# Solution 5

动态规划的**SRTBOT**分析法指的是：

- $\textbf{S(ubproblem)}$: The problem can be broken down into smaller, overlapping subproblems.
- $\textbf{R(elation)}$: There is a relation between the larger problem and its subproblems.
- $\textbf{T(opological order)}$: You solve the problem in a bottom-up manner.
- $\textbf{B(ase case)}$: You need to have a base case that can be used to terminate the recursion.
- $\textbf{O(riginal problem)}$: The original problem can be solved by solving all subproblems.
- $\textbf{T(ime complexity)}$: The time complexity of the algorithm.

本次作业中有关动态规划的题目都建议使用**SRTBOT**方法进行设计和分析。

1、上课时介绍了LIS（最长上升子序列）的$O(n^2)$解法。事实上，LIS问题有很多种方法都可以做到$O(n\log n)$，请设计一种$O(n\log n)$的LIS算法。

方法一：

因为最长上升子序列只和元素的相对大小有关，和元素的绝对大小无关，所以先用$O(n\log n)$的时间对序列进行离散化，使得每个元素的大小$\in[1,n]$

Subproblem: $f_{i,j}$表示只考虑前$i$个元素，结尾为$j$的最长上升子序列的长度

Relation: $f_{i,j}\leftarrow\max\{f_{i-1,j},\max\{f_{i-1,k}\}+1\},(k<j)$

Topological order: increasing $i$

Base case: $f_{0,j}=0,\forall\ j$

Original Problem: $\max\{f_{n,j}\}$

Time complexity: $O(n\log n)$

直接做复杂度是$O(n^2)$，但是注意到每次迭代$i++$只会更新一个状态，同时注意到$\max\{f_{i-1,k}\},(k<j)$为前缀最大值并且迭代过程中$f_i$中的值不会减小，因此可以使用**Fenwick Tree**维护$f_i$的前缀最大值，即可做到$O(\log n)$的状态转移。总复杂度$O(n\log n)$。

方法二：

Subproblem: $f_{i,j}$表示只考虑前$i$个元素，长度为$j$的上升子序列结尾的最小值

Relation: $f_{i,j}\leftarrow\min\{f_{i-1,j},f_{i-1,j-1}\},w_i>f_{i-1,j-1}$

$f_{i,j}\leftarrow f_{i-1,j},w_i\le f_{i-1,j-1}$

Topological order: increasing $i$

Base case: $f_{0,0}=-\infty$

$f_{0,j}=\infty,j\ge 1$

Original Problem: $\max\{f_{n,j}\cdot[f_{n,j}\ne\infty]\}$

Time complexity: $O(n\log n)$

注意到$f_i$单调递减，且每次迭代$i++$只会改变一个状态的值，二分找到$> w_i$的$j$最小的$f_{i-1,j}$，用这个状态更新。复杂度$O(n\log n)$

2、Range Partial Sum Query中需要使用到$O(1)$寻找区间$[l,r]$跨过的中线，请给出一个$O(1)$找到这个中线的方法。

先将$n$ pad到2的整次幂。

给定$[l,r]$，它跨越的中线就是$r \ \& \ (\sim (highbit(l \oplus r) - 1))$。其中$highbit(x)$表示$x$的最高位1。表达式的含义是$r$在highbit及之前的位保持不变，之后的位清零。

证明：在highbit之前，$l$和$r$在每次二分都在同一半。因为$l$和$r$在highbit这一位不同，所以在这次二分，$l$和$r$会被分到2边。因此跨越的中线就是$r \ \& \ (\sim (highbit(l \oplus r) - 1))$。

求$highbit(x)$的3种方法：

方法一：

```
int highbit(int x) {
    x |= x >> 1;
    x |= x >> 2;
    x |= x >> 4;
    x |= x >> 8;
    x |= x >> 16;
    return x + 1 >> 1;
}
```

前面几步操作相当于把$x$在最高位1之后的位全部设成1，那么$x+1$就相当于$x$的最高位+1。时间复杂度$O(\log\log x)$。

方法二：

一些接口可以通过硬件支持找到最高位1，如c++里的`__builtint_clz(x)`。时间复杂度$O(1)$。

方法三：

$O(n)$预处理，$O(1)$查询

```
for (int i = 1; i < n; ++i) highbit[i] = highbit[i >> 1] << 1;
```

3、在$\pm 1$RMQ（Range Maximum / Minimum Query）中，序列分为了$\frac{2n}{\log n}$个小块，每个小块大小为$\frac{1}{2}\log n$，请分别说明块内和块间是如何进行预处理的，以及是如何回答每个查询的。你需要保证你的预处理和回答的复杂度分别为$O(n)$和$O(1)$。

块内暴力求出$\log^2 n$种情况，块间使用**st表**或者课上提到的Range Partial Sum Query的$O(n\log n) - O(1)$版本（注意，不能使用$O(n) - O(\alpha(n))$版本，因为插叙复杂度不对）。每个询问如果在块内，直接找到对应的块$O(1)$查询，否则分别在$l$的块内，$r$的块内和$[l,r]$的块间一共查询3次。

4、画出课上提到*Bowling*问题状态转移对应的DAG。

*Bowling*问题：桌上有一排保龄球瓶编号$0,1,\cdots,n-1$，第$i$个保龄球瓶的价值为$w_i$（可能为负数）。可以扔任意次保龄球，每次击中1个或相邻的2个保龄球，得分为击中保龄球价值的乘积，求最大的得分。

$i \to i+1$，$i \to i+2$连边。图略。

5、给定一棵$n$个节点，$n-1$条无向边构成的树，且每条边都有一个整数权值（可能为负数）。请用动态规划的方法求出树的直径是多少，即树上的最长路径是多少。你的算法需要运行在$O(n)$的时间内。

随便选一个点作为根节点进行`dfs`，那么树会变成一棵有向树。记录每个点$u$的子节点中深度最大的2个节点的深度，则直径一定是某一个节点$u$的要么是这2个节点的深度和，要么是最深的节点的深度。

Subproblem: $f_{u,j},(j=0/1)$表示以$u$为根的子树，最深和次深的子节点的深度

Relation: $f_{u,0} \leftarrow \max\{f_{v,0}+w\}, if(u,v,w) \in E$

$f_{u,1} \leftarrow \text{second}\max\{f_{v,0}+w, f_{v,1}+w\}, if(u,v,w) \in E$

Topological order: decreasing depth of $u$

Base case: $f_{u,j}=0, \forall\, u \in \text{leafs of T}$

Original Problem: $\max\{\max\{f_{u,0}\}, \max\{f_{u,0}+f_{u,1}\}\}$

Time complexity: $O(n)$

6、有$n$堆石子围成了一个环，每堆石子分别有一定质量$w_i$。现在要将这$n$堆石子合并为一堆，每次选择将环上相邻的两堆石子合并成一堆石子，新堆的质量为原来两堆石子的质量和，合并的代价等于新堆的质量。我们希望找出一种合并的方法使得总代价最小。请设计一种$O(n^3)$的算法计算这个最小的代价。

一般的石子合并问题使用GarsiaWachs算法可以做到$O(n^2)$，加平衡树优化可以做到$O(n\log n)$，感兴趣的同学可以自行了解。

例：$n=4$，$w=[4,5,9,4]$，首先合并$4,4$代价为$8$，$w\rightarrow[8,5,9]$，然后合并$8,5$代价为$13$，$w\rightarrow[13,9]$，最后合并$13,9$代价为$22$，总代价$8+13+22=43$，可以证明不存在更优的合并方法。

先考虑不是环而是链的情况

Subproblem: $f_{i,j}$合并区间$[l,r]$的最小花费

Relation: $f_{i,j} \leftarrow \min\{f_{i,k}+f_{k+1,j}+\sum_{x=i}^{j} w_x\}$

Topological order: increasing $j-i$

Base case: $f_{i,i}=0$

Original Problem: $f_{0,n-1}$

Time complexity: $O(n^3)$

对于环的情况，考虑把环在某处断开，则会转化成链上的石子合并问题，把$w$复制一份添加到自己后面，枚举断开的位置则有，$ans=\max\{f_{i,i+n-1}\},(0\leq i<n)$。