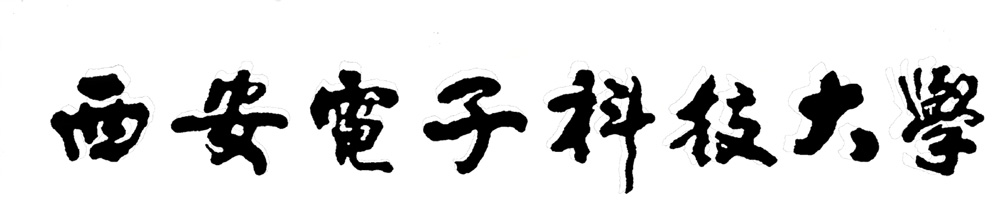
****

**算法分析与设计（本科）**

**上机报告**



学 院： 软件学院

专 业： 软件工程

**方 向**： **云计算方向**

姓 名： 孙 晖

学 号： 15130120141

目录

[一、上次实验错题 3](#_Toc513404401)

[二、实验内容 3](#_Toc513404402)

[三、实验过程 3](#_Toc513404403)

[3.1 实验一 3](#_Toc513404404)

[3.1.1实验内容 3](#_Toc513404405)

[3.1.2 实验过程 4](#_Toc513404406)

[3.1.3 实验结果 4](#_Toc513404407)

[3.1.4 实验小结 5](#_Toc513404408)

[3.2 实验二 5](#_Toc513404409)

[3.2.1实验内容 5](#_Toc513404410)

[3.2.2 实验过程 5](#_Toc513404411)

[3.2.3 实验结果 5](#_Toc513404412)

[3.2.4 实验小结 5](#_Toc513404413)

[3.3 实验三 6](#_Toc513404414)

[3.3.1实验内容 6](#_Toc513404415)

[3.3.2 实验过程 6](#_Toc513404416)

[3.3.3 实验结果 6](#_Toc513404417)

[3.3.4 实验小结 7](#_Toc513404418)

[3.4 实验四 7](#_Toc513404419)

[3.4.1实验内容 7](#_Toc513404420)

[3.4.2 实验过程 7](#_Toc513404421)

[3.4.3 实验结果 7](#_Toc513404422)

[3.4.4 实验小结 8](#_Toc513404423)

[四、本次实验小结 9](#_Toc513404424)

# 一、上次实验错题

首先，我想对上一次作业，也就是实验二的最后一题做出一些总结和更改。

多段图的动态规划问题，不是简单的想象那样子问题最短，而是要合理的分析怎样使得子问题最短就使得整体路径最短。

例如，我从初始节点到目标节点分析子问题，和从目标节点一点点往前推的最终结果是不同的。这说明，要想n段路径最短，首先需要n-1段路径最短，这样迭代下去，而不是n段最短从第一段开始就最短，这样得出的结果并不是正确的答案，逻辑上很好分析，只有前者得出的公式才是可以证明正确的，后者得出的方法只能被证明是错误的。下面两个博客具体讲解了这些问题。

<https://blog.csdn.net/u012432778/article/details/41623961>

<https://blog.csdn.net/u014359097/article/details/49852475>

# 二、实验内容

本次实验主要是关于贪心算法，一共有四道题。

题目一：著名的背包问题，一共有五个商品，有价值和重量，背包最多能装100磅，分别用分数背包和0/1背包来解决这个问题。

题目二：一个简单的调度问题，给定j1,j2,…jn这些工作。运行时间分别为t1,t2…tn。有一个简单的处理器，为了使平均完成时间最小，最好的调度方法是什么？假设这是一个不可抢占的调度：一旦一个工作开始了，就会一直运行到这个工作完成才结束。

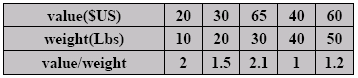
题目三：单源最短问题，给定了一个邻接矩阵，结点A是源。

题目四：所有结点对之间的最短路问题。

# 三、实验过程

## 3.1 实验一

### 3.1.1实验内容

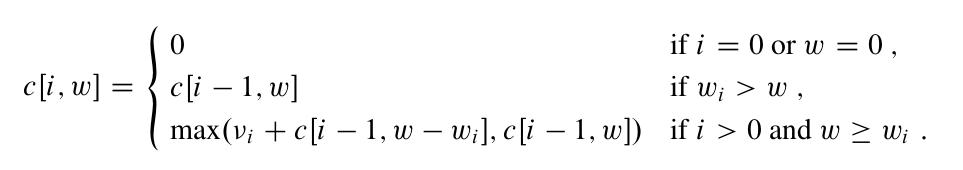
著名的背包问题，一共有五个商品，有价值和重量，背包最多能装100磅，分别用分数背包和0/1背包来解决这个问题。

### 3.1.2 实验过程

真正难的地方是0/1背包问题，0/1背包问题需要给出公式，而分数背包只需要装value/weight比值最高的就可以了，先写分数背包，然后再在网上看一看别人的代码是否通用性更高，学习一下。这里需要注意的是，0/1背包是DP问题，分数背包是贪心算法。

首先，分数背包问题。这里我考虑用结构体数组来存储整个表格，然后用冒泡排序把value/weight排序，同时将整个结构体数组按照value/weight的从大到小排序而排序。最终给出背包能够装的最大value。分数背包问题用的是结构体数组，这样究竟好不好呢？是个值得思考的问题，对于代码这种没有规范的东西，但是有很多东西一眼就可以看出优劣的。

0/1背包问题。要么全拿，要么全部不拿。看网上的解决办法都是一个二维数组，但是网上根本没有很好的解释思路，尤其是关于怎样使装法最佳，以及如何把这个最佳的装法和代码结合在一起。这里解释一个情景，当第一个物品被装进去时，你如何确定这个东西就是最佳的呢？并不能说明，但是把这问题反过来，我只看子问题，假设子问题是最佳的，那么我接下来要做的就是，看装进去哪一个最佳，如果装不进去，就意味着要把背包里已经装进去的东西腾出来，所以比较的就是腾出来装进去和其本身哪个价值最大。能装进去自然就不考虑这个问题。



### 3.1.3 实验结果

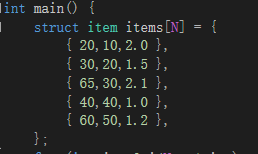
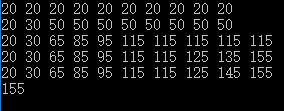


图1 分数背包初始化



图2 分数背包输出结果

图3 0/1背包输出结果

### 3.1.4 实验小结

算法是那种难者不会，会者不难的东西，没有用到编程语言中太难的地方，都是编程语言最简单的知识就可以做出来的题，而且老师还给的有伪代码，我认为只要把思路搞懂，写出代码都是时间问题，即使有地方报错，改一改就行了，主要的还是思路和伪代码看懂。

## 3.2 实验二

### 3.2.1实验内容

使用非抢占式调度的情况下求出平均完成时间最短的调度顺序。

### 3.2.2 实验过程

非抢占式调度，因此短作业优先。使用堆排序，建一个小顶堆，每次拿出小顶堆上的最小元素，算出各完成时间的总和时间sum，sum除以作业数就可以得出最短的平均完成时间。

### 3.2.3 实验结果

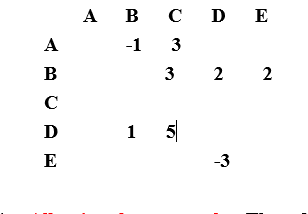
图1 如图所示为调度顺序和平均完成时间

### 3.2.4 实验小结

原本想着从小到大排序，但是那样就太简单了，显然老师要求的不是这样的，因此我做了这样一个通用性稍微强一些的非抢占式调度。

## 3.3 实验三

### 3.3.1实验内容

使用Bellman-Ford解决单源最短路径问题，其中A是唯一的源节点，计算A可以到达的所有结点之间的最短路径权重。

### 3.3.2 实验过程

Bellman-Ford算法是上个学期的课程中学的算法，但是具体是哪门课程的，我忘记了。主要说明一下Bellman-Ford算法。

Bellman-Ford是可以处理负边权的，而Dijkstra算法是不行的。适用于单源最短路径，适用于有向图或者无向图，无向图看做有向图的两个方向都可以通过。

步骤：

1. 首先将除了源点以外的所有最短路径估值置为无穷大，源点置为0；
2. 迭代求解：反复对边集中的每条边进行松弛操作，使得顶点集V中的每个顶点V的最短路径估计值逐步逼近其最短距离；
3. 检验负权回路，判断边集E中的每一条边的两个端点是否收敛。如果存在未收敛的顶点，返回false，否则返回true。

一开始上来看到要存储权值第一反应就应该是用结构体，然后结构体数组存储点。

### 3.3.3 实验结果

图1 最短路径如图所示

### 3.3.4 实验小结

Bellman-ford算法在写的过程中我发现main函数里的getchar();并不能为我的输出提供一个暂停的作用，一般来说我喜欢用getchar来暂停截图用，但是这一次却没有用了，只能用stdlib中的system pause。希望自己记住并找出原因。

## 3.4 实验四

### 3.4.1实验内容

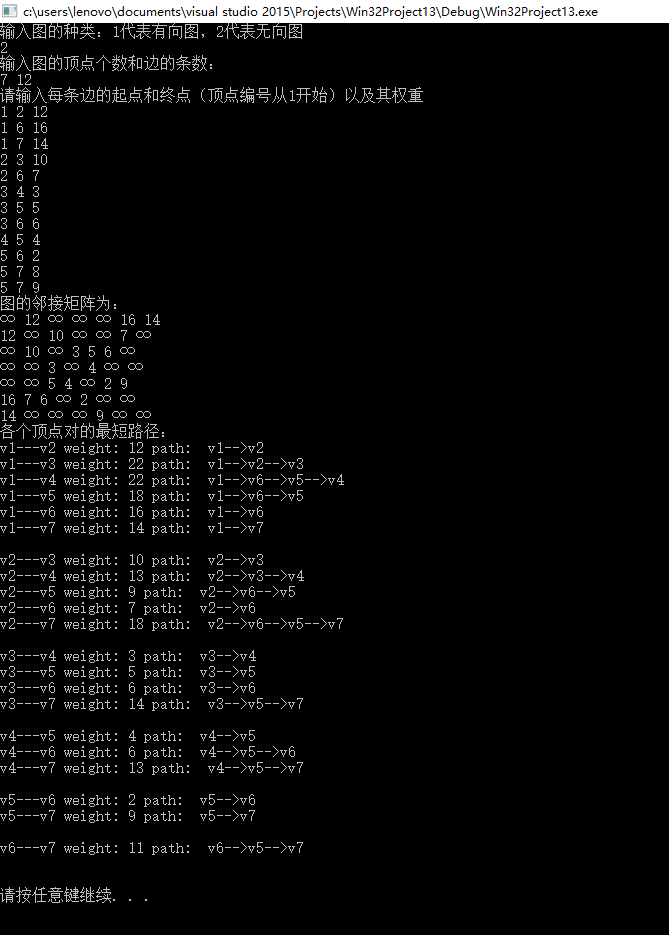
实验四是和实验三相关联的，使用Floyd算法计算实验三的图中任意两节点之间的最短路径，是一个动态规划问题。可以处理有向图（可以负权）的最短路径问题。

### 3.4.2 实验过程

该算法建立一个矩阵将对角线值设为0，其余值为无穷大，也就是对角线矩阵。看网上一般都是将权值以矩阵形式存储的。核心算法是三个for循环。但是这三个for循环理解起来挺难受的，而且算法时间复杂度应该是n^3，可以说是非常大了。在学习算法的过程中看到了网上的好代码，因此将其打包下来运行一下看一看。

### 3.4.3 实验结果

图1 矩阵如图所示

图2 别人的代码

### 3.4.4 实验小结

感觉别人的代码总是很美，自己的代码总是很烂，一开始想着吧矩阵中的无穷大设置为MAX显示在矩阵中，结果做不到。再看别人的代码，挫败感更大。

# 四、本次实验小结

1. 对于很多优化问题而言，DP就够了，这个情况是指一个问题可以分解为多个子问题时是可行的。贪心算法总是想使每一步看起来是最优的。也就是说，实现局部最优以期能够达到全局最优。有时这样成功，有时这样失败。
2. 看代码发现，有些时候，单纯的在一个函数里定义变量远远没有#define 一个常数要方便，因为#define这样一个常数意味着这个代码通用性更高，比如，背包问题，有的情况，有五个东西待装入，有的时候由六个东西待装入，相比于在函数里把函数值一个个改过来，显然在#define后面直接改更加方便快捷，而且更加实用。
3. 今天看了一下阿里的算法工程师面试题发现，认识到学好java的重要性，以后能用java就用java，然后要把C++忘记的重新捡回来，实际上，感觉C++已经忘完了。