طراحی الگوریتم

(برنامه‌ریزی پویا)

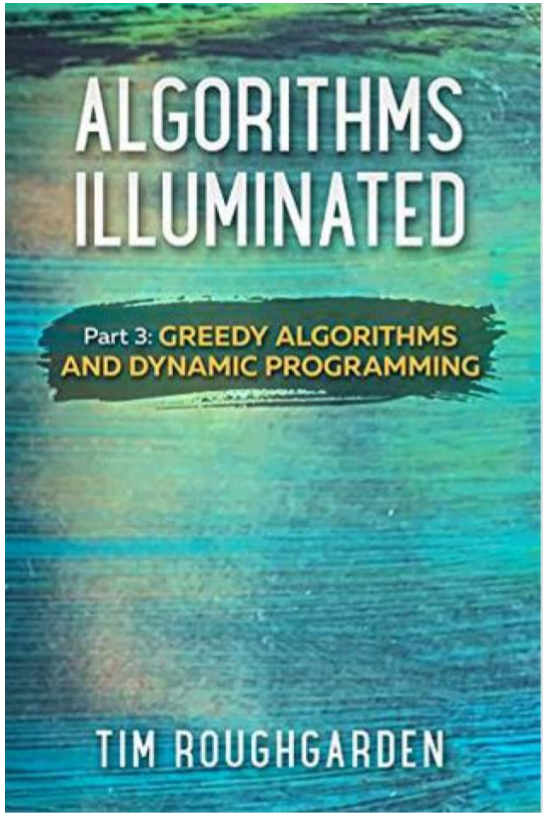
دانشکده مهندسی برق و کامپیوتر، دانشگاه صنعتی اصفهان

بهار ۹۹

نگاه کلی به برنامه‌ریزی پویا

- یک مجموعه نسبتا کوچک از زیر مساله‌ها را مشخص می‌نماییم.
- نشان دادن اینکه چگونه با داشتن جواب زیرمساله‌های کوچکتر می‌توان در زمان کمی جواب صحیح برای زیرمساله‌های بزرگتر را به دست آورد.
- چگونه می‌توان جواب صحیح نهایی را سریع از جواب تمام زیرمساله‌ها به دست آورد.

درخت جستجوی باینری بهینه

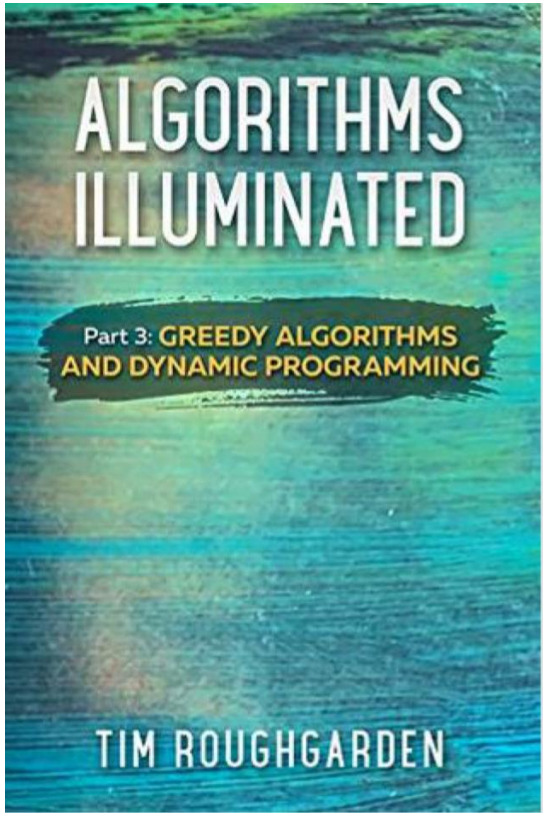


فصل هفدهم، صفحه ۱۴۸

درخت جستجوی باینری بهینه

ورودی: یک دنباله از کلیدها و برای هر کلید یک فراوانی نامنفی.

هدف: درخت جستجوی باینری شامل تمام کلیدها با کمترین زمان جستجو.



فصل هفدهم، صفحه ۱۴۸

ساختار جواب بهینه

ساختار جواب بهینه

رویکرد برنامه‌ریزی پویا

OptBST

Input: keys $\{1, 2, \ldots, n\}$ with nonnegative frequencies $p_1, p_2, \ldots, p_n$.
Output: the minimum weighted search time (17.1) of a binary search tree with the keys $\{1, 2, \ldots, n\}$.

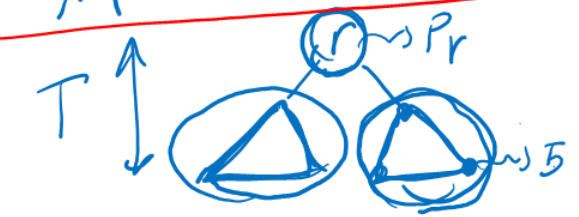
// subproblems (i indexed from 1, j from 0)
$A := (n+1) \times (n+1)$ two-dimensional array
// base cases ($i = j + 1$)
for $i = 1$ to $n+1$ **do**
    $A[i][i-1] := 0$
// systematically solve all subproblems ($i \leq j$)
for $s = 0$ to $n-1$ **do** // s=subproblem size-1
    for $i = 1$ to $n-s$ **do** // $i+s$ plays role of j
        // use recurrence from Corollary 17.5
        $A[i][i+s] :=$
            $\sum_{k=i}^{i+s} p_k + \min_{r=i}^{i+s} \{\underbrace{A[i][r-1] + A[r+1][i+s]}_{\text{Case } r}\}$
return $A[1][n]$ // solution to largest subproblem

نتیجه:

$W_{i,j}$ = زمان جستجو در درخت بهینه با کلیدهای $\{i, i+1, \ldots, j\}$ و فراوانی‌های $p_i, p_{i+1}, \ldots, p_j$ باشد

$W_{i,j} = 0$ زمانی که $j < i$

$$W_{i,j} = \min_{r \in \{i, \ldots, j\}} \{W_{i,r-1} + W_{r+1,j}\} + \sum_{k=1}^{n} p_k$$



رویکرد برنامه‌ریزی پویا

OptBST

Input: keys $\{1, 2, \ldots, n\}$ with nonnegative frequencies $p_1, p_2, \ldots, p_n$.

Output: the minimum weighted search time (17.1) of a binary search tree with the keys $\{1, 2, \ldots, n\}$.

// subproblems (i indexed from 1, j from 0)

$A := (n+1) \times (n+1)$ two-dimensional array

// base cases ($i = j + 1$)

for $i = 1$ to $n + 1$ **do**

  $A[i][i-1] := 0$

// systematically solve all subproblems ($i \leq j$)

for $s = 0$ to $n - 1$ **do** // s=subproblem size-1

  for $i = 1$ to $n - s$ **do** // $i + s$ plays role of j

    // use recurrence from Corollary 17.5

    $A[i][i+s] :=$

$$\sum_{k=i}^{i+s} p_k + \min_{r=i}^{i+s} \{\underbrace{A[i][r-1] + A[r+1][i+s]}_{\text{Case } r}\}$$

return $A[1][n]$ // solution to largest subproblem