JOI 春合宿 2012 Day1
ビルの飾り付け 2

秋葉 拓哉

Building (JOI 春合宿 '07)

Building: ビルの飾りつけ

input ファイル "building.in"
output 標準出力
ソースファイル building.c/building.cpp/Building.java
時間制限 1 秒 / データ

国際情報オリンピックが日本で開かれることとなり，世界の選手達を歓迎するため，空港から宿泊施設までの道沿いにある高層ビルを飾りつけることにした．ある著名なデザイナーにデザインを依頼したところ，飾りつけに利用するビルは，空港から宿泊施設に向けて高くなっていく必要があると言った．つまり，飾りつけに利用するビルの高さを，空港に近いものから順に $h_1, h_2, h_3, \cdots$ とおくと，$h_1 < h_2 < h_3 < \cdots$ となっていなければならない．

できるだけ飾りつけを華やかにするため，飾りつけに利用するビルの数をできるだけ多くしたい．入力として全てのビルの高さが与えられたとき，利用することのできるビルの数の最大

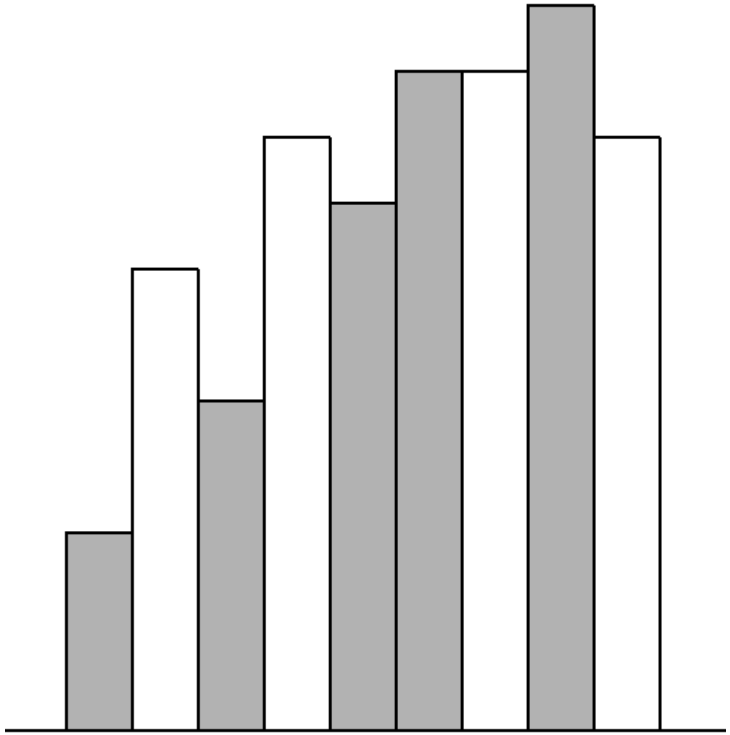
問題概要

- 最長増加部分列を求めよ
- ただしツリー上

「ツリー上のパスの部分列であって
増加しているものの最長のものを求めよ」

普通の最長増加部分列

- ナイーブな DP　(以下 *LIS-DP1* と呼ぶ)
- dp[i] := max {dp[j] + 1}
  - ただし j < i, a[j] < a[i]
- 某本 P. 64



dp[i]	1	2	2	3	3	4	4	5	4

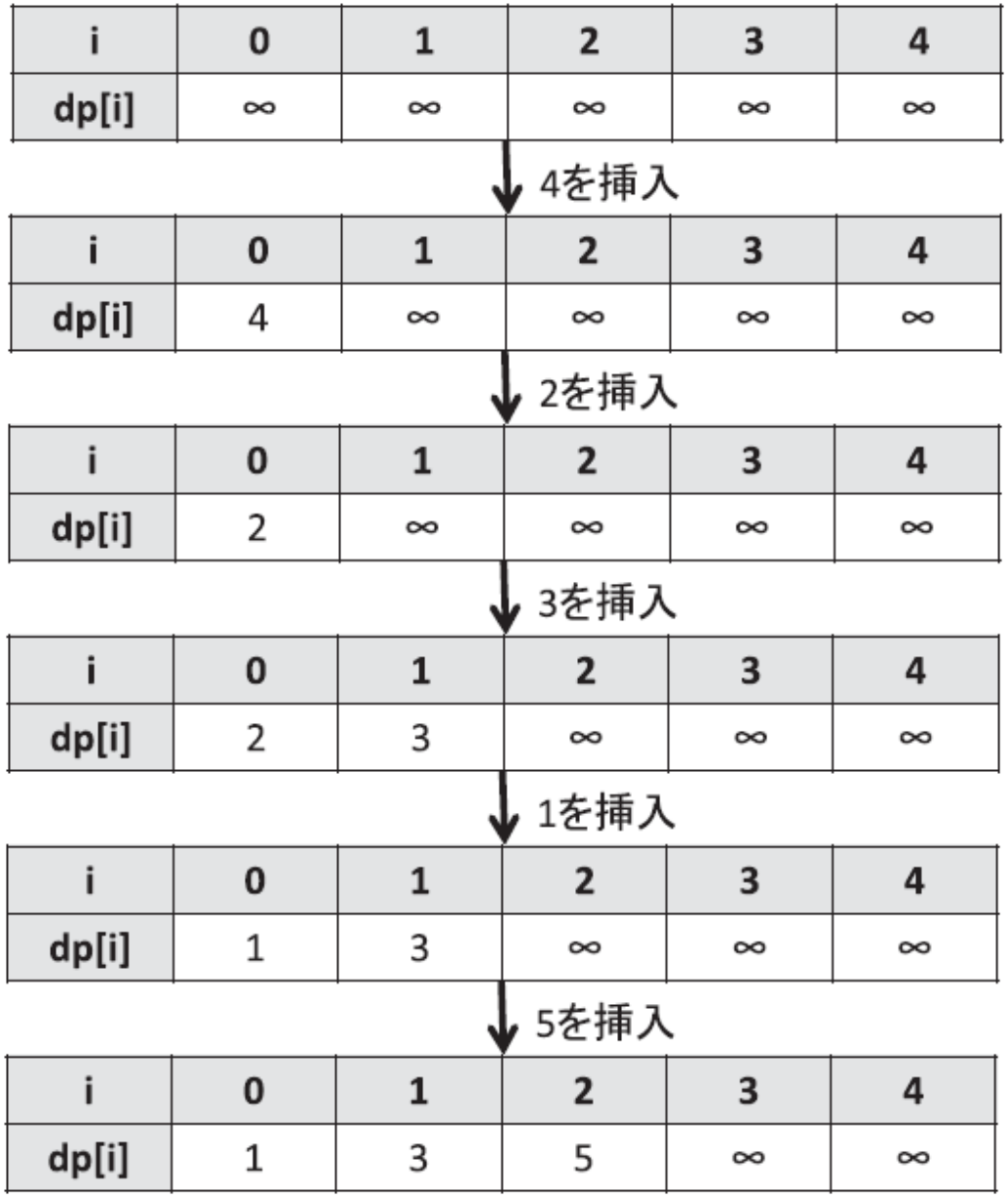
普通の最長増加部分列

- 高速な DP　(以下 *LIS-DP2* と呼ぼう)

- $dp_i[k]$:= 場所 i までで長さ k の列を作る時の
  最後の数字の最小値

- ナイーブに更新すると $O(n^2)$
- 更新位置を二分探索で求めれば $O(n \log n)$

- 某本 P. 65


i 0 1 2 3 4
dp[i] ∞ ∞ ∞ ∞ ∞
4を挿入
i 0 1 2 3 4
dp[i] 4 ∞ ∞ ∞ ∞
2を挿入
i 0 1 2 3 4
dp[i] 2 ∞ ∞ ∞ ∞
3を挿入
i 0 1 2 3 4
dp[i] 2 3 ∞ ∞ ∞
1を挿入
i 0 1 2 3 4
dp[i] 1 3 ∞ ∞ ∞
5を挿入
i 0 1 2 3 4
dp[i] 1 3 5 ∞ ∞

ナイーブな解法 (10 点)

- $O(N^4)$ / $O(N^3 \log N)$
  - 全部のパス N^2 個
  - 各パスについて $O(N^2)$ / $O(N \log N)$ のDP

- $O(N^3)$
  - 開始位置 N 個全部試す，再帰して木 DP
  - *LIS-DP1* と同様，各位置を末尾にした際の最長の LIS 長を覚えておき DP

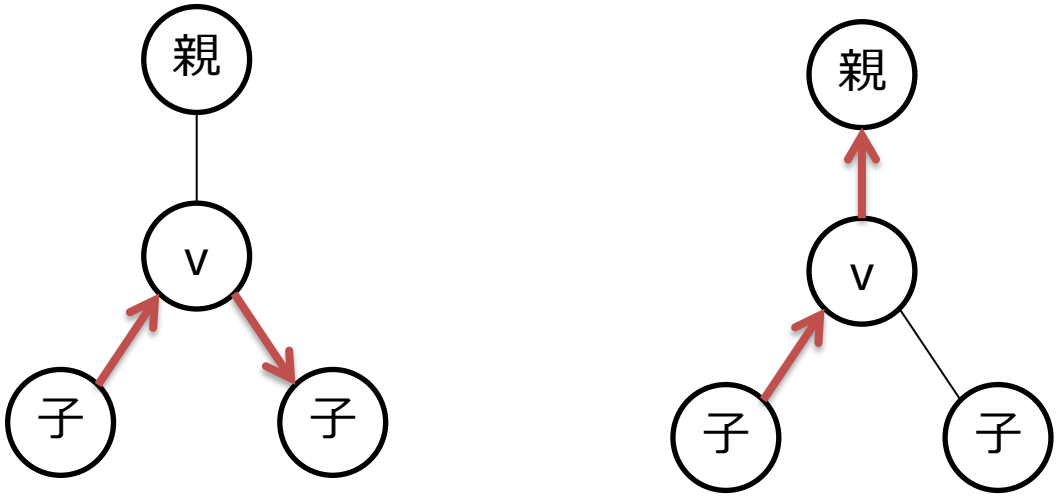
$O(n^2 \log n)$ の解法 (30 点)

- すべての開始位置を試す
- *LIS-DP2* をやりながら再帰
- 枝分かれは？
  - 再帰から戻るとき，自分のやった変更を**巻き戻す**ようにすればよい
    （そうすれば次の枝にもいける）

$O(n \log^2 n)$ の解法 (100 点)

- 適当に根を決めて DFS，木 DP

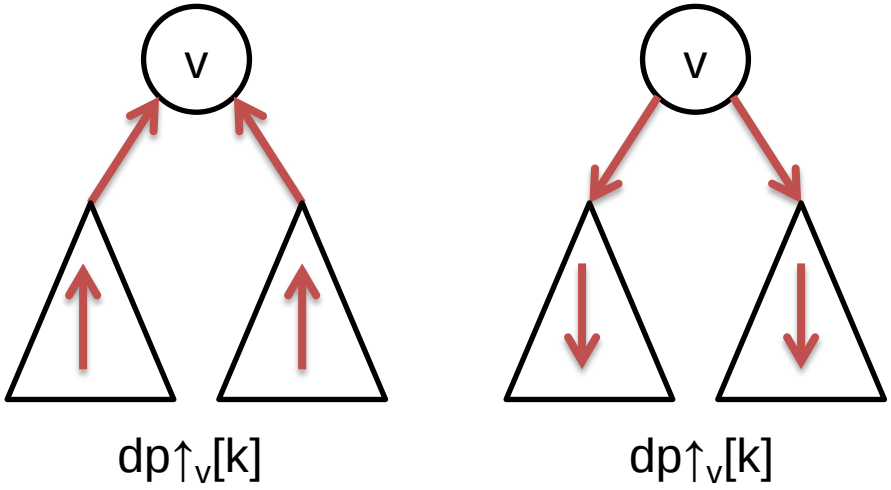
- あるノード v に関して，答えの移動経路が v を通るとすると，以下の 2 パターン



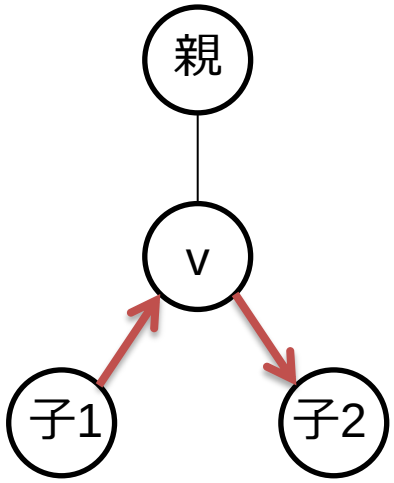
DP

LIS-DP2 の DP 列的な物をボトムアップに計算

- $dp\uparrow_v[k]$:= v に向かって遡ってくる LIS で長さ k の物を作る際の最後の数の最小値
- $dp\downarrow_v[k]$:= 逆 (Longest Decreasing Sequence)



頂点 v での処理 1

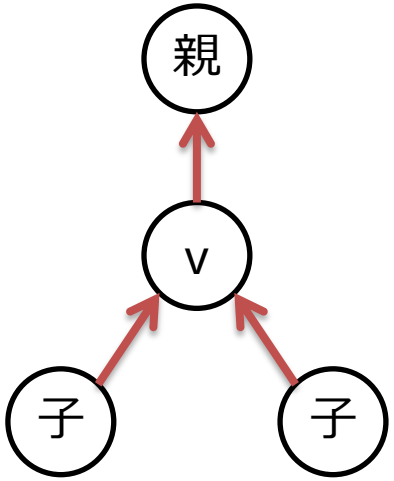


- 右図みたいなパターンの分を計算
- $dp\uparrow_{子1}$ と $dp\downarrow_{子2}$ とかから求まる
- v を使う場合
  - $dp\uparrow_{子1}[i] < H[v] < dp\downarrow_{子2}[j]$ → 長さ i + 1 + j は作れる
- v を使わない場合
  - $dp\uparrow_{子1}[i] < dp\downarrow_{子2}[j]$ → 長さ i + j は作れる
  - $dp\uparrow$ か $dp\downarrow$ の片方を全部なめつつ,
    もう片方を二分探索すればよい

頂点 v での処理 2

- 上にいくために $dp\uparrow_v$ とかを計算

- $dp\uparrow_v[k] = \min(dp\uparrow_{子1}[k], dp\uparrow_{子2}[k])$
- H[v] を挿入



計算量

- これ毎頂点でやったらやばくない？
  - 毎回長さ $O(n)$ の DP 列舐めてたら $O(n^2)$ ！
- 少し工夫すれば実はやばくない
  - 短い方を舐めるようにすれば良い
  - これだけで $O(n \log n)$ になる
  - 舐められた時，次にそいつが含まれる所のサイズは少なくとも 2 倍になる
    （併合先のほうがでかい）
  - よって各要素は $O(\log n)$ 回しか舐められない.
    （これはデータ構造をマージする一般的なテク）

…おわり

- 説明大変でしたが，基本的なテクの組合せ

- LIS の高速 DP
- 木上でのパスのための DP
- データ構造のマージ (的思想)

得点分布