第七周作业讲解
综合动态规划

Robert_JYH
洛谷网校

动态规划(DP)

- 对问题各状态维度进行分阶段、有顺序、无重复、决策性的遍历求解
- 三要素：状态、阶段、决策
- 三个基本条件：子问题重叠性、无后效性、最优子结构性质

动态规划(DP)

1.状态类型/结构：

序列、背包、区间、坐标、环形……

2.转移方式：递推、记忆化搜索

3.优化：

(1)预处理、前缀和

(2)状态量：跳过无用状态、改进状态表示、利用约束关系

(3)空间：滚动数组

T503417 【第七周】判断是否要使用 long long

在比赛中，根据数据范围，分析清楚变量的取值范围，是非常重要的。int 类型变量与 int 类型变量相乘，往往可能超出 int 类型可以表示的取值范围。

现在，给出两个 int 类型变量 x, y及其取值范围，请问 $x \times y$ 的值是否可能超过 int 类型可以表示的范围？

T503417 【第七周】判断是否要使用 long long

本题用于提醒大家，int 类型变量与 int 类型变量相乘，可能超出 int 类型可以表示的取值范围。

在实现上，直接用long long读入，判断两个变量的边界相乘是否会超出int范围即可。

要注意的是，不仅要判断左端点与左端点，右端点与右端点相乘结果，还要判断左端点与右端点，右端点与左端点相乘结果。

int 范围： [-2147483648, 2147483647]

T503417 【第七周】判断是否要使用 long long

```
long long a, b, c, d;
int main()
{
    cin >> a >> b >> c >> d;
    if (a * c < INT_MIN || a * c > INT_MAX || b * d <
INT_MIN || b * d > INT_MAX || a * d < INT_MIN || a * d >
INT_MAX || b * c < INT_MIN || b * c > INT_MAX)
    {
        cout << "long long int";
    }
    else
    {
        cout << "int";
    }
    return 0;
}
```

T503485 【第七周】游戏得分

- 一个游戏已经持续了 N 天，每天的得分记为 S_i，其中某些天的得分可能为负。
- 现在，为了在名片中展示，你需要找到一段连续的天数，使得这些天的得分总和最大。
- 请编写一个程序，计算出最大连续得分总和。
- $1 \leq N \leq 100{,}000$
- $-1{,}000 \leq S_i \leq 1{,}000$

- 例如，输入为：2 -4 3 -1 2 -4 3
- 答案为 4

T503485 【第七周】游戏得分

- 本题是经典的最大子段和问题。
- 设$f[i]$表示以第i个数结尾的最大子段和（最大连续得分总和）。
- 当$f[i-1] \leq 0$时，延续之前的子段不优，$f[i] = num[i]$
- 当$f[i-1] > 0$时，延续之前的子段更优，$f[i] = num[i] + f[i-1]$
- 综合有，$f[i] = num[i] + \max(f[i-1], 0)$
- $Ans = \max_{i=1}^{n} f[i]$

T503485 【第七周】游戏得分

```
scanf("%d", &n);
for(int i = 1; i <= n; i++) {
    scanf("%d", &num[i]);
    f[i] = num[i] + max(f[i-1], 0); //DP转移
    ans = max(f[i], ans);
}
printf("%d\n", ans);
```

T503519 【第七周】搬运货物

你正在搬运货物，将货物从仓库的地面运送到顶楼。这个仓库有 n 层，每一层的高度为 h_i。

你的运输工具可以每次向上移动一层或两层，但每次移动后工具会消耗大量能量，因此经过一层的储能才能再次使用。储能过程中，你需要手动搬运货物。

现在，你需要计算从地面到仓库顶层你需要手动搬运的最小总高度。

- $1 \leq n \leq 10^6$
- $1 \leq h_i \leq 100$

- 例如，输入为：5 2 4 5 3 2
- 答案为5

T503519 【第七周】搬运货物

设$f[i]$表示到达第 i 层时手动搬运的最小总高度。

我们可以枚举上一次手动搬运货物是在哪层。

- 若是当前层，则有$f[i]=f[i-1]+num[i]$
- 若是上一层，则有$f[i]=f[i-2]+num[i-1]$
- 若是上上层，则有$f[i]=f[i-3]+num[i-2]$

取三种情况中较小值即可。

T503519 【第七周】搬运货物

```
if(i <= 2){
    f[i] = 0;
    continue;
}
f[i] = f[i - 1] + num[i]; //第 i 层手动搬运
if(i - 2 >= 0) //第 i - 1  层手动搬运
    f[i] = min(f[i], f[i - 2] + num[i - 1]);
if(i - 3 >= 0) //第 i - 2  层手动搬运
    f[i] = min(f[i], f[i - 3] + num[i - 2]);
```

T503419 【第七周】算数游戏

- 你正在进行一个算数游戏，有 N个整数。
- 你可以任意选择若干个数字（不能选择 0 个），并在选择的每个整数（包括第一个数）前面选择放加号（+）或减号（-）。
- 你的任务是计算所有可能的选数与选加减号方法，求能够得到多少种绝对值不同的结果（不包含 0）。
- $1 \leq n \leq 100$
- $\sum A_i \leq 10^5$

T503419 【第七周】算数游戏

我们可以把数字看作物品的重量，将其转化成01背包的可行性问题（类似例题中的货币系统）。

设$f[i][j]$表示考虑了前i种数字时，是否能够算出数字j。

转移时需要考虑以下几种情况：

- 不选该数字/在该数字前放加号/在该数字前放减号。

另外，需要特判只选择了该数字的情况。

即当条件① $j == num[i]$ ② $f[i-1][j]$

③ $f[i-1][j+num[i]]$ ④ $f[i-1][abs(j-num[i])]$

有一个为真时，$f[i][j] = true$

T503419 【第七周】算数游戏

```
for(int i = 1; i <= n; i++){
    for(int j = sum; j ; j--)
        //考虑所有情况
        f[i][j] = (j == num[i]) | f[i - 1][j]
            | f[i - 1][j + num[i]] | f[i - 1][abs(j - num[i])];
}
for(int i = 1; i <= sum; i++)
    ans += f[n][i];
```

复习：背包其它设问

- 求方案数：

1.求方案总数：初值$f[]=\{1,0,0,\dots\}$，将max/min 改为求和

2.求最优方案数：两个数组，一个记录最大价值，一个记录最大价值的最优方案数

3.可行性：改为bool数组，由之前的可行性推现在方案的可行性

- 输出方案：

1.最优方案：记录下每个状态的最优值是由状态转移方程的哪一项推出来的，换句话说，记录下它是由哪一个策略推出来的。便可根据这条策略找到上一个状态，从上一个状态接着向前推即可。

2.第k优方案：对于每个状态维护一个大小为k的解的队列，每次转移将两个队列合并后的前k优解记录。

T503523 【第七周】分校

- luogu 在河两岸各建立了 n 个分校，东西两岸的学校间两两间成为了互助学校。每对互助学校都向总校申请要在两校间建立岸上通道。
- 为了避免发生事故，学校之间的通道不能交叉。现在请你求出能建立通道的分校对数最大值。
- $1 \leq n \leq 200000$

T503523 【第七周】分校

- 考虑按照西岸坐标由小到大对城市排序，此时选择出的城市东岸坐标必须单调上升，否则会出现交叉。
- 所以按照西岸坐标由小到大对城市排序后，问题即转化为最长上升子序列问题。
- 本题需要使用时间复杂度为 $O(n \log n)$ 的方法求最长上升子序列。

复习：$O(n \log n)$求最长上升子序列

- 在一个给定的序列中，找到一个最长的子序列，使得这个子序列元素的数值依次递增。

- 对于每个数单独考虑，维护一个上升子序列，如果当前数比序列尾更大，直接将其插入尾部，序列长度+1；
- 否则用它替换掉序列中第一个比它大的数（保持单调性）；
- 最后得到的序列长度即为最长上升子序列长度。

T503523 【第七周】分校

```
    for (int i = 1; i <= n; i++)
        scanf("%d%d", &a[i].l, &a[i].r);
    sort(a + 1, a + 1 + n, cmp);
    for (int i = 1; i <= n; i++)
    {
        if (a[i].r > b[len])
            b[++len] = a[i].r;
        else
        {
            int pos = lower_bound(b + 1, b + 1 + len,
a[i].r) - b;
            b[pos] = a[i].r;
        }
    }
    printf("%d\n", len);
```

T503517 【第七周】公交换乘

- 在一个城市中，有 n 个公交车站。
- 乘客可以在任意一个公交车站上车，并在编号大于上车车站的任何一个公交车站下车。 车站 i 到车站 j 之间的车费为 $cost(i,j)$ 。
- 现在，你需要计算从车站 1 到车站 n 所需的最少车费。
- $1 \leq n \leq 200$

T503517 【第七周】公交换乘

- 设$f[i]$表示到达车站i的最小费用，我们只需要枚举从哪个车站坐到车站i即可。
- 即有$f[i] = \min_{j<i}(f[i], f[j] + cost[j][i])$

T503517 【第七周】公交换乘

```
for(int i = 2; i <= n ; i++){
    f[i] = INT_MAX;
    for(int j = 1 ; j < i; j++) //枚举上一个乘坐的车站
        f[i] = min(f[i], f[j] + cost[j][i]);
}
```

P7774 [COCI2009-2010#2] KUTEVI

给定 N 个角（第 i 个角记作 a_i），作为初始角，另给定 M 个角（第 i 个角记作 b_i），作为目标角。

请求出对于每个 b_i，它是否能被若干个 a_i 之间的加、减运算得到。

注意同一个 a_i 可以用多次，也可以不用。

- $1 \leq N, M \leq 10, 0 < a_i, b_i < 360$

P7774 [COCI2009-2010#2] KUTEVI

与T503419算数游戏相似，是动态规划中的可行性问题。

对于第i个角a_i，我们可以遍历它的倍数，并记录所有可能的值，直到出现循环。

随后，我们可以应用与T503419算数游戏相同的背包的转移方式，分别考虑使用加/减运算可以转移到哪里。

由于有360种可能的角度，故总复杂度为$O(360^2n)$。

P7774 [COCI2009-2010#2] KUTEVI

```
for (int i = 0; i < n; ++i)
{
    memset(flag, 0, sizeof(flag));
    int j = 0;
    while (!flag[j]) //枚举所有情况
    {
        flag[j] = 1;
        j = (j + rad[i]) % MOD; //MOD=360，将角度换算到360内
    }
    for (int j = 0; j < MOD; ++j)
    {
        if (dp[j] == 0)  continue;
        for (int k = 0; k < MOD; ++k)
        {
            dp[(j + k) % MOD] |= flag[k];
            dp[(j - k + MOD) % MOD] |= flag[k];
        }
    }
}
```

P1091 [NOIP2004 提高组] 合唱队形

n 位同学站成一排，音乐老师要请其中的 $n-k$位同学出列，使得剩下的 k位同学排成合唱队形。

合唱队形是指这样的一种队形：设 k 位同学他们的身高满足 $t_1 < \cdots < t_i > t_{i+1} > \cdots > t_k$。

你的任务是，已知所有 n 位同学的身高，计算最少需要几位同学出列，可以使得剩下的同学排成合唱队形。

- $n \leq 100$

P1091 [NOIP2004 提高组] 合唱队形

- 题目要求：求一个最长子序列$[l,r]$使得存在一个位置k满足l 到 k 单调增，k 到 r 单调减。
- 分别求到某位置的最长上升子序列和从某位置开始的最长下降子序列，两者长度相加取最大即可。
- 具体地，设$b[i]$为以 $a[i]$ 结尾的最长上升子序列的长度，$c[i]$以 $a[i]$ 结尾的最长下降子序列的长度。
- 对于$b[i]$，$b[i]=\max_{1\le j<i \text{ and } a[i]>a[j]}(b[j]+1)$
- 对于$c[i]$，序列逆序后的最长上升子序列就是正序时的最长下降子序列，于是将序列逆序，做相同转移即可。
- 时间复杂度 $O(n^2)$ 。

复习：最长上升子序列

- 在一个给定的序列中，找到一个最长的子序列，使得这个子序列元素的数值依次递增。

- 设计状态（描述当前遇到的问题，涉及什么状态就放什么）：

  $f[x]$ 表示以 $a[x]$ 结尾的最长上升子序列的长度

- 决策：转移的条件（思考能从哪里转移来）：

  $1 \le j < i$ 且 $a[i] > a[j]$

- 转移（转移时会有什么影响）：

  最长上升子序列的长度相较之前加 1

- $f[i] = \max_{1 \le j < i \text{ and } a[i] > a[j]}(f[j] + 1)$

P1091 [NOIP2004 提高组] 合唱队形

```
// 正序
for (int i = 1; i <= n; i++)
{
    b[i] = 1;
    for (int j = 1; j <= i - 1; j++)
        if (a[i] > a[j])
            b[i] = max(b[i], b[j] + 1);
}
// 逆序
for (int i = n; i >= 1; i--)
{
    c[i] = 1;
    for (int j = i + 1; j <= n; j++)
        if (a[j] < a[i])
            c[i] = max(c[i], c[j] + 1);
}
for (int i = 1; i <= n; i++)
    ans = max(ans, b[i] + c[i] - 1);
cout << n - ans << endl;
```

P2066 机器分配

- 总公司拥有高效设备 M 台，准备分给下属的 N 个分公司。各分公司若获得这些设备，可以为国家提供一定的盈利。
- 问：如何分配这 M 台设备才能使国家得到的盈利最大？求出最大盈利值。
- $M \leq 15$，$N \leq 10$。
- 最大盈利值相同时，要求编号小的公司分得设备尽可能少。

P2066 机器分配

- 先不考虑输出方案的要求。
- 设 $f[i][j]$表示前i家公司分配j台机器的最大利润。
- 我们可以枚举第i家公司分走了k台机器。
- 即有：$f[i][j]=\max\limits_{0\le k\le j} f[i-1][j-k]$

- 接下来我们考虑如何使得最大盈利值相同时，要求编号小的公司分得设备尽可能少。
- 我们可以用一个数据结构保存每家公司分走的机器数。当存在两种方案最大利润相同时，我们依次比较两种方案每家公司分走的机器数，保留编号小的公司分得设备更少的方案。
- 这里的数据结构用数组/字符串/队列等均可，参考代码里为了比较方便，使用了字符串。

P2066 机器分配

```
struct node
{
    int val;//最大利润
    string s;//每一位表示每家公司分得机器数
    bool operator<(node b) const
    {
        //自定义比较规则
        return val < b.val || val == b.val && s > b.s;
    }
    node operator+(int num)
    {
        return node{val + num, s};
    }
    node operator+(char t)
    {
        return node{val, s + t};
    }
} f[N][N];
```

P2066 机器分配

```
for (int i = 1; i <= n; i++)
    for (int j = 0; j <= m; j++)
    {
        f[i][j] = f[i - 1][j] + char(0);
        for (int k = 1; k <= j; k++)
            if (f[i][j] < f[i - 1][j - k] + a[i][k] + char(k))
                f[i][j] = f[i - 1][j - k] + a[i][k] + char(k);
    }
printf("%d\n", f[n][m].val);
for (int i = 1; i <= n; i++)
    printf("%d %d\n", i, f[n][m].s[i - 1]);
```

P4141 消失之物

ftiasch 有 N 个物品，体积分别是 $W_1, W_2, \ldots, W_N$。

由于她的疏忽，第 i 个物品丢失了。

“要使用剩下的 $N - 1$ 物品装满容积为 x 的背包，有几种方法呢？”

-- 这是经典的问题了。

她把答案记为$Count(i, x)$，想要得到所有$1 \leq i \leq N, 1 \leq x \leq M$的 $Count(i, x)$ 表格。（输出末位数字）

- $1 \leq n, m \leq 2000$

P4141 消失之物

暴力做法是跑n次背包，时间复杂度$O(n^2m)$。考虑如何优化。

我们只需要使用01背包预处理出用 全部N件 物品装满背包的方案数，然后$O(n)$地枚举不选择哪个物品，然后将其影响$O(m)$地强制消去即可，时间复杂度$O(nm)$。

具体地，设$dp[j]$表示全部N件物品装满容量j的方案数，$ans[j]$表示$N-1$件物品装满容量j的方案数。

则有

$$dp[j] = dp[j] + dp[j - w[i]]$$

$$ans[j] = ans[j] - ans[j - w[i]]$$

P4141 消失之物

这种先忽略某限制预处理出DP结果，然后考虑限制撤销影响的做法，是DP的一种常见优化方式。

由于我们只需要输出末位数字，因此代码中要注意对10取模。

P4141 消失之物

```
dp[0] = 1;
for (int i = 1; i <= n; i++)
    for (int j = m; j >= w[i]; j--)
        dp[j] = (dp[j] + dp[j - w[i]]) % 10;

for (int i = 1; i <= n; i++, putchar('\n'))
{
    for (int j = 0; j <= m; j++)
        ans[j] = dp[j];
    for (int j = w[i]; j <= m; j++)
        ans[j] = (ans[j] - ans[j - w[i]] + 10) % 10;
    for (int j = 1; j <= m; j++)
        putchar(ans[j] + '0');
}
```

P1544 三倍经验

现在你在数字金字塔的顶部（第一行），你希望走到金字塔的底部（第 n 行），每一步你只能走向当前所在位置的左下方的数字或者右下方的数字。同时作为一个强大的小朋友，你可以选择金字塔中的不多于 k 个数字让他们成为原来的 3 倍。

你会收集你路上经过的所有位置上的数字，最后的得分即为收集的数字之和，求最大得分。

- $1 \leq n \leq 100, 0 \leq k \leq \frac{n(n+1)}{2}$

P1544 三倍经验

- 设$dp[i][j][l]$表示走到第 i 行第 j 个点，选择了 l 个数字变成原来的三倍时 的最大得分。
- 若本次不使用翻三倍，则与数字金字塔转移完全相同：
  $f[i][j][l] = \max(f[i-1][j][l], f[i-1][j-1][l]) + a[i][j]$
- 若本次使用翻三倍，则有：
  $f[i][j][l]$
  $= \max(f[i-1][j][l-1], f[i-1][j-1][l-1]) + a[i][j] \times 3$
- 取两种情况较大值即可。
- 本题输入不超过int范围，但加法和乘法会导致结果超出int范围，注意使用$long\ long$。

P1544 三倍经验

```
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= i; j++)
    {
        for (int l = 0; l <= min(k, i); l++)
        {
            f[i][j][l] = max(f[i - 1][j][l], f[i - 1][j - 1][l]) + a[i][j];
            if (l != 0)
                f[i][j][l] = max(f[i][j][l],
                    max(f[i - 1][j][l - 1], f[i - 1][j - 1][l - 1]) + a[i][j] * 3);
        }
    }
```

P1063 [NOIP2006 提高组] 能量项链

- 在项链上有N颗能量珠。能量珠是一颗有头标记与尾标记的珠子，这些标记对应着某个正整数。对于相邻的两颗珠子，前一颗珠子的尾标记一定等于后一颗珠子的头标记。这两颗珠子能聚合成一颗珠子。如果前一颗能量珠的头标记为a，尾标记为b，后一颗能量珠的头标记为b，尾标记为c，则聚合后释放的能量为$a \times b \times c$，新产生的珠子的头标记为a，尾标记为c。
- 通过聚合得到能量，直到项链上只剩下一颗珠子为止。显然，不同的聚合顺序得到的总能量是不同的，请你设计一个聚合顺序，使一串项链释放出的总能量最大。
- $N \leq 100$

P1063 [NOIP2006 提高组] 能量项链

- 有决策的区间DP一般为三层循环：第一层表示区间长度，第二层枚举起点并根据第一层区间长度算出区间终点，(或者前两层为枚举左右端点），第三层便在当前区间内枚举决策（有时候要在最外层加一层状态）
- 设$f[l][r]$表示一段区间内释放出的最大总能量
- $f[i][j]=\max(f[i][j],f[i][k]+f[k+1][j]+a[i]*a[k+1]*a[j+1])$
- 本题由于是环，需破环为链，可以开两倍大的数组，即$a[i]=a[i+n]$

P1063 [NOIP2006 提高组] 能量项链

```
    for (int j = 2; j <= 2 * n - 1; j++)
        for (int i = j - 1; i > 0 && j - i < n; i--)
            for (int k = i; k < j; k++)
                f[i][j] = max(f[i][j], f[i][k] + f[k + 1][j]
+ a[i] * a[k + 1] * a[j + 1]);
    for (int i = 1; i <= n; i++)
        maxx = max(maxx, f[i][i + n - 1]);
    printf("%d\n", maxx);
```

复习：区间DP

- 区间动态规划问题的基本思想是，对于每个区间，他们的最优值都是由几段更小区间的最优值得到，于是可以分而治之。
- 状态：一般设定$f[i][j]$表示$[i,j]$这个区间的最优值。
- 决策与转移：大区间是由小区间推导出来的，于是需要枚举分界点转移，并附加代价。

P1541 [NOIP2010 提高组] 乌龟棋

- 乌龟棋的棋盘是一行 N 个格子，每个格子上一个分数（非负整数）。游戏要求玩家控制一个乌龟棋子，从起点第 1 格 出发走到终点第 N 格。
- 乌龟棋中有 M 张卡片，卡片有四种花色，分别对应 1,2,3,4 四个数字。每次使用一张卡片，棋子就可以向前移动这张卡片所对应数字的格数。
- 玩家在本次游戏中的得分，就是移动乌龟棋从第一格到最后一格的过程中，经过的所有的格子上的分值的和，你的任务是要找到一种卡片使用顺序使得最终游戏得分最多。数据保证到达终点时刚好用光 M 张爬行卡片。
- $N \leq 350, M \leq 120,$ 每种卡片张数 ≤ 40

P1541 [NOIP2010 提高组] 乌龟棋

- $f[i][a1][a2][a3][a4]$表示玩到第 i 格，用了 $a1$ 张 1,$a2$ 张 2,$a3$ 张 3,$a4$ 张 4 的最大得分。
- 考虑这次用了哪张牌，从$f[i-1][a1-1][a2][a3][a4], f[i-2][a1][a2-1][a3][a4],$

  $f[i-3][a1][a2][a3-1][a4], f[i-4][a1][a2][a3][a4-1]$转移 。
- 我们发现有约束关系，$i = a1 + 2a2 + 3a3 + 4a4 + 1$，直接把第一维扔掉即可（时间优化）。
- 我们还可以滚动数组把剩下四维的其中一维滚动掉（空间优化）。

P1541 [NOIP2010 提高组] 乌龟棋

```
for (int i1 = 0; i1 <= c[1]; i1++)
    for (int i2 = 0; i2 <= c[2]; i2++)
        for (int i3 = 0; i3 <= c[3]; i3++)
            for (int i4 = 0; i4 <= c[4]; i4++) {
                int dis = 1 + i1 + i2 * 2 + i3 * 3 + i4 * 4;
                if (i1 != 0)
                    f[i2][i3][i4] = max(f[i2][i3][i4],
                        f[i2][i3][i4] + a[dis]);
                if (i2 != 0)
                    f[i2][i3][i4] = max(f[i2][i3][i4],
                        f[i2 - 1][i3][i4] + a[dis]);
                if (i3 != 0)
                    f[i2][i3][i4] = max(f[i2][i3][i4],
                        f[i2][i3 - 1][i4] + a[dis]);
                if (i4 != 0)
                    f[i2][i3][i4] = max(f[i2][i3][i4],
                        f[i2][i3][i4 - 1] + a[dis]);
            }
```

P3983 赛斯石（赛后强化版）

卖家想算出这些赛斯石经过某种合并方式来获得的最大收益。赛斯石的重量只能是整数赛斯重量，不同赛斯重量的赛斯石的价格也是不一样的。

然而目前有一个问题，市场在真程大殿附近（真程海洋中心位置），卖家需要租船送赛斯石过去（不考虑卖家自己租船过去的费用），目前有十种船可以租，载重量从 $1\ si$到 $10\ si$，每艘船的租价如下表所示：

载重(si)	1	2	3	4	5	6	7	8	9	10
价格(元)	1	3	5	7	9	10	11	14	15	17

商家将赛斯石运过来之后，只能按照之前合并好的卖，全部能卖出去。现在卖家他要计算总盈利（设总盈利=赛斯石的总收益-租船所需总费用），请你算出一种最佳方案，以获得最大总盈利。

对于所有输入数据，均在区间$(0,100000)$中，并且为整数。

P3983 赛斯石（赛后强化版）

我们可以将其看作是一个背包套一个背包。

对于每种船，我们可以用完全背包预处理出单艘船的最大收益，此时背包的容量是船大小，物品是石头，代价为石头的质量，价值为其对应市场价。

对于整个运输过程，我们可以接着用完全背包求出总的最大收益，此时背包的容量是石头总数，物品是船，每艘船的代价为能承载石头的最大质量，价值为前面预处理出的最大收益。此时所求即为答案。

P3983 赛斯石（赛后强化版）

```
// 完全背包预处理出单艘船的最大收益
for (int i = 1; i <= tot; i++)
    for (int j = i; j <= tot; j++)
        boat[j] = max(boat[j], boat[j - i] + a[i]);
for (int i = 1; i <= tot; i++)
    boat[i] -= val[i];
// 用完全背包求出总的最大收益
for (int i = 1; i <= tot; i++)
    for (int j = i; j <= n; j++)
        f[j] = max(f[j], f[j - i] + boat[i]);
for (int i = 1; i <= n; i++)
    ans = max(ans, f[i]);
```