北京科技大学实验报告

学院： 计算机与通信工程学院 专业： 计算机科学与技术 班级： 计 184

姓名： 王丹琳 学号： 41824179 实验日期： 2020 年 12 月 11 日

1. 实验名称：

上机实验-3 压缩歌曲

2. 实验目的：

蒜头君的电脑里面有一些歌。现在他需要把这些歌装进一个硬盘里面。

硬盘大小有限，由于蒜头君下载的都是无损版本，每首歌的占用空间比较大，硬盘不一定装得下，然后他需要压缩其中的一部分歌曲这样他才能将尽可能多的歌曲装进他的硬盘里。

但是他想尽量压缩的歌曲数量尽量少，他不知道该怎么做，就来找你帮忙了。

输入格式：输入的第一行包含两个整数 n 和 m（$1\le n\le 105$，$1\le m\le 109$），分别表示蒜头君电脑里面歌曲的个数和他的硬盘大小（单位：字节）。然后输入 n 行，每一行两个整数 ai 和 bi（$1\le bi<ai\le 109$），分别表示第 i 首歌曲原本的大小和被压缩后的大小（单位：字节）。

输出格式：输出只有一个整数，蒜头君至少需要压缩的歌曲的数量。如果所有的歌曲都压缩的硬盘还是装不下，输出 -1。

3. 实验环境：

编程语言：C++，环境：计蒜客

4. 实验原理：

通过题目可以发现，此题主要考察贪心算法，对尽可能少的压缩歌曲，只需考虑当前局部信息（可以通过数学方法来推导出）。局部最优（当前尽可能少压缩）就可以导致全局最优（最后的歌曲最少压缩数），与贪心算法基本思路相似，本题基本思路是从问题的某一个初始解出发逐步逼近给定的目标，在每一步都做一个不可回溯的决策，尽可能地少压缩歌曲，当磁盘放不下时，算法停止。

5. 实验内容与步骤：

5.1 问题分析

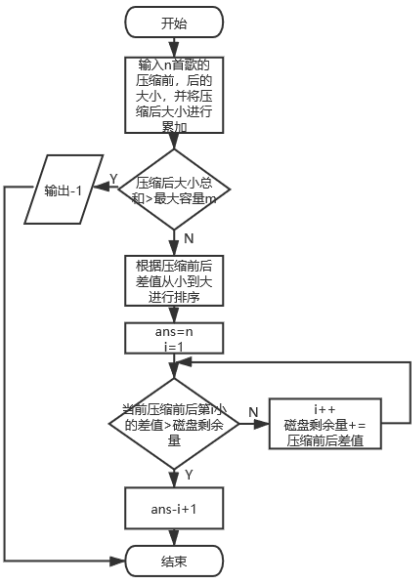
由于硬盘大小有限，所以需要压缩其中的一部分歌曲这样他才能将尽可能多的歌曲装进他的硬盘里。很明显此题运用到的是贪心算法。所以按照贪心算法，为了尽可能多的将歌曲装入硬盘，我们假设全部按照压缩后的量装入键盘，若全部压缩还不能装入的话，就不可能还能全部装入，按照题目要求输出-1，若可以全部装入，我们就可以开始思考如何尽可能的少压缩歌曲，根据贪心算法的思想，我们将压缩掉的大小进行排序，从压缩掉的最小值开始交换，这样，硬盘每次随着歌曲从压缩到被压缩的交换，增长的幅度比较小，可以交换进更多的无损歌曲，这样也达到了题目所要求的尽可能多的无损歌曲。

以上即为本题解题的主要思路。

5.2 算法设计

```
if (cur > m) {
        cout << -1;
        return 0;
    }
    sort(s, s + n, cmp);
    i = 0; ans = n;
    while (cur + s[i].c <= m&&i<n)
    {
        ans--;
        cur += s[i].c;
        i++;
    }
    cout << ans;
```

5.3 算法流程图



5.4 算法实现

```
struct song {
    int before;
    int after;
    int c;//能压缩掉的大小
};
int cmp(song a, song b)//按c从小到大排序
{
    return a.c < b.c;
}
```

5.5 实验测试方案

实验测试方案：

输入 4 21 10 8 7 4 3 1 5 4，即有 4 首歌（压缩前后大小分别为 10 和 8，7 和 4，3 和 1，5 和 4），磁盘总容量为 21，经过计算可以得出至少需要压缩的歌曲的数量为 2

输入 4 16 10 8 7 4 3 1 5 4，即有 4 首歌（压缩前后大小分别为 10 和 8，7 和 4，3 和 1，5 和 4），磁盘总容量为 16，经过计算可以得出即使歌曲全部压缩，也无法将歌曲全部放入，故输出为-1

实验测试数据：

输入 4 21 10 8 7 4 3 1 5 4，预期输出为 2

输入 4 16 10 8 7 4 3 1 5 4，预期输出为-1

5.6 实验数据：

样例输入 1

```
4 21
10 8
7 4
3 1
5 4
```

样例输出 1

```
2
```

样例输入 2

```
4 16
10 8
7 4
3 1
5 4
```

样例输出 2

```
-1
```

5.7 实验数据处理：

输入 4 21 10 8 7 4 3 1 5 4，即有 4 首歌（压缩前后大小分别为 10 和 8，7 和 4，3 和 1，5 和 4），磁盘总容量为 21，经过计算可以得出至少需要压缩的歌曲 2 首，即仅需把原大小为 7 压缩为 4，原大小为 10 压缩为 8，就可以全部装入并且压缩歌曲数最小。

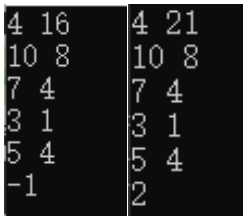
输入 4 16 10 8 7 4 3 1 5 4，即有 4 首歌（压缩前后大小分别为 10 和 8，7 和 4，3 和 1，5 和 4），磁盘总容量为 16，经过计算可以得出即使歌曲全部压缩（全部压缩后也得需要 17 大小的磁盘空间，大于现有的 16 大小的磁盘空间），也无法将歌曲全部放入，故输出为-1

6. 实验结果与分析

6.1 测评结果

计蒜客测评结果

线下模拟测试结果



6.2 实验结果和算法分析

对于测试结果的分析：测试结果符合预期

时间复杂度和空间空间复杂度计算:时间复杂度为 O(n),空间复杂度也为 O(n)

7. 实验总结：

通过这次实验，我深刻理解了贪心算法的基本思路与适用的问题，贪心算法是从问题的某一个初始解出发逐步逼近给定的目标,每一步都做一个不可回溯的决策，尽可能地求得最好的解，当某一部不需要在继续前进时，算法结束。

同时，在运用贪心算法进行解题时，需要保证该题“局部最优策略能导致全局最优解”，若不能保证则应该考虑动态规划等算法。

在做题的过程中，我也熟悉了贪心算法的算法框架：

从问题的某一个初始解出发;

while（朝着给定总目标前进一步）do

利用可行决策，求出可行解的一个解元素;

由所有解元素组合成问题的一个可行解;

北京科技大学实验报告 4

学院： 计算机与通信工程学院 专业： 计算机科学与技术 班级： 计 184

姓名： 王丹琳 学号： 41824179 实验日期： 2020 年 12 月 9 日

1. 实验名称：

上机实验-4 最多现金

2. 实验目的：

小明报名参加了一个活动并获得了奖励，奖励规则如下：在活动现场有 M 个一字排开的盒子，盒身标签写明了盒中的现金数，小明可以拿走任意不相邻盒子中的现金。请问小明最多可以拿走多少现金？

输入格式：输入的第一行是一个整数 T(T≤50) ，表示一共有 T 组数据。接下来的每组数据，第一行是一个整数 M(1≤M≤100,000)，表示一共有 M 个盒子。第二行是 M 个被空格分开的正整数，表示每个盒子里的现金数量。每个盒子的现金数量均不超过 1000。

输出格式：对于每组数据，输出一行。该行包含一个整数，表示小明可以得到的现金数量。

3. 实验环境：

编程语言：C++，环境：计蒜客

4. 实验原理：

在本题中，可以看出拿钱的决策不是线性的而是全面考虑不同情况分别进行决策，并通过多阶段决策来最终解决问题。这与动态规划的思想十分契合。同时此题与上面一题有所不同，本题若采用贪心算法，尝试用局部最优来获取全局最优是不可行的，因为本题需要全面考虑不同情况来取得最优解。此题体现了适合动态规划的问题特征：最优化原理（请问小明最多可以拿走多少现金）、无后向性（如果拿了某个盒子中钱，则从此阶段以后过程的发展变化仅与此阶段的状态有关，而与过程在此阶段以前的阶段所经历过的状态无关）、子问题重叠性质（都为拿或不拿此

盒子中的钱）。

5. 实验内容与步骤：

5.1 问题分析

根据题意，很明显这是一个动态规划问题，在当前状态采取的行动可根据前面状态采取的行动来决定。

在该问题中，小明不能拿相邻的盒子中的钱，那么对于第 m (m>2) 个盒子来说：

拿第 m 个盒子，则第 m-1 的盒子不能拿，因此，此时拿的总金额为前 m-2 个盒子的最大金额总和加上当前盒子里的金额。

不拿第 m 个盒子，则所拿的总金额为前 m-1 个盒子的最大金额总和。

在 1 和 2 选项中，选择金额最大的即为当前状态下的所能拿到的最高金额。

因此，可得到状态转移方程：dp[i] = max(dp[i - 1], dp[i - 2] + money[i]);

其中，边界条件为：

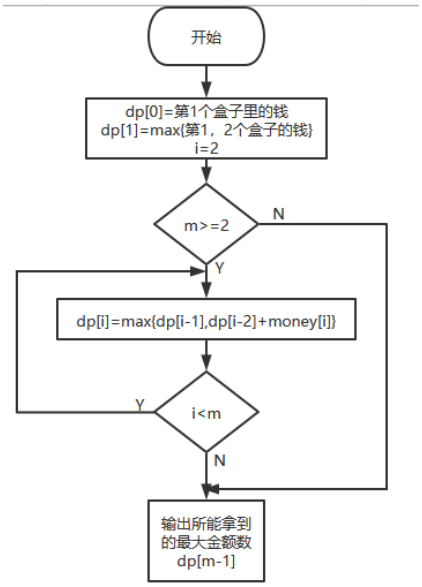
1）只有 1 个盒子，dp [0] = money[0]；

2）只有 2 个盒子，dp [1] = max(money[0], money [1])，拿钱多的那个盒子。

5.2 算法设计

```
int take_money(int* money,int m) {
    dp[0] = money[0];
    dp[1] = max(money[0], money[1]);
    for (i = 2; i < m; i++) //第三个开始
        dp[i] = max(dp[i - 1], dp[i - 2] + money[i]);//状态转移方程
    return dp[m - 1];
}
```

5.3 算法流程图



5.4 算法实现

5.5 实验测试方案

实验测试方案：

输入 2 3 1 8 2 4 10 7 6 14，即有 2 组盒子（第 1 组有 3 个盒子里面分别有 1，8，2；第 2 组有 4 个盒子里面分别有 10，7，6，14），经过计算可以得出第一组盒子最多可以拿走 8 现金，第二组盒子最多可以拿走 24 现金

实验测试数据：

输入 2 3 1 8 2 4 10 7 6 14，预期输出为 8 24

5.6 实验数据：

样例输入

```
2
3
1 8 2
4
10 7 6 14
```

样例输出

```
8
24
```

5.7 实验数据处理：

输入 2 3 1 8 2 4 10 7 6 14，即有 2 组盒子（第 1 组有 3 个盒子里面分别有 1，8，2；第 2 组有 4 个盒子里面分别有 10，7，6，14），经过计算可以得出第一组

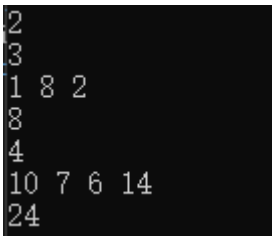
盒子最多可以拿走 8 现金（拿走第二个盒子里的钱），第二组盒子最多可以拿走 24 现金（拿走第 1 个和第 4 个盒子里的钱）

6. 实验结果与分析

6.1 测评结果

计蒜客测评结果

线下模拟测试结果



6.2 实验结果和算法分析

对于测试结果的分析：测试结果符合预期

时间复杂度和空间空间复杂度计算：时间复杂度：O(n)，n 为数组长度

空间复杂度：O(n)，n 为数组长度

7. 实验总结：

通过这次实验，我深刻理解了动态规划法的基本思路与适用的问题，动态规划法是通过多阶段决策过程来解决问题的。每个阶段的决策结果都是一个决策结果的序列，这个结果序列中最后采用哪一个结果都取决于以后每个阶段的决策。适合动态规划的问题主要都有最优化原理、无后向性、子问题重叠三个特征。在本题中，最优化原理体现在小明最多可以拿走的现金上、无后向性体现在拿了某个盒子中钱后，以后过程的发展变化仅与当前的状态有关，而与拿所经历过的状态无关该盒子之前的状态无关、子问题重叠性质体现在都是拿或不拿此盒子中的钱。

动态规划法的关键为找到递推公式与边界条件，主要步骤为定义子问题、写出子问题的递推关系、确定 DP 数组的计算顺序（自顶向下、使用备忘录的递归方法或是自底向上的循环方法）。本题采用的是自底向上的循环。在计算一个子问题时，它所依赖的那些子问题已经全部计算出来了。