Greedy Algorithms
(chapter 16)

مقدمه

- شبیه به dynamic programming.
- این روش هم برای مسائل بهینه‌سازی استفاده می‌شود.

ایده: وقتی که نیاز است که یک انتخاب انجام شود، آن انتخاب که به نظر می‌رسد در حال حاضر بهترین است.

- انتخاب locally optimal choice به امید آن انجام می‌شود که به یک globally optimal solution برسیم.

- Greedy Algorithm همیشه منجر به یک optimal solution نمی‌شود، ولی گاهی این اتفاق می‌افتد.

- در ادامه الگوریتم‌هایی را می‌بینیم که برای آن‌ها greedy منجر به جواب بهینه می‌شود.
- و همچنین نگاهی می‌کنیم به بعضی general characteristics هایی که در آن‌ها greedy algorithm جواب بهینه می‌دهند.

مثال

Activity Selection

n تا activity که نیاز به استفاده انحصاری از یک منبع مشترک دارند.

به عنوان مثال زمان‌بندی استفاده از یک classroom.

set of activities $S = \{a_1, \ldots, a_n\}$

a_i needs resource during period $[s_i, f_i)$

این بازه یک بازه نیم باز است که

s_i = start time

f_i = finish time

Goal : انتخاب بزرگترین مجموعه ممکن که شامل non overlapping activities است

(mutually compatible)

نکته :

- می توانست که این مسئله objective های بسیار دیگری نیز داشته باشد.

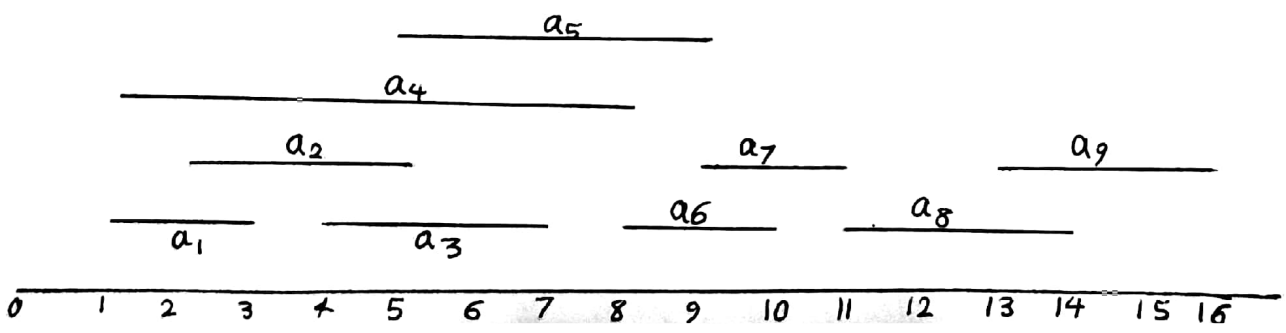
. زمان بندی اتاق برای longest time

. بیشترین درآمد سقف اجاره اتاق

Example : S sorted by finish time

i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16

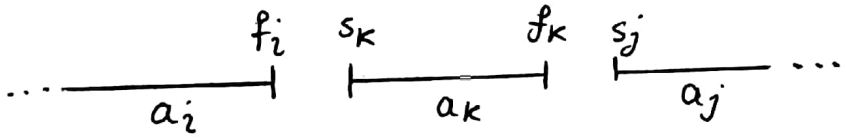
- Maximum-size mutually compatible set: $\{a_1, a_3, a_6, a_8\}$
- Not unique: also $\{a_2, a_5, a_7, a_9\}$



Optimal substructure of activity selection

$$S_{ij} = \{ a_k \in S : f_i \leq s_k < f_k \leq s_j \}$$

= شامل تمام activity هایی که بعد از پایان a_i شروع می‌شوند و قبل از شروع a_j پایان می‌پذیرند.



activity هایی که در S_{ij} هستند، با موارد زیر سازگار هستند

- تمام activity هایی که قبل f_i پایان می‌پذیرند.
- تمام activity هایی که بعد از s_j شروع می‌شوند.

برای نمایش تمام وکلیت مسئله، fictitious activities اضافه می‌کنیم:

$$a_0 = [-\infty, 0)$$

$$a_{n+1} = [\infty, "\infty+1")$$

اهمیتی به $-\infty$ در a_0 یا "$\infty+1$" در a_{n+1} نمی‌دهیم، پس

$$S = S_{0,n+1}$$

Range for S_{ij} is $0 \leq i, j \leq n+1$

فرض کنید که activity ها براساس زمان پایان افزایشی و یکنوا مرتب شده باشند:

$$f_0 \leq f_1 \leq f_2 \leq \cdots \leq f_n \leq f_{n+1}$$

بنابراین:

$$i \geq j \implies S_{ij} = \emptyset$$

- اگر $a_k \in S_{ij}$ وجود داشته باشد.

$$f_i \leq s_k < f_k \leq s_j < f_j \Rightarrow f_i < f_j$$

- امّا

$$i \geq j \Rightarrow f_i \geq f_j \rightarrow \text{تناقض}$$

بنابراین تنها به دنبال S_{ij} هایی هستیم که: $0 \leq i < j \leq n+1$

چون مابقی $S_{ij} = \emptyset$

فرض کنید که یک راه حل برای S_{ij} شامل a_k باشد، دو subproblem دارد:

- S_{ik} شروع بعد از پایان a_i و پایان قبل از شروع a_k
- S_{kj} شروع بعد از پایان a_k و پایان قبل از شروع a_j

راه حل برای S_{ij} = (solution to S_{ik}) $\cup \{a_k\} \cup$ (solution to S_{kj})

تا زمانی که a_k نه در subproblem باشد و نه subproblem ها جدا از هم باشند.

$$|\text{Solution to } S| = |\text{Solution to } S_{ik}| + 1 + |\text{Solution to } S_{kj}|$$

اگر یک جواب بهینه برای S_{ij} شامل a_k باشد باید جوابهای S_{ik} و S_{kj} نیز بهینه باشند. (cut & paste argument) که در این جواب بهینه به کار برده می‌شود نیز بهینه باشد.

Let A_{ij} = optimal solution to S_{ij}

So $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$

فرض می‌کنیم:

- S_{ij} is nonempty
- we know a_k.

Recursive Solution to activity selection

$c[i,j]$ = size of maximum-size subset of mutually compatible activities in S_{ij}.

- $i \geq j \Rightarrow S_{ij} = \emptyset \rightarrow c[i,j] = 0$

اگر $S_{ij} \neq \emptyset$، فرض می‌کنیم که می‌دانیم a_k در زیرمجموعه است، پس:

$$c[i,j] = c[i,k] + 1 + c[k,j]$$

و البته نمی‌دانیم که کدام k مورد استفاده قرار می‌گیرد، بنا براین

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

چرا این range برای k در نظر گرفته شده است؟

چون

$$S_{ij} = \{ a_k \in S : f_i \leq s_k < f_k \leq s_j \}$$

در نتیجه a_k نمی‌تواند a_i یا a_j باشد و همچنین a_k باید در S_{ij} باشد

$$i < k < j$$

از این جا به بعد می‌توانیم مسئله را با استفاده Dynamic programming حل کنیم.

قضیه:

Let $S_{ij} \neq \emptyset$ and let a_m be the activity in S_{ij} with the earliest finish time: $f_m = \min \{ f_k : a_k \in S_{ij} \}$ then:

1. a_m is used in some maximum-size subset of mutually compatible activities of S_{ij}.
2. $S_{im} = \emptyset$, so that choosing a_m leaves S_{mj} are the only nonempty subproblem.