目录

一、实验题目

1、Knapsack Problem. There are 5 items that have a value and weight list below, the knapsack can contain at most 100 Lbs. Solve the problem both as fractional knapsack and 0/1 knapsack.

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

2、A simple scheduling problem. We are given jobs j1, j2… jn, all with known running times t1, t2… tn, respectively. We have a single processor. What is the best way to schedule these jobs in order to minimize the average completion time. Assume that it is a nonpreemptive scheduling: once a job is started, it must run to completion. The following is an instance.

3、Single-source shortest paths. The following is the adjacency matrix, vertex A is the source.

4、All-pairs shortest paths. The adjacency matrix is as same as that of problem 3.(Use Floyd or Johnson’s algorithm)

二、实验目的

掌握贪心算法的思想，学会使用动态规划和贪心算法解决问题

三、实验设计与分析

1、Knapsack Problem.

题目分为两个部分，第一部分是分数背包，可以将部分物品放入背包中，可

以使用贪心算法得到最优解，按照物品单位重量的价值进行排序，依次放入背包中。0/1 背包需要使用动态规划进行求解，其递推式表示为：

$$\begin{cases} KS(i,j) = KS(i-1,j) & (j<wi) \\ KS(i,j) = \max\{KS(i-1,j),\ ks(i-1,j-wi)+vi\} & (j>wi) \end{cases}$$

具体的算法实现，对于分数背包首先将每件商品按上述规则排序，然后按顺序放入背包，对于无法完整放入的物品部分放入；对于 0/1 背包，如果背包容量大于当前物品重量，则比较背包装前 i-1 个物品的价值与前 i-1 个物品加第 i 个物品的价值，取最大的填入 ks[i][j]；否则将 ks[i-1][j]填入 ks[i][j]，即背包装不下。

2、A simple scheduling problem

进程调度的平均最小时间为所有进程等待时间的平均数，如果时间长的进程优先运行则所有进程都需等待更长的时间才能运行，即平均等待时间增加。所以要将时间段的进程优先运行。

具体的算法实现为对运行时间进行升序排列，然后顺序执行

3、Single-source shortest paths.

问题是单元最短路径问题且存在负权值的边，可以使用 Bellman-Ford 算法，每次遍历所有的边进行松弛，然后检查是否存在负权值的边，共遍历 n-1 次。递推公式如下：

```
dis1[s]=edge[s][t];//直达边，即是直接连接s到t的那条边
disk[s]=min(disk-1[s],min(disk-1[j]+edge[j][s])//经过k-1个点后，最短路径
```

具体的算法实现，除了源结点，所有的最短路径初始化为无穷大；遍历各边，若到该边后驱节点的最短路径长度大于到该边前驱节点的最短路径长度加改变权重，则将最短长度更新为后者。若存在边的后驱节点的最短路径可以被重复更新，则存在负环退出

4、All-pairs shortest paths.

本题是各节点之间的最短路径，Floyd 算法是求解任意两点间最短路径的经典算法，与 Bellman-Ford 算法相比同样使用了松弛的思想，与第三题相比要将 dist 矩阵扩展为 2 维矩阵，增加一重循环对每个点进行遍历，递推式如下：

$$A(k)[i][j] = \min \{ A(k-1)[i][j],\ A(k-1)[i][k]+A(k-1)[k][j],\ k=0,\ 1,\ 2,\ \ldots,\ n-1 \}$$

具体的算法实现，对于每个节点进行遍历，将该节点 k 作为中间节点，例如计算 i->j 和 i->k->j 两者距离的最小值对矩阵进行更新。

四、实验环境

Windows 10、IntelliJ IDEA 2020.3.2 x64、jdk1.8.0_281

五、项目测试（功能和性能）

1、Knapsack Problem.

运行结果如下图

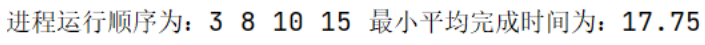
0/1 背包算法时间复杂度为 O(mn)，m 为背包的容量，n 为物品数量

分数背包问题不考虑排序所用时间，算法时间复杂度为 O(n)

结果正确

2、A simple scheduling problem

运行结果如下图



算法时间复杂度为 O(n) ，结果正确

3、Single-source shortest paths.

运行结果如下图

```
源结点到1的最短距离为：-1
源结点到2的最短距离为：2
源结点到3的最短距离为：-2
源结点到4的最短距离为：1
```

算法的时间复杂度为O(VE),结果正确

4、 All-pairs shortest paths.

运行结果如下图

```
0    -1   2    -2   1
NULL     0    3    -1   2
NULL     NULL      0    NULL      NULL
NULL     1    4    0    3
NULL     -2   1    -3   0
```

算法时间复杂度为O(n3) ,结果正确