深 圳 大 学 实 验 报 告

课程名称：算法分析与设计

实验项目名称：背包问题

学院：计算机与软件学院

专业：软件工程

指导教师：李荣华

报告人：洪继耀 学号：2014150120 班级：02

一、 实验目的

（1）掌握动态规划算法设计思想。

（2）掌握背包问题的动态规划解法。

二、 实验内容

1．算法综述以及 01 背包问题的分阶段。

在求解多阶段决策的动态过程的优化问题， 我们常用动态规划算法，它的核心是：

多阶段决策问题 ,求解的问题可以划分为一系列相互联系的阶段 ,在每个阶段都要做出决策，并且上一个阶段的决策的选择会影响下一个阶段的决策。我们分阶段地去求解 ,且记下每一个阶段的最优解，以后阶段的解都依赖前面阶段的解。

适用动态规划算法的问题的核心都是**分阶段**。

对于 0-1 背包问题，我们可以这样分阶段：

a） 第一阶段：考虑只放入 1 号物品，对于不同的背包承重(从 1 到 W)，求最大价值，记录在备忘录中；

b） 第二阶段：只考虑 1 号和 2 号物品，对于不同的背包承重，按照一定方法，求最大价值，记录在备忘录中；该问题求解在已知上个阶段最优解的基础之上进行，这个一定方法就是我们的状态转移方程（后面说）

c） 以此类推，直至第 n 阶段。

2. 核心算法-填表。

2.1 以下给出算法的核心代码之一——填表：

```
/**
 * 自底向上地填表
 */
private static void initMemo() {
    for (int i = 1; i <= N; i++) { // 一行一行填，穷举所有物品
        for (int j = 1; j <= W; j++) { // 对于前n个物品，遍历所有重量的情况
            if (j - itemsArray[i].weight >= 0) { // 背包放得下第i个物品，就有两种选择
                memo[i][j] = Math.max(
                        memo[i - 1][j], // 一种是不放
                        memo[i - 1][j - itemsArray[i].weight] + itemsArray[i].price // 一种是放
                );
            } else { // 放不下，只能不放了
                memo[i][j] = memo[i - 1][j];
            }
        }
    }
}
```

memo[i][j]的意义是，只考虑前 i 件物品且背包重量为 j 时背包可背负的最大价值。

此处使用**带备忘录的自底向上方法，**计算 memo 数组（备忘录）中所有值。

2.2 状态转移方程：

我们用 w[i]表示 i 的重量 v[i]表示其价值

在第 i 阶段，我们考虑第 i 个物品的时候，只有两种选择——放进背包，或者不放进背包。

于是我们考虑 i-1 阶段的最优解，从这个最优解出发：

A.此时当背包承重不足以放下当前物品，那么当然只能不放；

B.而当背包承重足够时，我们有放或不放两种选择：假设放 i 物品放入包中，那么背包负重就减少该物品的重量(j-w[i])，并且总价值增加该物品的价值，**而对于放进 i 后的背包的剩余空间，也就是一个容量为 j-w[i]的背包，它能容纳的最高价值我们已经计算过了，我们只需比较物品 i 的价值和剩余部分的最高价值的和，以及不放 i 的情况下背包的最大价值，这两个价值哪个大，就可以知道放这个物品到底是有益还是有害。**

也就是说，新的最优解，要么是 c[i-1][j-w[i]] + v[i]，也就是我们选择放进了物品 i，并根据剩余的质量求出剩下部分的最优解，两值相加

要么是 c[i-1][j] 不放物品 i 依旧是上个阶段的最优解

也就是说，状态转移的方程是

```
memo[i][j] = Max ( c[i-1][j-w[i]] + v[i] , c[i-1][j] )
```

2.3 例子

以一个 n=5,w=10 的背包问题作为例子，其生成的备忘录如下，memo[5][10]即为原问题的解：

```
自底向上方式最高价值：19
       n\w              1       2       3       4       5       6       7       8       9       10
-------------------------------------------------------------------------------------------------------
第1件物品，价值9,质量4:  0       0       0       9       9       9       9       9       9       9
第2件物品，价值6,质量3:  0       0       6       9       9       9       15      15      15      15
第3件物品，价值1,质量5:  0       0       6       9       9       9       15      15      15      15
第4件物品，价值4,质量2:  0       4       6       9       10      13      15      15      19      19
第5件物品，价值1,质量5:  0       4       6       9       10      13      15      15      19      19
```

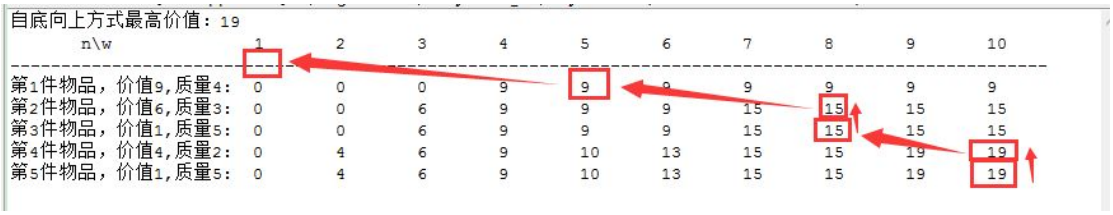
Figure 1 W=10,N=5 的某背包问题

3. 如何知道最优解背包中包含了哪些物品

3.1 以下给出算法的核心代码之二——回溯求物品：

```
/**
 * 根据备忘录的最后一个，回溯地去求背包中放进了什么东西
 */
private static void getPackages() {
    int j = W; // 从备忘录最右下角开始回溯
    for (int i = N; i >= 1; i--) {
        if (memo[i][j] > memo[i - 1][j]) { // 后面的价值更大，说明第i个物品是被放进去的
            j -= itemsArray[i].weight; // 回溯
            itemsIn.add(itemsArray[i]);
        }
        // 等于的情况就跳过这个i，不会出现小于的情况，因为重量增加了最大价值怎么可能变小啊
    }
}
```

3.2 算法解释



首先观察上表，可以看出右下角 memo[5][10]为最高价值，那么假设我们在第 10 列往上看：

A.如果 memo[4][10]与 memo[5][10]相等——意味着第五件物品没有被选择，或者有某件物品与之等价

B.如果 memo[5][10] 大于 memo[4][10]——意味着考虑了第五个物品以后，承重为 10 的背包最高价值大于只考虑了前 4 个物品的最高价值，也就是这个物品被放入了背包。

C. 在此例中，memo[5][10]=memo[4][10],说明第五个物品没有被选择，接着同样的，比较 memo[4][10]与 memo[3][10]，发现 memo[4][10]>memo[3][10]，意味着第四件物品被选择了，接下来，我们既然知道第四件物品被选择了，意味着背包容量少了 v[4]，因而应该去到 memo[3][w-v[4]]即 memo[3][8]处继续往上回溯。

D.从而我们知道，第 4,2,1 个物品被选择了

4. 随机生成 n 个物品的体积与价值，并对给出的三组数据进行验证。

第一组：

（n=10,W=30）

```
/usr/lib/jvm/java-8-oracle/bin/java ...
背包容量为30,物品个数为10的生成结果：
编号：4 重量：0 价值：487
编号：0 重量：2 价值：930
编号：5 重量：3 价值：854
编号：6 重量：4 价值：874
编号：9 重量：6 价值：448
编号：7 重量：9 价值：845
编号：3 重量：12 价值：972
编号：8 重量：23 价值：822
编号：1 重量：24 价值：300
编号：2 重量：26 价值：248
最大价值：4565
被放进去的物品：
编号：5 重量：3 价值：854
编号：3 重量：12 价值：972
编号：0 重量：2 价值：930
编号：6 重量：4 价值：874
编号：9 重量：6 价值：448
编号：4 重量：0 价值：487
填表耗时:0ms
求解耗时:0ms
总耗时:0ms
```

（n=10,W=40）

```
/usr/lib/jvm/java-8-oracle/bin/java ...
背包容量为40,物品个数为10的生成结果：
编号：9 重量：7 价值：658
编号：4 重量：16 价值：668
编号：6 重量：22 价值：777
编号：8 重量：23 价值：264
编号：7 重量：26 价值：596
编号：1 重量：32 价值：631
编号：2 重量：36 价值：348
编号：0 重量：37 价值：586
编号：5 重量：38 价值：294
编号：3 重量：39 价值：343
最大价值：1445
被放进去的物品：
编号：6 重量：22 价值：777
编号：4 重量：16 价值：668
填表耗时:0ms
求解耗时:0ms
总耗时:0ms

Process finished with exit code 0
```

（n=10,W=50）

```
/usr/lib/jvm/java-8-oracle/bin/java ...
背包容量为50,物品个数为10的生成结果：
编号：7 重量：1 价值：229
编号：1 重量：6 价值：593
编号：9 重量：25 价值：442
编号：3 重量：34 价值：40
编号：2 重量：36 价值：939
编号：4 重量：37 价值：690
编号：5 重量：40 价值：281
编号：8 重量：41 价值：198
编号：0 重量：43 价值：318
编号：6 重量：47 价值：305
最大价值：1761
被放进去的物品：
编号：2 重量：36 价值：939
编号：7 重量：1 价值：229
编号：1 重量：6 价值：593
填表耗时:1ms
求解耗时:0ms
总耗时:1ms

Process finished with exit code 0
```

第二组：

（n=20,W=30）

```
/usr/lib/jvm/java-8-oracle/bin/java ...
背包容量为30,物品个数为20的生成结果：
编号：17 重量：0 价值：47
编号：4 重量：1 价值：63
编号：19 重量：2 价值：371
编号：15 重量：3 价值：750
编号：13 重量：4 价值：934
编号：7 重量：8 价值：911
编号：16 重量：9 价值：687
编号：3 重量：11 价值：725
编号：6 重量：13 价值：951
编号：1 重量：15 价值：175
编号：8 重量：16 价值：67
编号：18 重量：17 价值：115
编号：12 重量：19 价值：322
编号：10 重量：20 价值：403
编号：0 重量：22 价值：566
编号：9 重量：23 价值：226
编号：2 重量：25 价值：631
编号：11 重量：26 价值：762
编号：14 重量：27 价值：213
编号：5 重量：29 价值：53
最大价值：3917
被放进去的物品：
编号：15 重量：3 价值：750
编号：6 重量：13 价值：951
编号：19 重量：2 价值：371
编号：13 重量：4 价值：934
编号：7 重量：8 价值：911
填表耗时:0ms
求解耗时:0ms
总耗时:0ms

Process finished with exit code 0
```

（n=20,W=40）

```
/usr/lib/jvm/java-8-oracle/bin/java ...
背包容量为40,物品个数为20的生成结果：
编号：9 重量：1 价值：943
编号：0 重量：2 价值：691
编号：16 重量：5 价值：466
编号：13 重量：7 价值：55
编号：2 重量：8 价值：98
编号：4 重量：9 价值：774
编号：5 重量：11 价值：702
编号：8 重量：14 价值：980
编号：12 重量：16 价值：359
编号：11 重量：17 价值：582
编号：7 重量：18 价值：654
编号：3 重量：21 价值：589
编号：1 重量：24 价值：760
编号：6 重量：25 价值：941
编号：14 重量：28 价值：56
编号：10 重量：30 价值：706
编号：15 重量：32 价值：465
编号：17 重量：36 价值：118
编号：19 重量：38 价值：465
编号：18 重量：39 价值：96
最大价值：4090
被放进去的物品：
编号：0 重量：2 价值：691
编号：8 重量：14 价值：980
编号：9 重量：1 价值：943
编号：4 重量：9 价值：774
编号：5 重量：11 价值：702
填表耗时:0ms
求解耗时:0ms
总耗时:0ms

Process finished with exit code 0
```

（n=20,W=50）

```
背包容量为50,物品个数为20的生成结果：
编号：1 重量：0 价值：7
编号：19 重量：3 价值：249
编号：18 重量：7 价值：830
编号：17 重量：8 价值：839
编号：4 重量：10 价值：228
编号：7 重量：17 价值：952
编号：3 重量：19 价值：552
编号：13 重量：20 价值：22
编号：5 重量：22 价值：40
编号：2 重量：23 价值：142
编号：6 重量：25 价值：299
编号：0 重量：27 价值：219
编号：11 重量：28 价值：111
编号：12 重量：30 价值：49
编号：10 重量：31 价值：777
编号：14 重量：36 价值：576
编号：9 重量：37 价值：257
编号：8 重量：40 价值：947
编号：15 重量：42 价值：250
编号：16 重量：43 价值：536
最大价值：3105
```

```
被放进去的物品：
编号：18 重量：7 价值：830
编号：7 重量：17 价值：952
编号：19 重量：3 价值：249
编号：17 重量：8 价值：839
编号：4 重量：10 价值：228
编号：1 重量：0 价值：7
填表耗时:1ms
求解耗时:0ms
总耗时:1ms
```

第三组：

（n=30,W=30）

```
/usr/lib/jvm/java-8-oracle/bin/java ...
背包容量为30,物品个数为30的生成结果：
编号：20 重量：0 价值：639
编号：21 重量：1 价值：69
编号：1 重量：2 价值：452
编号：11 重量：3 价值：427
编号：27 重量：4 价值：579
编号：16 重量：5 价值：789
编号：8 重量：6 价值：105
编号：13 重量：7 价值：226
编号：24 重量：8 价值：674
编号：26 重量：9 价值：816
编号：15 重量：10 价值：940
编号：2 重量：11 价值：187
编号：12 重量：12 价值：362
编号：22 重量：13 价值：452
编号：9 重量：14 价值：58
编号：4 重量：15 价值：530
编号：28 重量：16 价值：501
编号：25 重量：17 价值：129
编号：29 重量：18 价值：115
编号：5 重量：19 价值：143
编号：18 重量：20 价值：169
编号：23 重量：21 价值：392
编号：14 重量：22 价值：142
编号：10 重量：23 价值：671
编号：6 重量：24 价值：521
编号：0 重量：25 价值：424
编号：7 重量：26 价值：443
编号：3 重量：27 价值：165
编号：19 重量：28 价值：373
编号：17 重量：29 价值：191
最大价值：4073
被放进去的物品：
编号：27 重量：4 价值：579
编号：15 重量：10 价值：940
编号：1 重量：2 价值：452
编号：16 重量：5 价值：789
编号：24 重量：8 价值：674
编号：20 重量：0 价值：639
填表耗时:0ms
求解耗时:1ms
总耗时:1ms

Process finished with exit code 0
```

（n=30,W=40）

```
/usr/lib/jvm/java-8-oracle/bin/ja
背包容量为40,物品个数为30的生成结果：
编号：1 重量：0 价值：528
编号：4 重量：1 价值：770
编号：18 重量：3 价值：212
编号：13 重量：4 价值：615
编号：26 重量：5 价值：762
编号：8 重量：7 价值：432
编号：0 重量：10 价值：658
编号：21 重量：11 价值：989
编号：3 重量：13 价值：656
编号：10 重量：15 价值：72
编号：20 重量：16 价值：7
编号：15 重量：17 价值：661
编号：25 重量：18 价值：30
编号：16 重量：19 价值：418
编号：11 重量：20 价值：492
编号：22 重量：21 价值：68
编号：24 重量：22 价值：556
编号：6 重量：24 价值：448
编号：28 重量：25 价值：635
编号：9 重量：26 价值：677
编号：29 重量：28 价值：840
编号：23 重量：29 价值：704
编号：7 重量：31 价值：589
编号：27 重量：32 价值：578
编号：17 重量：33 价值：845
编号：2 重量：34 价值：883
编号：19 重量：35 价值：924
编号：12 重量：36 价值：933
编号：5 重量：38 价值：935
编号：14 重量：39 价值：517
最大价值：4754
被放进去的物品：
编号：26 重量：5 价值：762
编号：21 重量：11 价值：989
编号：1 重量：0 价值：528
编号：13 重量：4 价值：615
编号：8 重量：7 价值：432
编号：0 重量：10 价值：658
编号：4 重量：1 价值：770
填表耗时:0ms
求解耗时:0ms
总耗时:0ms
```

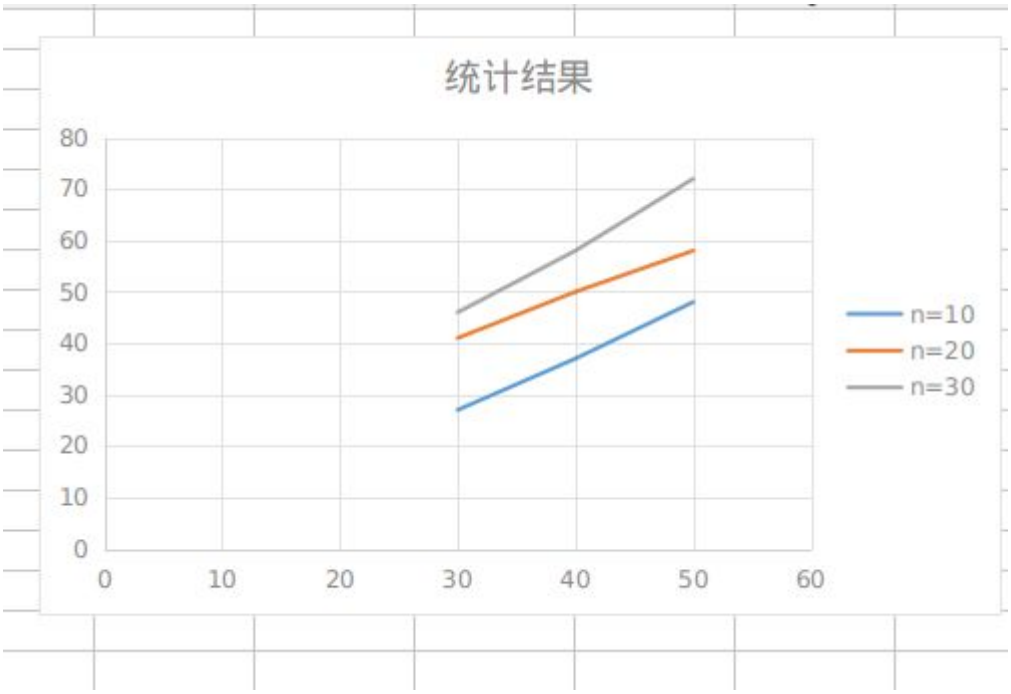
（n=30,W=50）

```
/usr/lib/jvm/java-8-oracle/bin/java ...
背包容量为50,物品个数为30的生成结果：
编号：19 重量：2 价值：849
编号：1 重量：5 价值：809
编号：6 重量：6 价值：20
编号：29 重量：7 价值：398
编号：14 重量：8 价值：304
编号：8 重量：9 价值：37
编号：9 重量：10 价值：556
编号：0 重量：11 价值：255
编号：17 重量：12 价值：681
编号：23 重量：15 价值：357
编号：27 重量：16 价值：992
编号：20 重量：17 价值：345
编号：3 重量：18 价值：931
编号：26 重量：19 价值：942
编号：28 重量：20 价值：858
编号：21 重量：22 价值：695
编号：5 重量：23 价值：139
编号：13 重量：27 价值：327
编号：12 重量：30 价值：602
编号：24 重量：32 价值：24
编号：7 重量：33 价值：256
编号：22 重量：34 价值：382
编号：16 重量：35 价值：136
编号：4 重量：36 价值：902
编号：15 重量：38 价值：371
编号：10 重量：40 价值：938
编号：2 重量：42 价值：980
编号：18 重量：46 价值：6
编号：25 重量：48 价值：920
编号：11 重量：49 价值：460
最大价值：4033
被放进去的物品：
编号：29 重量：7 价值：398
编号：27 重量：16 价值：992
编号：1 重量：5 价值：809
编号：14 重量：8 价值：304
编号：17 重量：12 价值：681
编号：19 重量：2 价值：849
填表耗时:1ms
求解耗时:0ms
总耗时:1ms
```

三、 数据处理分析

分别重复 1000 次实验，算出总时间：

	A	B	C	D
	第一组：(n=10)			
	w的值	30	40	50
	1000次花的时间	27	37	48
	第二组：(n=20)			
	w的值	30	40	50
	1000次花的时间	41	50	58
	第三组：(n=30)			
	w的值	30	40	50
	1000次花的时间	46	58	72



可以看到，曲线接近是十分完美的直线。在 n 固定的情况下，算法的运行时间随着 w 的增加而线性增加，这也与理论符合。因为该算法主要就是计算 memo 这个二维数组，该数组的大小是 n*w，当 n 确定，算法运行时间自然是线性增加的。

四、实验总结

通过本次实验我理解了动态规划算法设计的思想。其核心其实就是分治，并比分治多了一步解决冗余。

指导教师批阅意见： 成绩评定： 指导教师签字： 年　　月　　日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。