0-1 背包问题实验报告

学号：2111454 姓名：李潇逸

2023/4/17

1 实验源码

1.1 回溯算法解决 0-1 背包问题

```python
from typing import List
import time
class Node:
    def __init__(self,volume,weight) -> None:
        self.volume = volume
        self.weight = weight

def backpack(n : int,num : int,v : int,nodelist : List[Node]):#从n件物品中选，原有num个物品，背包可用空间为v，查找列表为nodelist
    if (n<=0) or (v<=0):#物品没有了，或背无可用空间，停止
        return 0
    if v<nodelist[num-n].volume:#如果背包可用空间比物品的体积小，则不放入
        ans=backpack(n-1,num,v,nodelist)
        return ans
    temp1=backpack(n-1,num,v-nodelist[num-n].volume,nodelist)+nodelist[num-n].weight#放入背包
    temp2=backpack(n-1,num,v,nodelist)

    #ans存放最大价值
    if temp1>=temp2:
        ans=temp1
    else:
        ans=temp2
    return ans

def main() :
    v_max , n = map(int,input().split()) #背包容量和物品总数
    node : List[Node] = []

    for _ in range(n):
        v , w = map(int,input().split())
        temp_node = Node(v,w)
        node.append(temp_node)

    ans = backpack(n,n,v_max,node)
    print(ans)

if __name__ == '__main__':
    start = time.time()
    main()
    end = time.time()
    print("run time is",end - start)
```

图 1.1: 回溯算法源码

1.2 动态规划算法解决 0-1 背包问题

2 实验结果

2.1 4 个物品

2.1.1 回溯算法

4 物品回溯算法计算结果为 37，用时约为 35.655ms。

2.1.2 动态规划算法

4 物品动态规划算法计算结果为 37，用时约为 2.952ms。

```python
from typing import List
import time

class Node:
    def __init__(self,volume,weight) -> None:
        self.volume = volume
        self.weight = weight

def backpack(n : int,v_max : int,nodelist : List[Node],ans : List[int]) -> None:
    for i in range(1,n+1):
        for j in range(1,v_max+1):
            if j < nodelist[i-1].volume:
                ans[i][j] = ans[i-1][j]
            else:
                ans[i][j] = max(ans[i-1][j],ans[i-1][j-nodelist[i-1].volume]+nodelist[i-1].weight)


def main() :
    v_max , n = map(int,input().split()) #背包容量和物品总数
    node : List[Node] = []
    ans : List[List[int]] = [[0 for _ in range(v_max+1)] for _ in range(n+1)]

    for _ in range(n):
        v , w = map(int,input().split())
        temp_node = Node(v,w)
        node.append(temp_node)

    backpack(n,v_max,node,ans)
    ans_num = max(max(ans))
    print(ans_num)


if __name__ == '__main__':
    start = time.time()
    main()
    end = time.time()
    print("run time is",end - start)
```

图 1.2: 动态规划算法源码

```
(base) lixiaoyi@lixiaoyideMacBook-Pro 算法设计作业 % python -u "/Users/lixiaoyi/Desktop/homework_code/算法设计作业/2111454_H7/
2111454_H7_Q1_force.py"
5 4
2 12
1 10
3 20
2 15
37
run time is 35.65533781051636
```

图 2.1: 4 物品回溯算法结果

2.2 25 个物品

2.2.1 回溯算法

25 物品回溯算法计算结果为 2268，用时约为 12.254ms。

2.2.2 动态规划算法

25 物品动态规划算法计算结果为 2268，用时约为 6.216ms。

3 结论

经过运行测试可以发现，回溯算法和动态规划算法运行所得的结果相同，且两次实验均为动态规划算法运行速度远远快于回溯算法运行速度，证明了解决此类问题时动态规划算法具有更好的效率。具体原因是：动态规划算法每一次运行后都主动记录了结果，因此不需进行大量重复运算；而回溯算法则不得不进行大量的重复运算。

```
(base) lixiaoyi@lixiaoyideMacBook-Pro 算法设计作业 % python -u "/Users/lixiaoyi/Desktop/homework_code/算法设计作业/2111454_H7/2
111454_H7_Q1.py"
5 4
2 12
1 10
3 20
2 15
37
run time is 2.952455997467041
```

图 2.2: 4 物品动态规划算法结果

```
(base) lixiaoyi@lixiaoyideMacBook-Pro 算法设计作业 % python -u "/Users/lixiaoyi/Desktop/homework_code/算法设计作业/2111454_H7/2111454_H7_Q1_force.
py"
300 25
22 132
40 111
8 125
28 153
38 148
38 139
40 152
35 117
19 194
39 164
7 92
10 160
10 96
24 108
30 54
37 115
16 115
16 151
27 116
25 146
5 182
25 145
34 96
7 157
32 112
2268
run time is 12.254281997680664
```

图 2.3: 25 物品回溯算法结果

```
(base) lixiaoyi@lixiaoyideMacBook-Pro 算法设计作业 % python -u "/Users/lixiaoyi/Desktop/homework_code/算法设计作业/2111454_H7/2111454_H7_Q1.py"
300 25
22 132
40 111
8 125
28 153
38 148
38 139
40 152
35 117
19 194
39 164
7 92
10 160
10 96
24 108
30 54
37 115
16 115
16 151
27 116
25 146
5 182
25 145
34 96
7 157
32 112
2268
run time is 6.216109037399292
```

图 2.4: 25 物品动态规划算法结果