

LAB2

实验名称：动态规划算法实验

实验内容：

1. 动态规划算法实现思路：

类似于 0-1 背包问题，我们使用数组 `bgt` 来表示对应预算（下标）下的最大支出，另外再使用 `select[i]` 记录最后一个选择的物品用于回溯。考虑到每个物品选择一次，我们先对物品价格 `price` 遍历，对于每个递减的大于 `price` 的预算，令 $bgt[i] = \max(bgt[i], bgt[i - price] + price)$ 。这样遍历结束后 `bgt` 就存储了相应的最大支出，并且保证了仅选择一遍。后面使用 `select` 数组进行回溯即可。源代码文件已保存在目录下。

2. dp 部分伪代码说明：

```
Initialize bgt[0..budget] to 0
Initialize select[0..budget] to -1
for i from 0 to number_of_items - 1 do:
    for j from budget down to price[i] do:
        if bgt[j] < bgt[j - price[i]] + price[i] then:
            bgt[j] = bgt[j - price[i]] + price[i]
            select[j] = i
Initialize item_list as an empty list
m = budget
while m > 0 do:
    itemIndex = select[m]
    if itemIndex != -1 then:
        item_list.append(itemIndex)
        m = m - price[itemIndex]
    else:
        break
return bgt[budget]
```

3. 处理数据集性能的比较和分析。

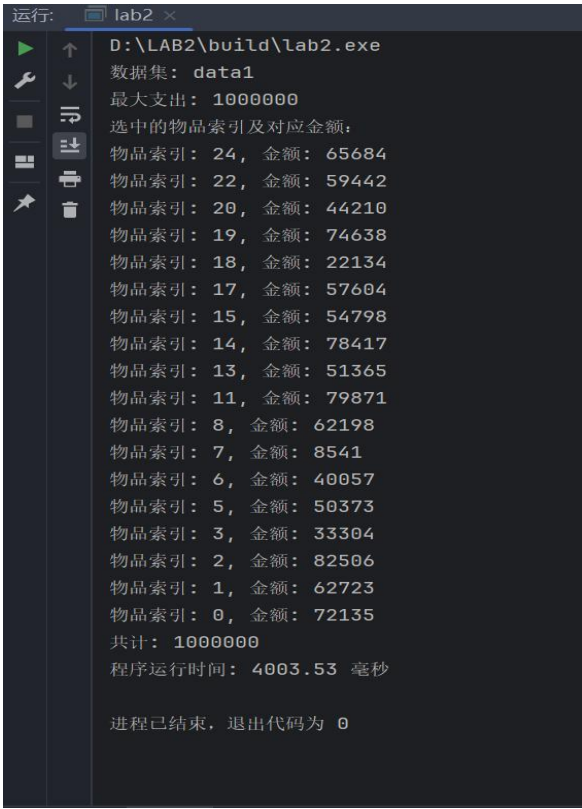
尽管 `data1` 数据中有共计 1000 组物品价格，而 `data2` 中仅有 3 组数据，人工

也可以很快地得出结论，但在动态规划算法中，data2 的计算反而比 data1 更为复杂。在 Windows11 + clion + MinGW11.0 + cmake3.27 + Debug 环境下，测试 data1 数据集为 4003.5ms，data2 数据集为 18301ms，是前一数据集的 4.57 倍。在占用内存上，data1 仅占用了 8.2MB 内存，data2 占用了 7629.3MB，占运行内存的 63.8%。

4. 局限性。

不难发现，是两个数据集的预算差距产生的运行时间与内存占用的巨大差距，在物品数量远远小于预算时，往往是预算决定了性能的不同。从算法层面上解释，是由于预算决定了数组的大小，而数组的大小决定了需要计算的数量，从而在两方面都影响程序性能。因此，动态规划更适合预算小而物品数量大的问题，不适合解决如 data2 此类预算大而物品数量小的问题。

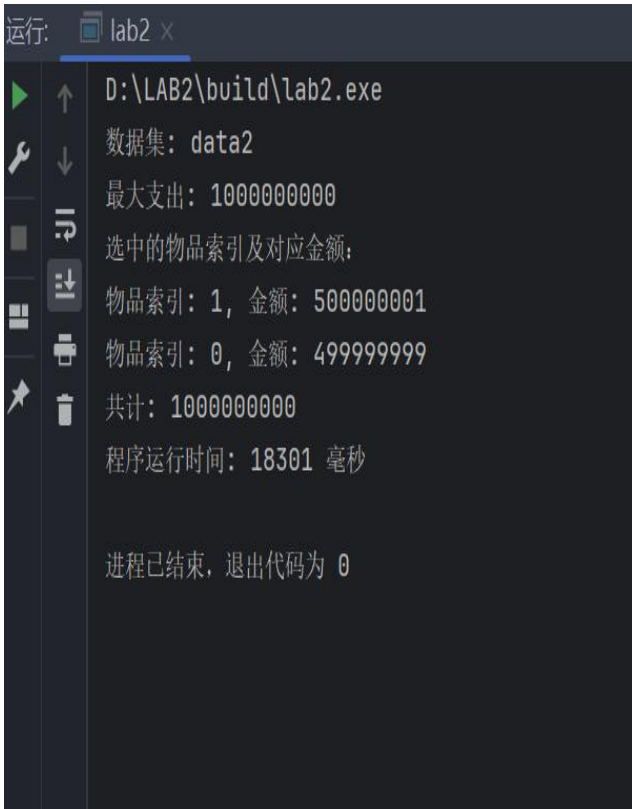
5. 运行结果截图。



```
运行: lab2 x
D:\LAB2\build\lab2.exe
数据集: data1
最大支出: 10000000
选中的物品索引及对应金额:
物品索引: 24, 金额: 65684
物品索引: 22, 金额: 59442
物品索引: 20, 金额: 44210
物品索引: 19, 金额: 74638
物品索引: 18, 金额: 22134
物品索引: 17, 金额: 57604
物品索引: 15, 金额: 54798
物品索引: 14, 金额: 78417
物品索引: 13, 金额: 51365
物品索引: 11, 金额: 79871
物品索引: 8, 金额: 62198
物品索引: 7, 金额: 8541
物品索引: 6, 金额: 40057
物品索引: 5, 金额: 50373
物品索引: 3, 金额: 33304
物品索引: 2, 金额: 82506
物品索引: 1, 金额: 62723
物品索引: 0, 金额: 72135
共计: 10000000
程序运行时间: 4003.53 毫秒

进程已结束, 退出代码为 0
```

图 1 data1 计算结果



```
运行: lab2 x
D:\LAB2\build\lab2.exe
数据集: data2
最大支出: 10000000000
选中的物品索引及对应金额:
物品索引: 1, 金额: 5000000001
物品索引: 0, 金额: 4999999999
共计: 10000000000
程序运行时间: 18301 毫秒

进程已结束, 退出代码为 0
```

图 2 data2 计算结果