算法导论 课程设计

成绩	

题　　目：贪心算法

学院班级：1613012

学　　号：16130120191

姓　　名：罗阳豪

主讲教师：覃桂敏

日　　期：2019 年 05 月 02 日

目录

题目一　背包问题

1.1 实验题目

背包问题。5 个物品的价值和重量列表如下，背包最多可以装下 100 磅。请解决，部分背包问题和 0-1 背包问题。

	1	2	3	4	5
value (US $)	20	30	65	40	60
weight (Lbs)	10	20	30	40	50
value / weight	2	1.5	2.1	1	1.2

1.2 实验目的

认识贪心算法，了解构思贪心算法的一般步骤，通过经典的“背包问题”，加深对于贪心算法的理解。了解背包问题的一般解法。

1.3 实验设计与分析

对于 0-1 背包问题，每个物品只能选择完整放入背包中或者不放入背包中，不能部分放入背包。

我觉得贪心算法算得上是动态规划算法在某些特殊情况下的简化。同样首先需要划分形式与原问题差不多的子问题。在这里，如果我们考虑背包 100 磅时，物品 1-5 的背包问题其实就是两个子问题，如果装上第 5 个物品，那么背包剩余 50，只需要考虑物品 1-4，或者不装上第 5 个物品，背包剩余 100，考虑物品 1-4.

这样的逻辑可以写成如下递归式

f(c, items(1, i)) = max(f(c-item(i).weight, items(1, i-1)) + item(i).value, f(c, items(1, i-1)))

所以类似动态规划的自底向上方法，首先考虑背包为 0，只有物品 1，然后逐渐增加需要考虑的物品。然后逐渐增加背包容量，最后得到原问题背包 100，物品 1-5 的背包问题的解。

对于部分背包问题，由于物品可以部分地被装入背包，所以只要从“性价比”从高到低装物品，直到背包装满。一个简单的方法就是对物品按价值重量比进行排序，然后从大到小装入背包。但是这样有一个比较明显的问题，那就是那些排在后面，不会被装入背包的物品的顺序没有意义，而得到这些顺序浪费了宝贵的时间。所以实际上我们只需要部分排序，使用堆排序是不错的选择，每次从大顶堆顶获取最大元素，然后维

持堆性质，然后继续获取，直到背包装满，后面的元素就不需要继续排序了。

1.4 实验环境

Python 3.7.0

1.5 项目测试

将上述测试用例作为输入，得到如下结果

0-1 背包问题：

(155, [1, 2, 4])

第一个数是总价值，后面列表表示装入的物品序号

部分背包问题：

(163.0, [(2, 30), (0, 10), (1, 20), (4, 40)])

第一个数是总价值，后面列表是装入的物品的序号和重量

题目二 调度问题

2.1 实验题目

一个简单的调度问题，我们给出作业 j1， j2， ...， jn，他们分别具有已知的运行时间 t1，t2，...，tn。我们有单个处理器，为了最小的平均完成时间，调度这些作业的最好方法是什么，假设这是一个非抢占式的调度：一旦启动一个作业，他必须完成。以下是一个用例。

1. （j1， j2， j3， j4）:（15， 8， 3， 10）

2.2 实验目的

通过操作系统方面最经典的"进程调度问题"，加深对于贪心算法的理解。理解贪心的思想，了解在某些特殊情况下贪心算法的实现。

2.3 实验设计与分析

对于对操作系统有基本认识的人来说，知道如下结论并不困难：对于非抢占式的调度，要得到最短平均完成时间，应该使用"最短作业优先"（SJF）算法。

所以对于这个题目中简单的例子来说，我们只要对上述 4 个作业排序，然后由小到大执行，统计平均完成时间即可。

但是实际情况没有那么简单。真实的操作系统一般不会提前知道自己将要调度多少个作业，而且在作业运行的同时，待执行作业可能会发生变化，比如说某作业被撤销，某作业预期执行时间发生改变，有新的作业被提交。在每次发生类似变更时都重新排序整个待执行作业列表并不切实际。一个合理的方法是使用最小优先队列，因为这样一个队列可以以较小的开销适应队列的各种变更，并且始终能从队列首部找到下一个应该执行的作业。

2.4 实验环境

Python 3.7.0

2.5 项目测试

对于上述测试用例，得到如下调度结果

（[[2， 3， 3]， [1， 8， 11]， [3， 10， 21]， [0， 15， 36]]， 17.75）

平均完成时间是 17.75

调度顺序是<j3， j2， j4， j1>

题目三 单源最短路

3.1 实验题目

求解单源最短路问题，以下是邻接矩阵，以顶点 A 为源

	A	B	C	D	E
A		-1	3		
B			3	2	2
C					
D		1	5		
E				-3	

3.2 实验目的

通过单元最短路问题，了解图论的基础，了解最短路问题的一般解决方法。了解贪心、动态规划如何应用于最短路问题。

3.3 实验设计与分析

由于该图中含有负权值的边，不能使用经典的笛杰斯特拉算法。由于是单源最短路问题，所以考虑使用 Bellman Ford 算法

3.4 实验环境

Python 3.7.0

3.5 项目测试

将上述测试用例作为输入得到如下结果

(0, None)
(-1, 0)
(2, 1)
(-2, 4)
(1, 1)

表示以 A 为源：

到 A 开销为 0，前驱节点无

到 B 开销为-1，前驱节点为 A

到 C 开销为 2，前驱节点为 B

到 D 开销为-2，前驱节点为 E

到 E 开销为 1，前驱节点为 B

题目四 所有节点对最短路

4.1 实验题目

所有及诶但对最短路径。邻接矩阵同第三题。(请使用 Floyd 或 Johnsons 的算法)

4.2 实验目的

通过单元最短路问题，了解图论的基础，了解最短路问题的一般解决方法。了解贪心、动态规划如何应用于最短路问题。

了解经典的最短路算法，如 Floyd 或 Johnsons 的算法

4.3 实验设计与分析

由于本人觉得 Floyd 算法比较简单，所以对于该问题，我使用了 Floyd 算法。对于该算法的原理和实现细节，课本有非常详细的说明，便不在此做赘述。

4.4 实验环境

Python 3.7.0

4.5 项目测试

将测试用例输入，得到如下结果

[(0, None), (-1, 0), (2, 1), (-2, 4), (1, 1)]

[(inf, None), (0, None), (3, 1), (-1, 4), (2, 1)]

[(inf, None), (inf, None), (0, None), (inf, None), (inf, None)]

[(inf, None), (1, 3), (4, 1), (0, None), (3, 1)]

[(inf, None), (-2, 3), (1, 1), (-3, 4), (0, None)]

该结果表示两节点间最短路径的开销和前驱节点矩阵。