طراحی الگوریتم

(برنامه‌ریزی پویا)

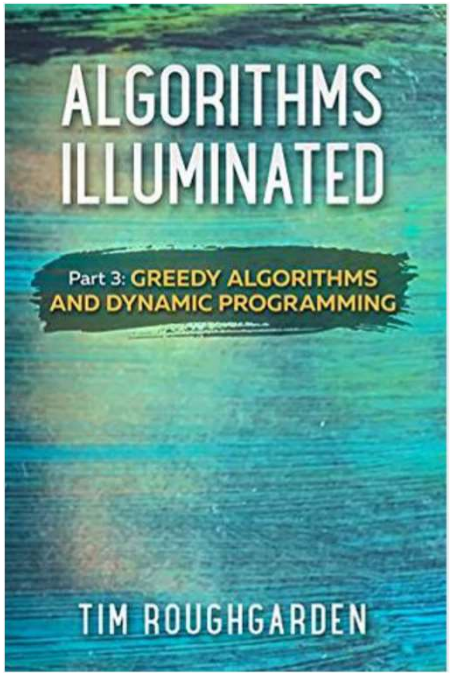
دانشکده مهندسی برق و کامپیوتر، دانشگاه صنعتی اصفهان

بهار ۱۴۰۰

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.

کوله‌پشتی (حالت صحیح)



فصل شانزدهم، صفحه ۱۲۳

ورودی: ظرفیت کوله‌پشتی و ارزش n کالا به همراه حجم هر کالا.

هدف: پیدا کردن مجموعه‌ای از کالاها با بیشترین ارزش به‌طوریکه مجموع حجم آن‌ها از ظرفیت کوله‌پشتی بیشتر نشود.

ساختار جواب بهینه

ساختار جواب بهینه

رویکرد برنامه‌ریزی پویا

Knapsack

Input: item values $v_1, \ldots, v_n$, item sizes $s_1, \ldots, s_n$, and a knapsack capacity C (all positive integers).
Output: the maximum total value of a subset $S \subseteq \{1, 2, \ldots, n\}$ with $\sum_{i \in S} s_i \leq C$.

```
// subproblem solutions (indexed from 0)
A := (n + 1) × (C + 1) two-dimensional array
// base case (i = 0)
for c = 0 to C do
    A[0][c] = 0
// systematically solve all subproblems
for i = 1 to n do
    for c = 0 to C do
        // use recurrence from Corollary 16.5
        if s_i > c then
            A[i][c] := A[i − 1][c]
        else
            A[i][c] :=
              max{A[i − 1][c], A[i − 1][c − s_i] + v_i}
                  Case 1          Case 2
return A[n][C] // solution to largest subproblem
```

نتیجه: با خازهای کم قبل، زیرمساله های زیر را در نظر می‌گیریم.

$V_{i,c}$ = جواب بهینه زیرمساله با ورودی از کالای اول و ظرفیت کوله پشتی c

برای هر $0 \leq i \leq n$ و $0 \leq c \leq C$،

$V_{0,c} = 0$

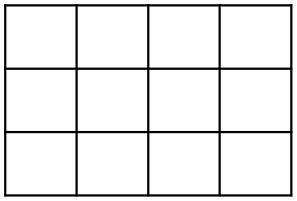
$$V_{i,c} = \begin{cases} \underbrace{V_{i-1,c}}_{\text{حالت اول}} & s_i > c \\ \max\{\underbrace{V_{i-1,c}}_{\text{حالت اول}}, \underbrace{V_{i-1,c-s_i} + v_i}_{\text{حالت دوم}}\} & s_i \leq c \end{cases}$$

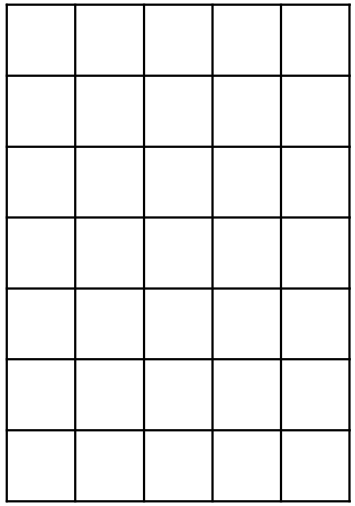
رویکرد برنامه‌ریزی پویا

Knapsack

Input: item values $v_1, \ldots, v_n$, item sizes $s_1, \ldots, s_n$, and a knapsack capacity C (all positive integers).
Output: the maximum total value of a subset $S \subseteq \{1, 2, \ldots, n\}$ with $\sum_{i \in S} s_i \leq C$.

```
// subproblem solutions (indexed from 0)
A := (n + 1) × (C + 1) two-dimensional array
// base case (i = 0)
for c = 0 to C do
    A[0][c] = 0
// systematically solve all subproblems
for i = 1 to n do
    for c = 0 to C do
        // use recurrence from Corollary 16.5
        if s_i > c then
            A[i][c] := A[i − 1][c]
        else
            A[i][c] :=
              max{A[i − 1][c], A[i − 1][c − s_i] + v_i}
                  Case 1          Case 2
return A[n][C] // solution to largest subproblem
```





بازیابی جواب بهینه

$$V_{i,c} = \begin{cases} \underbrace{V_{i-1,c}}_{Case\ 1} & if\ s_i > c \\ \max\{\underbrace{V_{i-1,c}}_{Case\ 1}, \underbrace{V_{i-1,c-s_i} + v_i}_{Case\ 2}\} & if\ s_i \le c. \end{cases}$$

Knapsack Reconstruction

Input: the array A computed by the `Knapsack` algorithm with item values $v_1, v_2, \ldots, v_n$, item sizes $s_1, s_2, \ldots, s_n$, and knapsack capacity C.
Output: an optimal knapsack solution.

$S := \emptyset$ `// items in an optimal solution`
$c := C$ `// remaining capacity`
for $i = n$ downto 1 **do**
  if $s_i \le c$ and $A[i-1][c-s_i] + v_i \ge A[i-1][c]$ **then**
    $S := S \cup \{i\}$ `// Case 2 wins, include` i
    $c := c - s_i$ `// reserve space for it`
  `// else skip` i`, capacity stays the same`
return S

1	2	3	4	
3	2	4	4	
4	3	2	3	

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

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.