樹與 DP 進階

waynetuinfor

2017 年 11 月 7 日

1 樹

樹是競賽場上很常出現的結構，樹也有很多特別的性質，大部分在上一堂圖論課就介紹過了，所以這個章節會著重於一些處理樹上操作的技巧。

1.1 樹壓平

樹壓平又稱為樹序列化或樹上尤拉路徑，主要的功能就是將一棵樹壓平成一個序列，然後用資料結構或是對序列使用的演算法來對付樹上的問題。

考慮在 DFS 樹的時候，紀錄下每個節點進入以及離開的時間，會發現這個性質（令 $in(u)$ 為進入的時間、$out(u)$ 為離開的時間）：若 u 是 v 的祖先，則 $in(u) \leq in(v)$ 且 $out(u) \geq out(v)$。

根據上述的性質也可以發現，任一子樹（根為 u）在壓平後形成 $[in(u), out(u)]$ 的區間，因此如果遇到對一個子樹做動態操作的時候，將樹壓平後或許可以使用線段樹之類的資料結構來做到。

1.2 樹壓平 LCA

上一堂圖論課有講過使用倍增法求 LCA 的方法，這裡介紹另一種技巧。

不妨將樹上的節點以 DFS 經過的順序編號，並在每一次走訪點 u 的時候記錄下 u，這樣會得到一個長度為 $2N-1$ 的序列，可以發現任兩點 u, v 的 LCA 就是序列上出現 u 以及出現 v 這個區間中，編號最小的那一點。事實上任何一次出現的時間都可以，所以可以維護每個點第一次出現在序列中的位置，這樣 LCA 就轉為 RMQ 問題，可以用 Sparse Table 等資料結構解決。

1.3 樹鍊剖分

樹壓平提供了一個將子樹問題簡化的方法，但如果是路徑問題呢？在樹壓平後的序列上，一條路徑可能變得支離破散，不過一條一條的「樹鍊」卻仍然是一個個區間，所以可以將路徑拆成樹練們，對於每個樹鍊用資料結構維護動態操作即可。切樹練的方法有很多種，隨便亂做的話可能會造成經過的樹練太多，複雜度太差，以下介紹一種可以滿足任兩點路徑上只會經過 $O(\log N)$ 條數鍊的方法。

1.4 輕重鍊剖分（Heavy-Light Decomposition）

輕重鍊剖分是競賽場上最常使用的樹練剖分方法，顧名思義，將樹上的邊分為輕邊以及重邊，一個點只會連出一條重邊到大小最大的子樹，其餘子樹皆連輕邊，然後把接在一起的重邊當作樹練來維護。實作上只要預先處理每棵子樹的大小，然後對每個節點紀錄連接它的重鍊的頂端在哪，查詢任兩點的路徑的時候只要沿著重鍊往上爬並不斷跳鍊即可。
由於對於一個大小為 k 的子樹，若進行跳鍊則代表必存在另一棵子樹的大小 $\geq k$，所以跳上去以後的子樹大小 $\geq 2k+1$，所以可以保證執行輕重鍊剖分後，任兩點路徑間經過的輕邊數量至多為 $O(\log N)$。

2 樹分治

樹其實有很好的分治結構，對一個有根樹，如果可以將這棵樹的答案分成由它每一個子樹的答案們合併後求得，那不妨考慮採用樹分治。由於視習慣有人會將答案儲存在節點上而非 return 回來，因此有人又叫樹分治「樹 DP」。

用一個之前講過的例子來體會樹分治的精髓：找直徑。用跟分治演算法相同的想法來做，先將問題分為若干個小問題，一一求解後再合併答案。對於現在的關注的子樹 T 的根 u，T 的直徑分為有經過 u 或沒經過 u 的，第一種直徑必定是由某個葉節點連到 u 在連到另一個葉節點，因此必須找到 T 中兩個最深的點，也就是 T 的子樹中最深的點深度 $+1$ 的結果，因此把這個問題丟給子樹去做。第二種沒通過 u 的直徑一定只存在於 T 的其中一個子樹中，所以一樣把問題遞給子樹再取 max 即可。
上述例子中可以發現，樹分治的精髓就在於把跟自己不相干的東西都交給子樹處理，自己只要負責統合出自己的答案就好，所以如果遇到的問題可以由子樹答案拼湊出來的話，不妨考慮樹分治。

2.1 重心剖分

先看一下這個例題：給一棵無根樹 T，樹上有的節點有黑白兩種，詢問某個節點距離最近的黑節點為多遠。
靜態的版本可以用樹 DP 解決，不過如果是動態修改點上顏色的版本呢？如果這時再將重新 DFS 一次更新答案，顯然會太慢，因此需要重心剖分來解決它。
先定義 $M(u)$ 為將節點 u 刪除後最大子樹的大小，那麼可以證明在一棵 N 個節點的樹上，至少有一個、最多兩個節點滿足：$M(u) \leq \frac{N}{2}$，我們便稱 u 為 T 的重心。
如果將 T 的重心當作新的根，並遞迴將重心連到子樹的重心們上，就會得到所謂的重心樹。重心樹一個最直接的好處就在於它的深度量級是 $O(\log N)$ 的，因此可以以較好的複雜度支援一些動態樹上修改查詢的問題。
以上述例題當說明，對於每次的修改 u，就沿著重心樹上的父節點往上爬，並更新父節點們的答案（當然要先維護好點跟點之間的距離），查詢時只要一樣往根結點爬將答案與距離算好後取 min 即可。每次修改及查詢最多經過 $O(\log N)$ 的節點比起原先的 $O(N)$ 是個極大的改進。

一個常見的實作方法為，對於現在要重心剖分的樹算好每一個節點拔掉它後的最大子樹大小，找出一個答案 $\leq \frac{N}{2}$ 的節點，把它設為第 i 層重心樹的根，砍掉後對所有子樹遞迴即可。當然重心樹的運用很靈活，視題目不同需要在節點上維護不同的資訊以便更新以及計算答案，比較常見需要維護的有該節點到第 i 層的父節點的距離，每個節點需要 $\log N$ 的空間，其他的資訊大部分也都可以用這個形式維護好，因此一般來說重心剖分的空間複雜度是 $O(N \log N)$。

2.2 習題

1. SPOJ QTREE 系列：題敘略。
2. Codeforces 543D：給一個 $N(\leq 10^5)$ 個節點的無根樹，將每條邊編號 0 或 1，對於每個節點 u 求滿足每個節點到 u 都至多經過一條 1 號邊的編號方法數模 $10^9 + 7$。
3. Codeforces 342E：實作上述重心剖分的例題。
4. Codeforces 321C：給一個 $N(\leq 10^5)$ 個節點的無根樹，求一種將點編碼的方式（可用的編碼為 0 25），使得任兩個編碼相同的節點路徑上都有至少一個點編碼小於它。
5. TIOJ 1171：給一個 $N(\leq 10^5)$ 個節點的帶權無根樹，一開始每個節點都是白色，支援兩種操作
   (a) 將一個點塗成黑色
   (b) 詢問一個點到所有黑色點的距離總和

3 DP 優化

在上一次的 DP 課程中學到的 DP 是將所有有可能變成轉移來源的子問題們都先算過一遍後，再慢慢轉移求解。然而在有些問題中，並不是所有的子問題都可能被轉移到，因此，將不可能的來源略過不算進而減少複雜度的技巧就稱為「DP 優化」。
在以下的章節中，我們會用 eD/tD 來描述一個問題，代表這個問題共有 $O(N^e)$ 個狀態，每次轉移需要 $O(N^t)$ 的時間，總複雜度為 $O(N^{e+t})$。

3.1 斜率優化（Convex Hull optimization）

斜率優化能處理的 DP 式子需要符合下列形式：

$$dp[i] = \max_{j \le R_i}\{a[j]x[i] + b[j]\}$$

其中 R_i 代表第 i 個 DP 的轉移來源的右界。直接算的話複雜度為 $O(N^2)$。可以觀察到若 R_i 是遞增的話，假如存在一個 j' 使得 $a[j'] > a[j]$ 且 $a[j']x[i] + b[j'] > a[j]x[i] + b[j]$，那麼對於所有的 $i' > i$，假設有 $x[i'] \ge x[i]$ 的話，j 都再也不可能是轉移的來源，因此可以略過不算。

若將所有還有可能被轉移的來源都在座標平面上做一條斜直線 $y = a[j]x + b[j]$ 的話，會形成一個斜率遞增的線集。如果對於每個 x 再將其在線集中對應有最大值的那條線標出來的話，那看起來會形成一個下凸包（因此斜率優化也稱為凸包優化）。那麼每次要計算 $dp[i]$ 的時候，只要找出 $x = x[i]$ 與下凸包的的交點，並在加入一條新的線後好好維護這個凸包就好了。
首先是找到交點的部分，由於斜率遞增，因此可以二分搜答案位於那條線上。不過如果 $x[i]$ 有單調性的話，只要看上一次的轉移來源是否還是這次的轉移來源，如果是就直接轉移，不是的話就把上次的轉移來源丟掉再繼續做下去就好了。
再來是如何在計算完 $dp[i]$ 之後把直線 L_i 加入線集並維護好凸包。先二分搜 L_i 在線集中的位置，並且把加入 L_i 後不再可能成為轉移來源的線們都丟掉。實作上可以比較線的交點大小，不過由於浮點數精度問題以及方便查詢時的二分搜，更好且簡單的方法是維護每一條線 L_j 的有效範圍 $[l_j, r_j]$，若 L_j 在 $[l_j, r_j]$ 中都輸給 L_i 的話，就把 L_j 踢除，並重複此操作直到不能再踢除別的線為止。
因為要支援二分搜以及刪除加入元素，set 是一個不錯的選擇。由於每條線最多被加入 set 一次以及拔出 set 一次，故總複雜度可以優化為 $O(N \log N)$。

以 ZJOI 2007 倉庫建設為例：在 x 軸上有 N 個點，其中第 i 個點上有 $p[i]$ 個物品，在這個點上蓋倉庫的費用為 $c[i]$ 且這個點距離第 1 個點的距離為 $d[i]$（$d[i]$ 遞增），所有物品只能往座標大的點移動，且將 1 單位物品移動距離 x 需要 x 元，求在一些點上蓋倉庫並將

所有物品移至倉庫的最小費用。

令 $dp[i]$ 為在點 i 上蓋倉庫且將 $[1,i]$ 中的物品都移動至位於 $[1,i]$ 的倉庫的最小值，不難發現由於第 N 個點一定得蓋倉庫，因此問題的答案就是 $dp[N]$：

$$dp[i] = c[i] + \min_{j<i}\{dp[j] + \sum_{k=j+1}^{i}(d[i]-d[k])p[k]\}$$

將式子化簡：

$$dp[i] = c[i] + \min_{j<i}\{dp[j] + d[i]\sum_{k=j+1}^{i}p[k] - \sum_{k=j+1}^{i}d[k]p[k]\}$$

維護 p 的前綴和以及 d 與 p 的前綴乘積和，令 $s[i] = \sum_{j=1}^{i}p[j], t[i] = \sum_{j=1}^{i}p[j]d[j]$：

$$\begin{aligned} dp[i] &= c[i] + \min_{j<i}\{dp[j] + d[i](s[i]-s[j]) - (t[i]-t[j])\} \\ &= c[i] + \min_{j<i}\{dp[j] + d[i]s[i] - d[i]s[j] - t[i] + t[j]\} \\ &= c[i] + d[i]s[i] - t[i] + \min_{j<i}\{-s[j]d[i] + dp[j] + t[j]\} \end{aligned}$$

變成斜率優化的形式了（$a[j] = -s[j], b[j] = dp[j] + t[j], x[j] = d[j]$），用 set 維護上凸包可以將問題優化為 $O(N\log N)$。不過這樣還是無法拿到滿分，必須利用斜率與查詢的單調性做進一步的優化。

3.2 單調隊列優化

單調隊列優化本質上可視為斜率優化。當斜率（$a[i]$）以及查詢（$x[i]$）皆具有單調性時，不再需要使用二分搜判斷當前的轉移來源以及新的直線加入線集的位置。以下凸包為例，若每次加入的線斜率遞增，且查詢也遞增時，每次只要把斜率小的且不再是最好的元素丟掉，加入新的直線時直接比較會將哪些斜率大的線淘汰便可以完成。實作上因為需要對頭尾進行操作，因此可以使用 deque，每個元素進出至多進出 deque 各一次，總複雜度可以再優化為 $O(N)$。

3.3 四邊形優化（Knuth's optimization）

相較於斜率優化，四邊形優化較難直接套用，通常都需要對題目有比較縝密的觀察加上證明才能好好使用它。

在說明四邊形優化前，先介紹兩個之後會利用到的單調性以及一些性質（以下單調性定義在矩陣 B 上）：

1. 凹完全單調性（concave totally monotone）：

$$\forall i < i', j < j'，B[i][j] \leq B[i'][j] \Rightarrow B[i][j'] \leq B[i'][j']$$

2. 凸完全單調性（convex totally monotone）：

$$\forall i < i', j < j'，B[i][j] \geq B[i'][j] \Rightarrow B[i][j'] \geq B[i'][j']$$

3.4 $1D/1D$ 凹性優化

首先先定義一個 $1D/1D$ 的 DP 問題：

$$dp[j] = \min_{i<j}\{f(i, j, dp[i])\}$$

其中 f 為代價函數，通常會包含 $dp[i]$ 的值（若沒有包含 $dp[i]$ 的話，可以用分治法做到 $O(N \log N)$ 的複雜度，在後面的章節會做介紹），為了方便起見，我們通常將 $f(i, j, dp[i])$ 寫為 $f(i, j)$。
若把 $f(i, j)$ 用一個矩陣 B 來表示（基於 f 的定義，矩陣 B 只有上三角是合法的），原本求的 $dp[j]$ 就等價於求矩陣第 j 欄的最小值。樸素的做法複雜度為 $O(N^2)$，因此我們接著討論若矩陣 B 符合凹單調性，會對複雜度有怎樣的改善。

不妨假設矩陣中所有元素皆相異，觀察後可以發現若 B 符合凹單調性，且已知 $B[i][j]$ 為第 j 欄的最小值，那麼對於所有 $j' < j, i' < i$ 都有 $B[i'][j'] > B[i][j']$（因為 $B[i][j] < B[i'][j]$，可由凹單調性的逆否命題推得），因此所有位於 $B[i][j]$ 左上角的元素都不再可能成為轉移來源，同理，對於所有 $j' > j, i' > i$ 也有 $B[i'][j'] > B[i][j]$，因此所有右下角的元素也都被淘汰了。
如何好好的利用這個性質呢？當我們一欄一欄的的計算最小值時，做完第 j 欄時要將 M 的第 j 列插入，因此我們在這之前維護前 $j-1$ 列的最小值們，基於剛剛的觀察可以發現所有可能的轉移來源們會形成一個左下到右上的趨勢，因此不妨用 $\{[i_1, j : r_1], [i_2, r_1 + 1 : r_2], \ldots, [i_k, r_{k-1} + 1 : r_k]\}$ 來維護這些轉移來源，其中 $[i, l : r]$ 代表第 i 列是最小值的部分為第 l 欄到第 r 欄。不難發現，每一列的最右元素 r_k 扮演著關鍵角色：若新插入的元素使得 $B[i_k][r_k]$ 被淘汰，那在這整個區間中，第 i_k 列都不在可能是轉移來源。因此維護這些區間，如果有 $B[j][r_k] < B[i_k][r_k]$ 的話就將這個區間淘汰。

實作上可以維護一個資料結構紀錄尚未被淘汰掉的區間們，照左界遞增的順序排好。每加入一個元素時不斷檢查資料結構中左界最小的的區間並判斷 (令它為 $[i_k, l_k : r_k]$)：

1. $B[j][r_k] < B[i_k][r_k]$：代表這整段區間已經被淘汰了，把它丟掉後重複繼續做下去。
2. $B[j][r_k] \geq B[i_k][r_k]$：由於這是資料結構中第一個不完全被淘汰的區間，因此必存在一個 $l_k \leq c < r_k$ 使得 $B[j][c] < M[i_k][c]$ 且 $B[j][c+1] \geq B[i_k][c+1]$。$c$ 的位置可以

二分搜，找到後便修改資料結構中左界最小的區間為 $[i_k, c+1 : r_k]$ 並將新的區間 $[j, j+1 : c]$ 加入資料結構中。

以上的流程忽略一些邊界條件（例如當資料結構為空時或元素過期時），不過細節應該不難想，實作時再判掉就好。由於每次需要存取的都是左界最小的區間，插入的區間也會是左界最小，因此一個 stack 可以勝任上述所有操作。
最後分析複雜度：每個元素最多進出 stack 一次，因此均攤 $O(N)$，每次迴圈進行一次二分搜複雜度 $O(\log N)$，因此套用凹單調優化後的總複雜度降為 $O(N \log N)$。

3.5 $1D/1D$ 凸性優化

如果在 $1D/1D$ 的問題中，$f(i,j)$ 符合凸單調性的話，優化的方法與凹單調大同小異。一樣先觀察矩陣 B，可以發現若 $B[i][j]$ 為第 j 欄最小值，則對於所有 $i' < i, j' > j$ 以及 $i' > i, j' < j$ 都有 $B[i'][j'] > B[i][j']$，因此一樣維護每一列是最小值時的欄位所形成的區間們，不過改成 $[i_k, l_k : l_{k+1} - 1]$ 的形式。每一個區間最具影響力的從 r_k 變為 l_k，因此從左界最小的轉移，從左界最大的地方插入新的區間，並淘汰所有滿足 $B[j][l_k] < B[i_k][l_k]$ 的元素，在第一個符合 $B[j][l_k] > B[i_k][l_k]$ 的區間上二分搜最大的 $l_k \le c \le r_k$ 使得 $B[j][c] > B[i_k][c]$，將此元素修改為 $[i_k, c : r_k]$ 並在資料結構中加入 $[j, c+1 : N]$。由於這次資料結構必須要提供兩端的存取操作，因此 deque 是一個不錯的選擇。凸性優化的複雜度一樣為 $O(N \log N)$。

Algorithm 1: $1D/1D$ convex optimization

```
1  void solve() {
2      deque<segment> dq; dq.push_back({ 0, 1, n });
3      for (int i = 1; i <= n; ++i) {
4          dp[i] = f(dq.front().i, i);
5          while (dq.size() && dq.front().r < i + 1)
6              dq.pop_front();
7          dq.front().l = i + 1;
8          while (dq.size() && f(i, dq.back().l) < f(dq.back().i, dq.back().l))
9              dq.pop_back();
10         segment new_seg = { i, i + 1, n };
11         if (dq.size()) {
12             int c = bs();
13             // c is the maximal k such that f(i, k) > f(dq.back().i, k)
14             dq.back().r = c;
15             new_seg.l = c + 1;
16         }
17         if (new_seg.l <= n) dq.push_back(new_seg);
18     }
19 }
```

3.6 Monge condition

上述兩種 $1D/1D$ 的優化，最關鍵的點在於代價函數的單調性，不過看出這個單調性除了需要直覺以外，有時還需要更嚴謹的證明，而 Monge condition 正可以提供一些幫助。
先定義一個 $N \times M$ 的矩陣 B 符合 Monge Condition：

1. $\forall 1 \le i < i' \le N, 1 \le j < j' \le M$ 都有 $B[i][j] + B[i'][j'] \le B[i][j'] + B[i'][j]$，則符合 convex Monge condition。
2. $\forall 1 \le i < i' \le N, 1 \le j < j' \le M$ 都有 $B[i][j] + B[i'][j'] \ge B[i][j'] + B[i'][j]$，則符合 concave Monge condition。

另外，如果 $i, j \in \mathbb{N}_0$，我們可以把它換成比較好驗證的形式：

$$\forall 1 \le i < N, 1 \le j < M \text{都有} B[i][j] + B[i+1][j+1] \le (\ge) B[i+1][j] + B[i][j+1]$$

則符合 convex（concave）Monge condition。
事實上可以由 convex（concave）Monge condition 直接推得凸（凹）單調性，不過由於 Monge condition 較為嚴謹，因此反向的推導並不成立。

3.7 $2D/1D$ 優化

$2D/1D$ 的問題指的是：

$$dp[i][j] = w(i,j) + \min_{i \le k < j} \{dp[i][k] + dp[k+1][j]\}, dp[i][i] = 0$$

顯然有 $O(N^3)$ 的做法：枚舉 $i \le k < j$ 的每一種可能性並轉移。
令 $h_{i,j}$ 為令 $dp[i][j]$ 最佳的 k 值。如果 dp 表格符合 convex Monge condition 的話，則可以證明：

$$h_{i,j-1} \le h_{i,j} \le h_{i+1,j}$$

也就是說，$dp[i][j]$ 的轉移來源必會在 $dp[i][j-1]$ 到 $dp[i+1][j]$ 的轉移來源之間，如果此時我們改以 $j - i$ 遞增的方向計算 dp 表格的話，那麼 $dp[i][j-1]$ 以及 $dp[i+1][j]$ 都會在 $dp[i][j]$ 之前算好，我們就可以將要枚舉的 k 的範圍縮小了。
不過到底 $h_{i,j-1}$ 以及 $h_{i+1,j}$ 的值會讓我們的複雜度將為多少呢？考慮計算完一個 dp 表格的斜排所需的時間：

$$\begin{aligned}
h_{i-2,j-3} \le h_{i-2,j-2} \le h_{i-1,j-2} \\
h_{i-1,j-2} \le h_{i-1,j-1} \le h_{i,j-1} \\
h_{i,j-1} \le h_{i,j} \le h_{i+1,j} \\
h_{i+1,j} \le h_{i+1,j+1} \le h_{i+2,j+1} \\
h_{i+2,j+1} \le h_{i+2,j+2} \le h_{i+3,j+2}
\end{aligned}$$

從上面的式子中可以發現同一斜排所需要轉移的量不超過 N 個，整個 dp 表格中又大概有 $2N$ 的斜排，故總複雜度為 $O(N^2)$。

3.8 分治優化

在 $1D/1D$ 的章節中有提到，若代價函數 f 中不包含 DP 值，或代價函數可以離線求得，則有更簡單的優化方法。
令 h_i 為讓 $dp[i]$ 最小的 j 值，若可以證明 $\forall i' < i$，$h_{i'} \leq h_i$，則可以知道對於一個要計算的區間 $[L, R]$，如果令 $M = \frac{L+R}{2}, t = h_M$，那 $[L, M-1]$ 的轉移來源必定小於等於 t 且 $[M+1, R]$ 區間的轉移來源必大於等於 t，又因為 f 是可以離線的，所以每次將要求的區間砍半遞迴下去做，根據主定理 $T(N) = 2T(\frac{N}{2}) + O(N)$，複雜度可以從 $O(N^2)$ 優化為 $O(N \log N)$。

一個相當常見的分治優化的例子為：將一個長度為 N 的序列分為 K 個連續非空且不相交的區間，定義 $w(l, r)$ 為將 $[l, r]$ 分為一個區間的代價，求分為 K 段後代價總和最小。

$$dp[k][i] = \min_{j<i}\{dp[k-1][j] + w(j+1, i)\}$$

其中 $dp[k][i]$ 為 $[1, i]$ 的前綴分為 k 段的最小代價，由於每次轉移都不會用到 $dp[k][j]$ 的值，因此若能證明轉移來源單調，即可以套用分治優化把複雜度從 $O(KN^2)$ 降為 $O(KN \log N)$。

3.9 小結

感謝去年學長（姐 (?)）編的講義讓這份講義可以誕生，這章節大部分的內容都是讀完去年講義＋ IOIC 講義＋網路上的 Blog 搞懂後重新整理出來的，如果有看不懂的可以去翻去年講義或是自己 DFS 網路，如果再不能理解的話那就比賽寫 DP 發現 TLE 時再亂套試試看就好了 (?)。

3.10 習題

1. 有限背包問題，複雜度 $O(NW)$
2. APIO 2010 Commando（TIOJ 1745）：給定長度為 $N(\leq 10^6)$ 的序列以及一二次方程 $f(x) = ax^2 + bx + c$，將序列分為若干塊，令第 i 塊區間和為 S_i，求 $\sum f(S_i)$ 的最大值。
3. IOI 2002 Batch Scheduling（wcipeg）：有 $N(\leq 10^6)$ 個工作，每個工作有 T_i, F_i 兩個係數，用一台待機時間為 S 的機器執行這些工作。將工作們分為若干塊（每一塊必須包含連續元素，且分塊後的順序遵守原本工作的順序），每一塊的工作會同時完成（若

前一塊工作完成時間為 t 且這塊工作包含編號 $x, x+1, x+2, \ldots, x+k$ 的工作們，則此塊工作完成時間為 $t + S + \sum_{i=x}^{x+k} T_i$），將每個工作的結束時間標為 O_i，求 $\sum_{i=1}^{N} O_i F_i$ 的最小值。

4. ACM-ICPC World Final 2011 pF（TIOJ 1921）：有 $N(\leq 10^5)$ 臺機器，其中第 i 臺會在時間 d_i 拍賣，價格為 p_i，此機器每天會生產 g_i 元，且將這台機器轉賣可得 r_i 元，一個時間只能擁有一台機器，你一開始有 C 元，求在第 D 天後你最多可以賺進多少錢？

5. TIOJ 1347：將一長度為 $N(\leq 10^5)$ 的序列分為若干塊，第 i 塊的長度 M_i 為 $\sum_{j=l_i}^{r_i} A_j + \sum_{j=l_i}^{r_i-1} L_j$（$l_i, r_i$ 分別為此塊最左及最右的編號），求 $\sum |M_i - K|^p$ 的最小值。

6. TIOJ 1283：給定一個由不斷往右往下的線段所形成的多邊形，求多邊形中能塞得下的最大矩形面積（多邊形的邊數 $N \leq 2 \times 10^5$）。

7. Codeforces 321E：有 $N(\leq 4000)$ 個人要搭車，你需要將他們分到 $K(\leq 800)$ 個車子裡，且一台車裡只能載編號連續的人們。人跟人之間都有互相討厭的程度，而整車的不友善指數就是車裡每對人討厭程度的總和，求 K 個車不友善程度最小的總和。

8. Codeforces 833B：將一個長度為 $N(\leq 35000)$ 的序列分為 $K(K \leq \min(N, 50))$ 段，每段的代價是區段中相異數的個數，求總代價最小。（這題有不用 DP 優化的做法）

9. BZOJ 1096：斜率優化例題。

10. TIOJ 1639：題敘略

11. TIOJ 1676：對於一個長度為 $N(\leq 5 \times 10^5)$ 的序列，將序列分成連續的區段，總代價為 $\sum_{i=1}^{m} \{T_i(i-1) - S_i^2\}$（其中 m 為你分的區段個數、T_i, S_i 分別為第 i 段的區間和及大小），且每區段長度不能超過 K 的情況下，求總代價最小。

12. Codeforces 631E：給一個長度為 $N(\leq 2 \times 10^5)$ 的序列，定義一個操作為將序列中的元素拔出再插入任意位置，求在做一個操作後，$\sum_{i=1}^{N} a[i] \cdot i$ 最大可為多少。

13. Codeforces 311B：x 軸上有 $N(\leq 10^5)$ 個點，第 i 個點與第 $i-1$ 個點的距離為 d_i，有 M 的物品會在 t_i 時間點出現在標號為 h_i 的點上，你可以在任意時間，總共派出 $p(\leq 100)$ 台機器從原點出發以 1 單位的速度往 x 軸正向走，撿起每個出現的點，求每個點出現到被抓到的時間差總和最小為多少。

14. IOI 2016 Aliens（TIOJ 1961）：給定一個 $M \times M$（$M \leq 10^6$）的網格以及網格上 $N(\leq 10^5)$ 的點，拍攝至多 $K(\leq N)$ 張照片（照片對角線必須在網格主對角線上）且 N 個點都至少被一張照片拍到，求被照片照到的格數最小為何。