Ch 16. Greedy Algorithms

Greedy Algorithms

- 각 단계에서 가장 좋을 거라 생각되는 선택을 취함
- 반드시 최적의 해를 구한다고 보장할 수는 없다.

→ greedy-choice property 를 가지는 경우에만 최적해를 구한다.

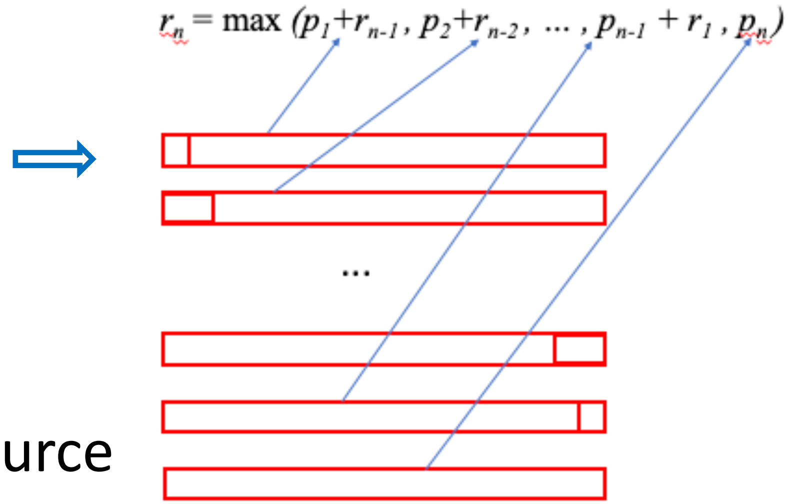
- Examples :

  16.1 activity selection

  16.3 Huffman code

  23. Minimum Spanning Tree algorithms

  24. Dijkstra's algorithm for shortest paths from a single source



rod-cutting problem 은 greedy algorithm 으로 풀 수 없다.

16.1 An activity-selection problem

(시작 시간, 끝나는 시간) 으로 주어진 activity 들이 finish time 의 단조 증가 순으로 정렬되어 있을 때

	i	1	2	3	4	5	6	7	8	9	10	11
시작 시간	s_i	1	3	0	5	3	5	6	8	8	2	12
끝나는 시간	f_i	4	5	6	7	9	9	10	11	12	14	16

활동 시간이 겹치지 않게 compatible activities 의 최대 집합을 찾는 문제

$\{a_3, a_9, a_{11}\}$

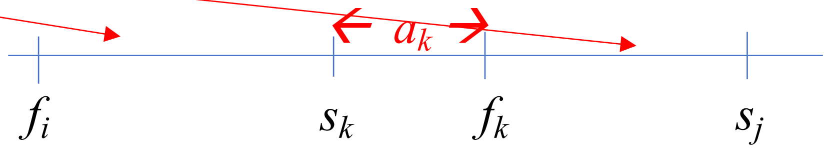
$\{a_1, a_4, a_8, a_{11}\}$

$\{a_1, a_4, a_9, a_{11}\}$

the activity-selection problem exhibits optimal substructure

- S_{ij}: f_i 이후에 시작하고 s_j 이전에 끝나는 a_i 들의 집합
- $c[i,j]$ = size of optimal solution for S_{ij}

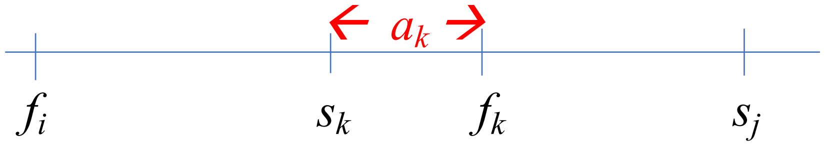
$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset , \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset . \end{cases}$$



the activity-selection problem exhibits optimal substructure

- S_{ij}: f_i 이후에 시작하고 s_j 이전에 끝나는 a_i 들의 집합
- $c[i,j]$ = size of optimal solution for S_{ij}

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \,, \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset \,. \end{cases}$$



- dynamic programming ?
- the greedy choice : a_i with smallest f_i (= the first activity) is always in the solution
- Let $S_k = \{a_i \in S : s_i \geq f_k\}$ a_k 가 끝난 이후에 시작하는 activities 의 집합

→ solution = { a_1, solution of S_1 }

RECURSIVE-ACTIVITY-SELECTOR

Let $S_k = \{a_i \in S : s_i \geq f_k\}$

- returns a maximum-size set of mutually compatible activities in S_k .

size of the original problem

RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)

```
m = k + 1
while m ≤ n and s[m] < f[k]        // find the first activity in S_k to finish
    m = m + 1
if m ≤ n
    return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
else return ∅
```

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

solution = { a_1, solution of S_1 }

RECURSIVE-ACTIVITY-SELECTOR

Let $S_k = \{a_i \in S : s_i \geq f_k\}$

- returns a maximum-size set of mutually compatible activities in S_k .

size of the original problem

RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)

```
m = k + 1
while m ≤ n and s[m] < f[k]     // find the first activity in S_k to finish
    m = m + 1
if m ≤ n
    return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
else return Ø
```

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

solution = $\{a_1$, a_4, solution of S_4 }

RECURSIVE-ACTIVITY-SELECTOR

Let $S_k = \{a_i \in S : s_i \geq f_k\}$

- returns a maximum-size set of mutually compatible activities in S_k.

size of the original problem

RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)

```
m = k + 1
while m ≤ n and s[m] < f[k]      // find the first activity in S_k to finish
    m = m + 1
if m ≤ n
    return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
else return Ø
```

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

solution = $\{a_1, a_4, a_8$, solution of $S_8\}$

RECURSIVE-ACTIVITY-SELECTOR

Let $S_k = \{a_i \in S : s_i \geq f_k\}$

- returns a maximum-size set of mutually compatible activities in S_k .

size of the original problem

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]        // find the first activity in S_k to finish
    m = m + 1
if m ≤ n
    return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
else return Ø
```

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

solution = $\{a_1, a_4, a_8,$ solution of S_8 $\}$

RECURSIVE-ACTIVITY-SELECTOR

Let $S_k = \{a_i \in S : s_i \geq f_k\}$

- returns a maximum-size set of mutually compatible activities in S_k .

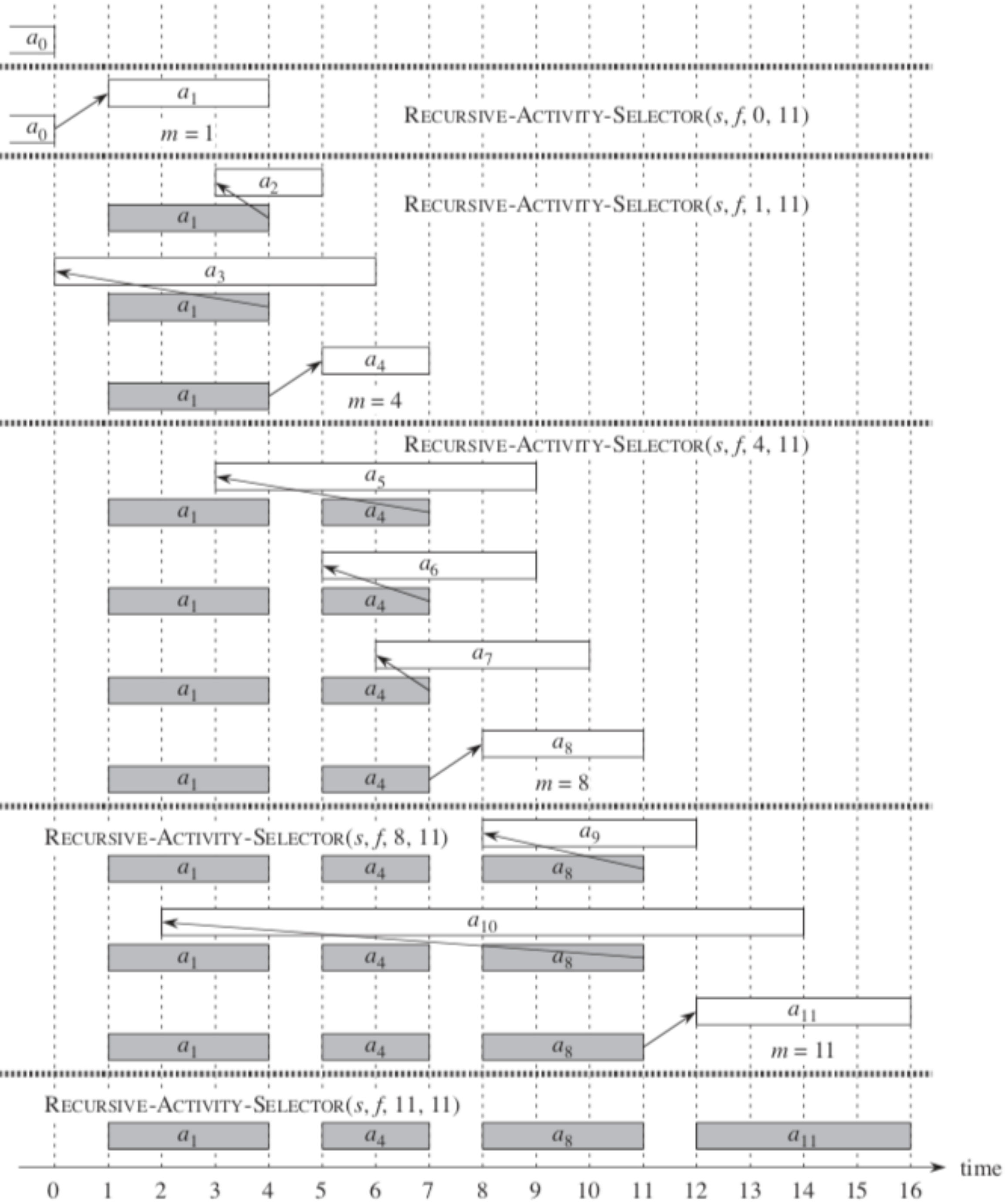
size of the original problem

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
  m = k + 1
  while m ≤ n and s[m] < f[k]      // find the first activity in S_k to finish
        m = m + 1
  if m ≤ n
        return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
  else return Ø
```

solution = $\{a_1, a_4, a_8, a_{11}$, solution of S_{11} }

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

k	s_k	f_k
0	–	0
1	1	4
2	3	5
3	0	6
4	5	7
5	3	9
6	5	9
7	6	10
8	8	11
9	8	12
10	2	14
11	12	16



Iterative function

GREEDY-ACTIVITY-SELECTOR(s, f)

```
n = s.length
A = {a_1}
k = 1
for m = 2 to n
        if s[m] ≥ f[k]
                A = A ∪ {a_m}
                k = m
return A
```

$$= \Theta(n)$$

$m \rightarrow$

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

A = { a_1, solution of S_1 }

A = {a_1, a_4, solution of S_4 }

A = {a_1, a_4, a_8, solution of S_8 }

A = {a_1, a_4, a_8, a_{11} }