实验报告

实验题目：栈、队列及其应用

计算机学院3班，雷雨轩，PB18111791

完成日期：2019年11月14日

实验要求(背包问题)

假设有一个能装入总体积为T的背包和n件体积分别为w1 , w2 , ... , wn 的物品，能否从n件物品中挑选若干件恰好装满背包，要求找出所有满足上述条件的解。

提示：利用回溯法的设计思想来解决背包问题，请勿枚举或动态规划。

基本要求：

输出所有恰好装满背包的方案。

选做要求：

若不能恰好装满背包，则输出所有占用背包空间最大的方案。

输入输出样例：

```
Input:
10 6         //背包体积T与物品数n
1 8 4 3 5 2 //n件物品的体积
Output:
4            //解的个数，下面每一行输出一组解
1 4 3 2
1 4 5
8 2
3 5 2
```

设计思路

- 本次实验关键是利用递归来模拟人为求解该问题的思路。

  输入n个物品体积以及背包体积后，对人来说，正常求解顺序是：把第一个物品放入背包（如果放得下），再放第二个物品，依次放，直到背包剩余容积为0（此为一个解，记录），或为负（说明最后放入的那个物品不满足要求），两种情况都需要取出该物品（回溯），再尝试放入下一个物品）。当一轮尝试完成后，则回退倒数第二放入背包的物品，尝试放入下一个物品。

这样的回溯法思想正好与递归层级相契合

- 代码分为两个模块：

  主函数模块

  实现输入，函数调用，结果输出

  void search(int pos,int rest)函数：

记录从pos号物品开始（按输入顺序，从0计数），将物品放入剩余容积为rest的背包的方案数。

显然该递归函数的实现要考虑

1. 参数
2. 递归终止条件
3. 将大问题拆分为小问题。只需拆解为两部分
    - 将i号物品最先放入背包
    - 求解从i+1号物品开始，将物品放入剩余容积为T-wi的背包的方案

- 另外，也通过使用vector模块来存放每个方案，极大简化了存数据的操作
- 选做要求的实现，有两种方法
    - 简单粗暴的方式就是将V(不能装满时，占用最大背包空间)从T开始，找是否有满足条件的方案，有则输入，无则V--，但此方法在T与V相差较大时，需要在主函数中调用多次递归函数耗时巨大
    - 另一种则可以在递归函数中做一定条件判断，总能在一次递归函数调用后就找到V，此时再对V做一次递归函数调用则可以找到所有方案，主函数中总共只需两次递归函数调用

关键代码讲解

```
//数据定义
int T, n;
int a[100];                             //n件物品的体积
int count1 = 0;                         //记录方案数
int V = 0;                              //不能装满情况下，能占用的最大空间
vector<vector<int>> result;             //装所有可行方案
vector<int> temp;                       //在递归过程中临时装一组方案
```

基本要求:

```
//递归函数
void search(int pos, int rest) {    //pos为还未尝试放置的第一个物品位置(按输入
//先后顺序),rest为背包剩余容积
    if (rest < 0) return;           //如果背包剩余容积为负，返回

    if (rest == 0) {                //如果背包容积为0，说明此时temp里装的数据
//为一组答案
        result.push_back(temp);
        count1++;
        return;
    }
    for (int j = pos; j < n; j++) { //思路：将未尝试放置物品依次放入temp尝试
        temp.push_back(a[j]);
        search(j+1, rest - a[j]);
        temp.pop_back();            //回溯
    }
    return;
}
```

选做要求:

```
//递归函数
void search(int pos, int rest) {     //pos为还未尝试放置的第一个物品位置(按输入
//先后顺序),rest为背包剩余容积
    if (rest < 0) return;            //如果背包剩余容积为负，返回

    if (rest == 0) {                 //如果背包容积为0，说明此时temp里装的数据
//为一组答案
        result.push_back(temp);
        count1++;
        return;
    }
    for (int j = pos; j < n; j++) { //思路：将未尝试放置物品依次放入temp尝试
        temp.push_back(a[j]);
        if (((T - V) > (rest - a[j]))&&(rest-a[j])>=0) V = 10 - rest + a[j];
//条件判断。看当前记录的V与放入a[j]后T中已被占用的空间大小相比较
        search(j + 1, rest - a[j]);
        temp.pop_back();             //回溯
    }
    return;
}
```

```
//主函数
int main()
{

    cin >> T >> n;   //背包体积T,物品数n
    for (int i = 0; i < n; i++) {
        cin >> a[i];  //n件物品的体积
    }
    search(0, T);
    if (count1 == 0) {  //此if语句为选做要求的代码
        search(0, V);
    }
    cout << count1 << endl;
    for (auto it = result.begin(); it != result.end(); it++) {  //输出
        for (auto iter = it->begin(); iter != it->end(); iter++) {
            cout << *iter << " ";
        }
        cout << endl;
    }
    return 0;
}
```

事实上，选做要求的代码只需在基础要求的基础上添加简单几行代码就能在同一文件同时实现，详见提交地代码

调试分析

时间复杂度:

主要考虑search函数，因为每次进入有n次for循环，所以时间复杂度为O(n^n)

空间复杂度:

输入数据所占空间只取决于问题本身，和算法无关，但考虑到vector二维空间的使用，所以复杂度为O(n^2)

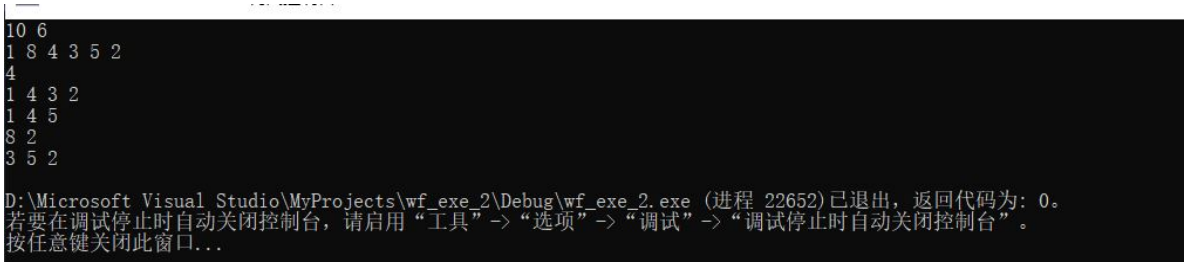
遇到的问题

- 实验中面临的主要问题还是在于如何具体实现递归函数，这个题类似于以前做过的N皇后问题，关键在于拆分+回溯的模拟。这个题的思路还是要弄懂，人手动计算的方式如何在计算机中实现。除此之外，再把递归结束条件找全，不漏，那么结果自然就正确了
- 关于数据存放问题，自己想过构建相应结构体、链表来存放，但却是会增加任务量，而且实现比较繁琐，用vector来装数据，一方面方便，另一方面也可以更高效的着重于递归算法的设计

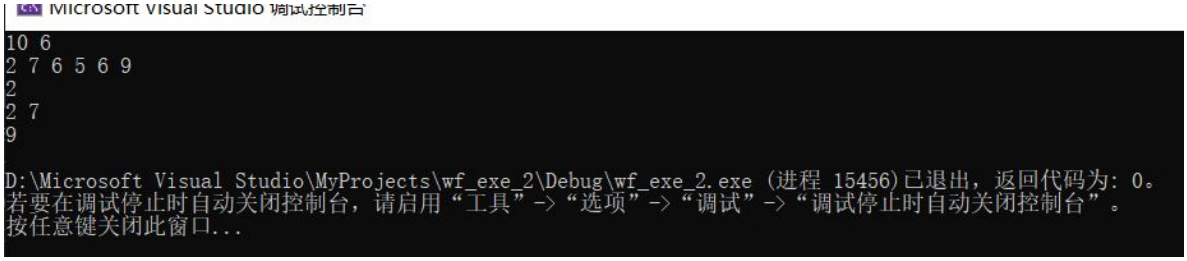
代码测试

按输入样例的格式运行程序并输入即可

基本要求



选做要求



实验总结

通过本次实验，加深了我对栈特性的理解，在回溯、递归算法的实现过程中，自己不断调试修改，让判断条件等不断趋于正确，深刻体会到用递归回溯来模拟人工操作的可行性与优势。选做内容的训练，通过在递归函数中适当添加判断条件，让我能认真地学会准确的把握整个递归调用过程，与栈队列章节中所学的栈实现递归相关知识很好地结合起来。