算法第三次上机报告

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.

重量	价值	性价比
10	20	2
20	30	1.5
30	65	2.1
40	40	1
50	60	1.2

思路：

此题是经典的背包问题，0-1背包和分数背包，前者用dp即可，后者使用贪心更简单。对于 前者，贪心不一定获得最优解所以这里不适用贪心解0-1背包。

其实，**贪心算法**最大的特点，就是在每一步中取最优化的解，不会回溯处理。这样的策略，自然在执行速度上更快，但是因为这种方法的**短视**。会导致得的解并不是真正的全局最优解，但是**贪心算法**得到的依然是一个近似最优解。

0-1背包

对于0-1背包问题，运用dp的思想可以有两种常见状态转移：*认为物品下标从0开始*

- `dp[i][j]`表示从前i-1个物品选，在重量不超过j的情况下的最大价值。（正向）
- `dp[i][j]`表示从第i个物品开始选，在重量不超过j的情况下的最大价值。（反向）

状态转移方程分别如下：

```
#初始化dp[0][j]=0
if j<w[i]:
     dp[i+1][j]=dp[i][j]
else:
    dp[i+1][j]=max(dp[i][j],dp[i][j-w[i]]+v[i])
```

```
#初始化dp[n][j]=0
if j<w[i]:
    dp[i][j]=dp[i+1][j]
else:
    dp[i][j]=max(dp[i+1][j],dp[i+1][j-w[i]]+v[i])
```

这里，针对第一种**dp**思想，解此问题：

```
def knapsack(w,v,c):
    n=len(w)
    m=[[0]*(c+1) for i in range(n+1)]
    for k in range(1,n+1):
        for r in range(1,c+1):
            if w[k-1]>r:
                m[k][r]  = m[k-1][r]
            else:
                m[k][r]=max(m[k-1][r],m[k-1][r-w[k-1]]+v[k-1])
    return m[n][c]
```

分数背包

这一种就可以用贪心了，每次把性价比最高的放入口袋即可。思路比较简单：

```
#分数背包
def fractional_knapsack(w,v,c):
    item=[]
    for weight,value in zip(w,v):
        item.append({'weight':weight,'value':value,'ratio':value/weight})
    item.sort(key=lambda x:x['ratio'],reverse=True)#默认从小到大排序
    print(item)
    total=0
    for x in item:
        if c>x['weight']:
            c-=x['weight']
            total+=x['value']
        else:
            total+=(c/x['weight'])*x['ratio']
            break
    return total
```

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.
>
> (j1, j2, j3, j4) : (15，8，3，10)

思路：

这个问题就是操作系统学过的进程调度的一种，为了让平均周转时间最短，显然应该每次优先调度时间最短的进程。因此，只要做个排序即可。

```
def schedule(a):
    a=sorted(a)#非原地排序
    L=[sum(a[:i]) for i in range(1,len(a)+1)]
    return sum(L)/len(a)
```

前两个问题的实验结果截图如下：

```
D:\python3.5\python.exe E:/pycharm/algorithms/test_hw3.py
0-1背包的结果： 155
分数背包问题的结果： 115.96
调度问题的结果 17.75
```

Single-source shortest paths.

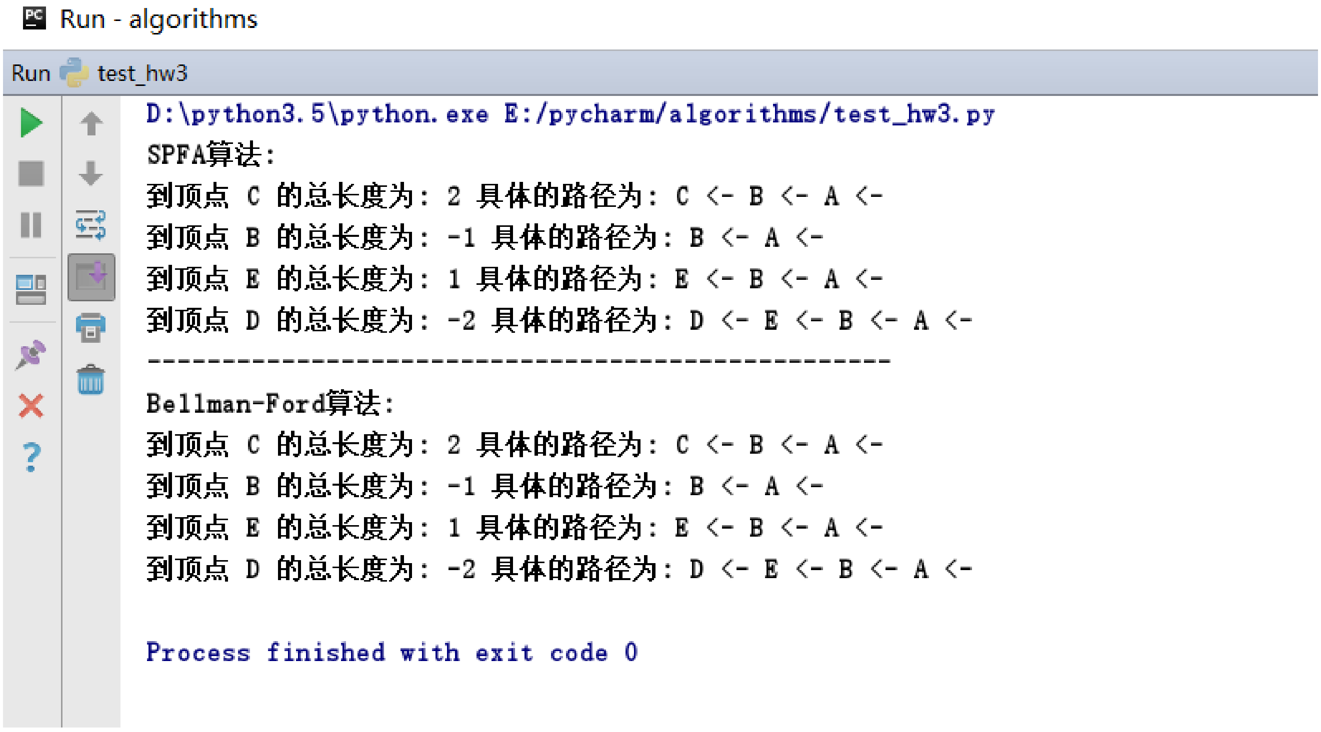
思路：

这里给的图有负权值，那么基于贪心的**Dijkstra**算法将不能使用，所以只能使用**Bellman-Ford**算法。即松弛所有的边，重复**v-1**轮。

注意:原始的**bellman**算法实际使用中常常用队列优化，即每次把松弛成功的顶点放入队列中，下次从队列中取出顶点直到队列为空或者发现负环。源码中两种方法我都实现了。

```
def Bellman_Ford(G,v):
    '''
    每一轮对所有顶点进行松弛，如果哪一轮松弛后最短距离都不变化就提前结束。否则，如果执行n
    轮之后还在变化则说明存在负环。最多n-1次松弛就可以得到答案。（n为顶点个数）
    :param G: 图的邻接表
    :param v: 起点
    :return: dis,path
    '''
    dis=dict((k,float('inf'))for k in G)
    dis[v]=0
    path={}
    for rounds in G:#一共执行n次，如果n次之后最短路径还没求出来说明有负环
        changed=False
        #对所有顶点进行松弛
        for u in G:
            for w in G[u]:
                if dis[w] > dis[u] + G[u][w]:
                    dis[w] = dis[u] + G[u][w]
                    path[w] = u
                    changed=True
        if not changed:
            break
    else:
        sys.exit('存在负环，算法求不出来')
    return dis,path
```

报告中我只把原始的算法放在这，spfa算法见附件源码。



All-pairs shortest paths.

任意节点对的最短路径问题，经典的算法是基于**DP**的**Folyd_Warshall**算法。所以，关键思路在于从**u**到**v**的最短路径经不经过**k**的问题。也因此，**k**是最外层循环。

```
def Folyd_Warshall(G):
    G2=deepcopy(G)
    #强行补成邻接矩阵的形式
    for u in G2:
        for v in G2:
            if u==v:
                G2[u][v]=0
            if G2[u].get(v)==None:
                G2[u][v]=float('inf')

    for k in G:
        for u in G:
            for v in G:
                if G2[u][v]>G2[u][k]+G2[k][v]:
                    G2[u][v]=G2[u][k]+G2[k][v]
    return G2
```

实验结果截图如下:

test_hw3

```
D:\python3.5\python.exe E:/pycharm/algorithms/test_hw3.py
Folyd-Warshall:
A {'E': 1, 'D': -2, 'C': 2, 'B': -1, 'A': 0}
D {'E': 3, 'D': 0, 'C': 4, 'B': 1, 'A': inf}
C {'E': inf, 'D': inf, 'C': 0, 'B': inf, 'A': inf}
B {'A': inf, 'D': -1, 'C': 3, 'B': 0, 'E': 2}
E {'E': 0, 'D': -3, 'C': 1, 'B': -2, 'A': inf}

Process finished with exit code 0
```