

第九章 DP 专题

一、区间 DP

1、课堂练习

区间 dp 就是在区间上进行动态规划，求解一段区间上的最优解。主要是通过合并小区间的最优解进而得出整个大区间上最优解的 dp 算法。

既然让我求解在一个区间上的最优解，那么我把这个区间分割成一个个小区间，求解每个小区间的最优解，再合并小区间得到大区间即可。所以在代码实现上，我可以枚举区间长度 len 为每次分割成的小区间长度（由短到长不断合并），内层枚举该长度下可以的起点，自然终点也就明了了。然后在这个起点终点之间枚举分割点，求解这段小区间在某个分割点下的最优解。

区间 dp 的模板：

```
for(int len = 1;len<=n;len++){//枚举长度
    for(int j = 1;j+len-1<=n;j++){//枚举起点，ends<=n
        int ends = j+len - 1;
        for(int i = j;i<ends;i++){//枚举分割点，更新小区间最优解
            dp[j][ends] = min(dp[j][ends],dp[j][i]+dp[i+1][ends]+something);
        }
    }
}
```

2057: 【入门】合并石子

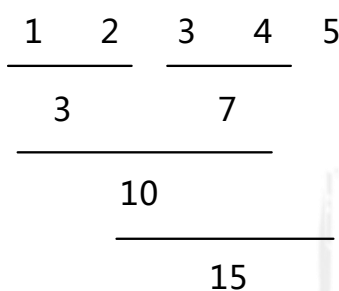
时间复杂度: $O(n^3)$

状态转移方程:

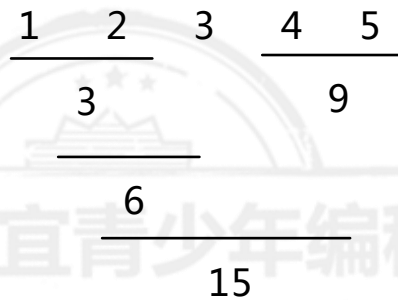
$f[l][r]=\min(f[l][r],f[l][k]+f[k+1][r]+s[r]-s[l-1]);$

假设有 5 个数:

1 2 3 4 5, 合并方案有:

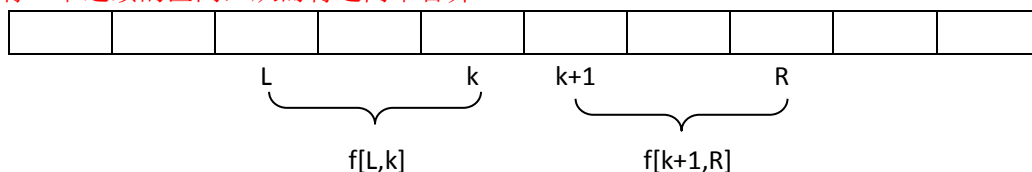


得分 = 3 + 7 + 10 + 15 = 35



得分 = 3 + 6 + 9 + 15 = 33

注意：每次只能合并相邻的两堆，因此在最后一步合并，一定是左侧有一堆连续的区间，右侧有一堆连续的区间，从而将这两堆合并！



1. 设 $f[L,R]$: 为从位置 L 到位置 R , 进行数字合并的最小得分;

2. 设 k 为 $[L,R]$ 的分割点;

3. 枚举所有的分割点, 得到:

$f[L,k]$: 合并 $[L,k]$ 的最小代价

$f[k+1,R]$: 合并 $[k+1,R]$ 的最小代价

$s[i]$: 代表了前缀和

再将左右合并: 左右合并相当于是得到 $[L,R]$ 的区间和: $\Sigma_R^L = S[R] - S[L - 1]$

$f[l][r] = \min(f[l][r], f[l][k] + f[k+1][r] + s[r] - s[l-1]);$

边界条件: $f[i][i] = 0$, 自己这个数字合并的代价为 0, 除此以外, $f[i][j] = \infty$

```
#include <bits/stdc++.h>
using namespace std;

/*
1. 每次只能选相邻的 2 堆石子合并成新的一堆
2. 并将新的一堆石子数记为该次合并的得分
3. 求将 N 堆石子合并成一堆的最小得分
*/
const int N = 110;
int f[N][N]; // 表示 [l, r] 合并区间内的数对应的最小得分
int s[N]; // 表示前缀和
int n;

int main(){
    cin >> n;
    // 边界条件
    memset(f, 0x3f, sizeof(f)); // 默认初始化为无穷大

    for(int i = 1; i <= n; i++){
        cin >> s[i];
        s[i] += s[i-1]; // 求出前缀和
        f[i][i] = 0; // 合并自己的默认得分为 0
    }

    // 穷举所有可能的长度
    for(int len = 2; len <= n; len++){
        // 穷举区间可能的出发点
        for(int l = 1; l + len - 1 <= n; l++){
            int r = l + len - 1; // 区间的终点
            // 穷举分割点
            for(int k = l; k < r; k++){
                f[l][r] = min(f[l][r], f[l][k] + f[k+1][r] + s[r] - s[l-1]);
            }
        }
    }

    // for(int i = 1; i <= n; i++){
    //     for(int j = 1; j <= n; j++){
    //         // cout << f[i][j] << " ";
    //         if(f[i][j] == 0x3f3f3f3f) cout << "-- ";
    //         else cout << f[i][j] << " ";
    //     }
    //     cout << endl;
    // }

    cout << f[1][n];
    return 0;
}
```

2058: 【入门】石子合并（环形）

	4	5	9	4	4	5	9	4	
0	1	2	3	4	5	6	7	8	

思路：本题将数组复制出来，然后求出从 $[1, 2*n]$ 范围内，合并 n 个数的最代价。再枚举所有可能的起始点，求合并的最小和最大代价。

```
#include <bits/stdc++.h>
using namespace std;

/*
将数据规模翻倍，将 a 数组完整拷贝一遍
求区间最长长度为 n 的情况下，合并得到的最优解
再从每个可能的起点出发，打擂台求最小、最大值
*/
const int N = 410;
int a[N*2], s[N*2]; // s: 前缀和
int mi[N*2][N*2], ma[N*2][N*2];
int n;

int main(){
    cin >> n;
    // 初始化
    memset(mi, 0x3f, sizeof(mi));
    memset(ma, -0x3f, sizeof(ma));
    for(int i = 1; i <= n; i++){
        cin >> a[i];
        a[i+n] = a[i]; // 数组完整的赋值一遍
    }

    // 求前缀和
    for(int i = 1; i <= n * 2; i++){
        s[i] = s[i-1] + a[i]; // 前缀和
        mi[i][i] = 0;
        ma[i][i] = 0; // 初始化
    }

    // 区间 dp 求解
    // 穷举所有可能的区间长度
    for(int len = 2; len <= n; len++){
        // 穷举区间的左端点
        for(int i = 1; i + len - 1 <= n * 2; i++){
            // 右端点
            int j = i + len - 1;
            // 穷举分割点
            for(int k = i; k < j; k++){
                mi[i][j] = min(mi[i][j], mi[i][k] + mi[k+1][j] + s[j] - s[i-1]);
                ma[i][j] = max(ma[i][j], ma[i][k] + ma[k+1][j] + s[j] - s[i-1]);
            }
        }
    }

    // 穷举所有可能的出发点，打擂台求最小、最大
    int minn = INT_MAX, maxn = INT_MIN;
    for(int i = 1; i <= n; i++){
        minn = min(minn, mi[i][i+n-1]);
        maxn = max(maxn, ma[i][i+n-1]);
    }

    cout << minn << endl << maxn;
    return 0;
}
```

2213: 【基础】相同数的合并-USACO16OPEN

相邻两个区间的值相等才能合并。

状态转移方程:

$$f[l][r] = \max(f[l][r], f[l][k] + 1);$$

边界:

$$f[i][i] = a[i];$$

$$f[i][j] = -\infty$$

```
#include <bits/stdc++.h>
using namespace std;
```

/*

1. 合并相邻两个相同的数

2. 注意合并后的数值并非加倍而是+1

求: 序列中出现的最大数字的值是多少

合并条件: $f[L, K] == f[K+1][R]$

$$f[L, R] = \max(f[L, R], f[L, K] + 1)$$

边界条件:

$$f[i][j] = -\infty$$

$$f[i][i] = a[i] \text{ 自己这个数合并得到的结果就是自己}$$

*/

```
const int N = 260;
```

```
int a[N], f[N][N];
```

```
int n, ans = 0; // ans 表示最大数
```

```
int main(){
```

```
    cin >> n;
```

```
    // 初始化
```

```
    memset(f, -0x3f, sizeof(f));
```

```
    for(int i = 1; i <= n; i++){
```

```
        cin >> a[i];
```

```
        f[i][i] = a[i];
```

```
    }
```

```
    // 枚举可能的长度
```

```
    for(int len = 2; len <= n; len++){
```

```
        // 枚举所有区间可能的起点
```

```
        for(int i = 1; i + len - 1 <= n; i++){
```

```
            int j = i + len - 1; // 区间终点
```

```
            // 枚举所有的分割点
```

```
            for(int k = i; k < j; k++){
```

```
                // 合并条件
```

```
                if(f[i][k] == f[k+1][j]) {
```

```
                    f[i][j] = max(f[i][j], f[i][k] + 1);
```

```
                    ans = max(f[i][j], ans);
```

```
                }
```

```
            }
```

```
        }
```

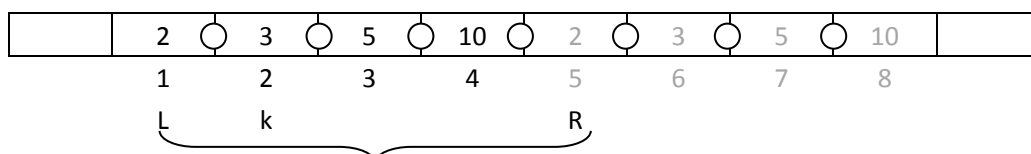
```
    }
```

```
    cout << ans;
```

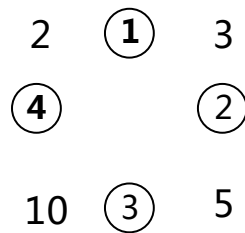
```
    return 0;
```

```
}
```

2059: 【基础】能量项链



k -> (L,R)



第 1 颗和第 4 颗珠子合并得到的能量 = $10 * 2 * 3$

最佳合并 = $((4 \oplus 1) \oplus 2) \oplus 3$

能量 = $10 * 2 * 3 + 10 * 3 * 5 + 10 * 5 * 10$

1. 设 $f[L,R]$: 为从位置 L 到位置 R, 进行数字合并的最大能量;

2. 设 k 为 $[L,R]$ 的分割点;

3. 枚举所有的分割点, 得到:

$f[L,k]$: 合并 $[L,k]$ 的最小代价

$f[k,R]$: 合并 $[k,R]$ 的最小代价

$f[l][r] = \min(f[l][r], f[l][k] + f[k][r] + a[l] * a[k] * a[r]);$

```
#include <bits/stdc++.h>
using namespace std;
const int N = 210;
int n;
int a[N]; // a 存储珠子的头标记
// dp[l][r]: 合并 l~r 得到的最大值
int dp[N][N];

int main() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        a[i+n] = a[i]; // 将区间复制
    }

    // 状态计算
    // 阶段: 穷举区间长度
    for (int len = 3; len <= n + 1; len++) {
        // 状态: 穷举区间的起点
        for (int l = 1; l + len - 1 <= 2 * n; l++) {
            // 区间终点
            int r = l + len - 1;
            // 决策: 穷举分割点
            for (int k = l+1; k < r; k++) {
                dp[l][r] = max(dp[l][r], dp[l][k] + dp[k][r] + a[l] * a[k] * a[r]);
            }
        }
    }

    // 求结果
    int r = 0;
    for (int i = 1; i <= n; i++) {
        r = max(r, dp[i][i+n]); // dp[1,n+1] dp[2,n+2]... dp[n,2*n]
    }
    cout << r;
    return 0;
}
```

}

