01背包： 1049. 最后一块石头的重量 II

问题分析：

问题背景

- 石头的重量是 [2, 7, 4, 1, 8, 1]。
- 我们的目标是将这些石头分成两堆，使得两堆石头重量差最小。
- 使用动态规划的方法，我们尝试找到一个子集，使它的重量尽量接近总重量的一半。

1. 初始化

cpp 复制

```cpp
int sum = 0;
for (auto s : stones) {
    sum += s;
}
```

- 先计算石头总重量：

cpp 复制

```cpp
sum = 2 + 7 + 4 + 1 + 8 + 1 = 23
```

- 我们的目标是找到一个子集，其重量尽量接近 sum / 2，即：

cpp 复制

```cpp
n = sum / 2 = 23 / 2 = 11
```

2. 初始化 dp 数组

cpp 复制

```cpp
vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
```

- 初始化一个二维 dp 数组，大小为 (m + 1) x (n + 1)，其中 m 是石头的个数，n 是总重量的一半 11。dp[i][j] 表示使用前 i 块石头，是否能组成和为 j 的子集重量。
- 初始化时，dp 全部为 0，表示还没有任何石头被选择。

3. 动态规划填表

cpp 复制

```cpp
for (int i = 1; i <= m; i++) {
    for (int j = 0; j <= n; j++) {
        dp[i][j] = dp[i - 1][j];  // 不选当前石头
        if (j >= stones[i - 1]) {  // 如果当前石头的重量小于等于 j
            dp[i][j] = max(dp[i][j], dp[i - 1][j - stones[i - 1]] + stones[i - 1]);  // 选择当前石头
        }
    }
}
```

逐步解释

假设石头的重量为 [2, 7, 4, 1, 8, 1]，我们逐步填充 dp 表格：

- **第1块石头 (2)**：
  - 尝试将石头 2 放入容量为 0 ~ 11 的背包。
  - 对于容量小于 2 的背包 (即 j < 2)，不能放入石头，dp[1][j] = dp[0][j] = 0。
  - 对于容量大于等于 2 的背包，放入石头后，dp[1][2] = 2，dp[1][3] = 2，一直到 dp[1][11] 都是 2。

cpp 复制

```cpp
dp[1] = [0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
```

- **第2块石头 (7)**：
  - 对于容量 0 ~ 6，不能放入石头 7，所以 dp[2][j] = dp[1][j]，即：

cpp 复制

```cpp
dp[2][0] = 0, dp[2][1] = 0, ..., dp[2][6] = 2
```

  - 对于容量 7，可以放入石头 7，因此 dp[2][7] = max(dp[1][7], dp[1][7 - 7] + 7) = 7。

- 对于容量 8 ~ 11，可以放入石头 7，并且原来放入了石头 2，所以 dp[2][9] = 9，表示选择了石头 7 和 2。

cpp 复制

```
dp[2] = [0, 0, 2, 2, 2, 2, 2, 7, 7, 9, 9, 9]
```

- **第3块石头 (4)**：
  - 对于容量 4 ~ 11，可以选择石头 4，更新部分状态。

cpp 复制

```
dp[3] = [0, 0, 2, 2, 4, 4, 6, 7, 7, 9, 9, 11]
```

- **第4块石头 (1)**：
  - 对于容量 1 ~ 11，可以选择石头 1，更新部分状态。

cpp 复制

```
dp[4] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 9, 11]
```

- **第5块石头 (8)**：
  - 对于容量 8 ~ 11，可以选择石头 8，更新部分状态。

cpp 复制

```
dp[5] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

- **第6块石头 (1)**：
  - 对于容量 1 ~ 11，可以选择石头 1，更新部分状态。

cpp 复制

```
dp[6] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

4. 返回结果

最后我们得到 dp[6][11] = 11，表示通过选择石头，可以使其中一堆的重量最大为 11。

另一堆的重量是：

cpp 复制

代码：

```cpp
class Solution {
public:
    int lastStoneWeightII(vector<int>& stones) {
        int sum = 0;
        for(auto s : stones)
        {
            sum += s;
        }
        int m = stones.size();
        int n = sum/2;

        vector<vector<int>> dp(m+1,vector<int>(n+1));

        for(int i = 1; i <= m ;i++)
        {
            for(int j = 0;j <=n ;j++)
            {
                dp[i][j] = dp[i-1][j];
                if(j >= stones[i-1])
                dp[i][j] = max(dp[i][j],stones[i-1] + dp[i-1][j-stones[i-1]]);
            }
        }

        return sum - 2*dp[m][n];
    }
};
```

空间优化以后代码：

```cpp
class Solution {
public:
    int lastStoneWeightII(vector<int>& stones) {
        int m = stones.size();
        int sum = 0;
        for(auto s : stones)
        {
            sum += s;
        }
        int n = sum/2;
        vector<int> dp(n+1);
        for(int i = 1; i <= m ;i++)
        {
            for(int j = n; j >= stones[i-1];j--)
            {
                dp[j] = max(dp[j],stones[i-1] + dp[j-stones[i-1]]);
            }
        }
        return sum - 2*dp[n];
    }
};
```

BFS：675. 为高尔夫比赛砍树

代码：

```cpp
class Solution {
    int m, n;
    int dx[4] = {1, -1, 0, 0};
    int dy[4] = {0, 0, 1, -1};
    bool vis[51][51] = {0};

public:
    int bfs(vector<vector<int>>& f, int bx, int by, int ex, int ey) {
        if (bx == ex && by == ey)
            return 0;
        queue<pair<int, int>> q;
        memset(vis, 0, sizeof vis);
        q.push({bx,by});
        vis[bx][by] = true;

        int step = 0;
        while (q.size()) {
            step++;
            int sz = q.size();
            while (sz--) {
                auto [a, b] = q.front();
                q.pop();
                for (int i = 0; i < 4; i++) {
                    for (int j = 0; j < 4; j++) {
                        int x = a + dx[i];
                        int y = b + dy[i];
                        if (x >= 0 && x < m && y >= 0 && y < n && f[x][y] &&
                            !vis[x][y]) {
                            if (x == ex && y == ey) {
                               return step;
                            } q.push({x, y});
                                vis[x][y] = true;
                        }
                    }
                }
            }
        }
        return -1;
    }
```

```
    int cutOffTree(vector<vector<int>>& f) {
        m = f.size();
        n = f[0].size();

        // 排序
        vector<pair<int, int>> trees;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (f[i][j] > 1)
                    trees.push_back({i, j});
            }
        }
        sort(trees.begin(),trees.end(),
             [&](const pair<int, int>& p1, const pair<int, int>& p2) {
                 return f[p1.first][p1.second] < f[p2.first][p2.second];
             });

        // 砍树
        int bx = 0;
        int by = 0;
        int ret = 0;
        for (auto &[a,b] : trees) {
            int step = bfs(f, bx, by, a, b);
            if (step == -1)
                return -1;
            ret += step;
            bx = a;
            by = b;
        }
        return ret;
    }
};
```