种情况下的背包能够得到的最大价值； 第二阶段，只装入前 2 个物品，确定在各种情况下的背包能够得 到的最大价值；依此类推…直到第 n 个阶段。最后，V(n,C)便是在容量为 C 的背包中装入 n 个物品时取得的最大价值。

代码如下：

#include <iostream>

#include <vector>

#include <utility>

**int** main(){

**int** limited\_weight=10;

**int** w[] = {0,2,2,6,5,4}; //每个物体的重量

**int** v[] = {0,6,3,5,4,6}; //每个物体的价值

**int** V[6][11]={0};

**for**(**int** i=1;i<6;i++)

**for**(**int** j=1;j<=limited\_weight;j++)

**if**(j<w[i]) V[i][j]=V[i-1][j];

**else** V[i][j]=std::max(V[i-1][j], V[i-1][j-w[i]]+v[i]);

**for**(**int** i=0;i<6;i++) {

**for**(**int** j=0;j<=limited\_weight;j++)

std::cout << V[i][j] << "\t";

std::cout << "\n";

}

std::cout<< "最大价值为" << V[5][limited\_weight] << std::endl;

}

**带权的区间调度问题（动态规划法）：**

设计一个类，属性包括开始时间和结束时间和价值，先根据结束时间对所有请求排序，初始化数组p。我们的目的是求取一个最优的活动组合使得奖励最大化，因此，只需要考虑上面两种选择哪一种的奖励更多即可，奖励越多的选择越优秀。于是进行求解并输出。

代码如下：

#include <iostream>

#include <stack>

#include <sstream>

**struct** Request {

**int** beginTime;

**int** endTime;

**int** value;

**friend** **bool** **operator**<(**const** Request& r1, **const** Request& r2) {

**return** r1.endTime < r2.endTime;

}

};

**const** **int** requestNum=6;

**int** beginTime[] = {0,1,5,2,8,8};

**int** endTime[] = {4,6,7,9,10,11};

**int** values[] = {2,4,4,7,2,1};

Request reqs[requestNum + 1];

**int** p[requestNum + 1];

**int** optimal[requestNum + 1];

**int** record[requestNum + 1][2];

**void** init() {

**for** (**int** i = 1; i <= requestNum; ++i) {

reqs[i].beginTime = beginTime[i-1];

reqs[i].endTime = endTime[i-1];

reqs[i].value = values[i-1];

}

}

// 预备，根据结束时间对所有请求排序，初始化数组p

**void** prepare() {

std::sort(reqs + 1, reqs + requestNum + 1);

memset(p, 0, **sizeof**(p));

**for** (**int** i = 1; i <= requestNum; ++i) {

**for** (**int** j = i-1; j > 0; --j) {

**if** (reqs[j].endTime <= reqs[i].beginTime) {

p[i] = j;

**break**;

}

}

}

memset(record, 0, **sizeof**(record));

}

// 动态规划算法

**void** solve() {

optimal[0] = 0;

**for** (**int** i = 1; i <= requestNum; ++i) {

**if** (optimal[p[i]] + reqs[i].value >= optimal[i-1]) {

optimal[i] = optimal[p[i]] + reqs[i].value;

record[i][0] = 1;

record[i][1] = p[i];

} **else** {

optimal[i] = optimal[i-1];

record[i][0] = 0;

record[i][1] = i-1;

}

}

}

**template** <**typename** T>

std::string stack\_to\_str(std::stack<T> stack){ //输出栈 转换为字符串

std::stringstream ss;

ss<<'[';

**for**(;!stack.empty();stack.pop()) {

T element = stack.top();

ss<< element;

ss<<", ";

}

std::string s=ss.str().substr(0,ss.str().length()-2); //删除最后一个", "

s+=']';

**return** s;

}

std::stack<**int**> result;

// 输出结果

**void** output() {

std::cout << "Max Value: " << optimal[requestNum]

<< "\nActivities Sequence:";

**for** (**int** i = requestNum; i > 0; i = record[i][1]) {

**if** (record[i][0] == 1) {

result.push(i);

}

}

std::cout << stack\_to\_str(result) << std::endl;

}

**int** main() {

init();

prepare();

solve();

output();

**return** 0;

}

**心得：通过本次实验更进一步了解了动态规划算法**