DESIGN AND ANALYSIS OF ALGORITHMS

CS 4120/5120

GREEDY STRATEGY – ACTIVITY SELECTION

AGENDA

- Activity selection problem
- Greedy method

WHO GETS TO SEE THE DENTIST AND WHEN

- Dentist Smileyface is very popular in Hollywood. She routinely receives timeslots from celebrities who want to come in on a certain day.
- Being rich, all celebrities pay her a flat $5000 fee, irrespective of how long they stay.
- Considering they are celebrities, you cannot have more than one in the office at the same time (they bite).

WHO GETS TO SEE THE DENTIST AND WHEN

- Also, considering they are celebrities, they only come in as per their own schedule.
- Example request (on the same day) are shown on the right
  - Some requests overlap
  - All celebrities need to see Dr. Smileface (obviously)
  - They do not stay for the same amount of time.

- How can we help Dr. Smileyface to earn maximum revenue possible?

Tom: [8AM – 2PM]

Meg: [9am – 10AM]

Emily: [11AM-4PM]

Britney: [5PM-520PM]

SCHEDULING FOR RUNNING COMPETITIONS

- The athletic department is trying to schedule *a day of running competitions*. The competitions include potentially the activities shown in the table.
  - Some activities have more contestants than other activities, hence they need more hours

Running Competition	Distance (meter)	Time	Running Competition	Distance (meter)	Time
Sprint	100	8am ~ 9 am	Middle distance	800	11 am ~ 1 pm
	200	9:30am ~ 11am		1500	4 pm ~ 5 pm
Hurdles	100	2 pm ~ 2:30 pm	Relays	4×100	10:30 am ~ 11:10 am
	400	3 pm ~ 3:30 pm		4×400	12 pm ~ 12:40 pm

- Assume two activities can be schedule back-to-back.
- How can we help the Athletic Department schedule as many activities as possible in one day?

EXAMINE THE TWO PROBLEMS

- Fill out the table

Problem	# of Event (Celebs)	Resources required	Goal
Celebrities seeing a dentist	Four celebrities	Dr. SmileyFace	The doctor earns maximum revenue (or see as many patients as possible in a day)
Scheduling running competitions	Eight running competitions	One running track	Schedule maximum-size running competitions in a day
Summary	Several competing events	Exclusive use of a common resource	Select maximum-size activities that do not overlap

ACTIVITY-SELECTION PROBLEM

- The two stories can be modeled as one ***activity-selection problem***.
- In the activity-selection problem, we wish to select a ***maximum-size*** subset of ***mutually compatible*** activities.
- Next, we shall see rigorous definition of the problem.

THE DEFINITION OF ACTIVITY-SELECTION PROBLEM

- Given a set $\boldsymbol{S} = \{\boldsymbol{a_1}, \boldsymbol{a_2}, \dots, \boldsymbol{a_n}\}$ of n proposed activities that wish to **use a resource**.

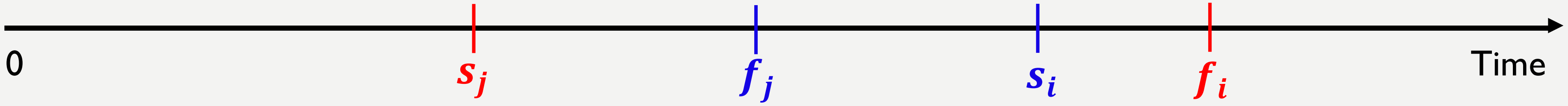
- Each activity a_i has a ***start time*** $\boldsymbol{s_i}$ and a ***finish time*** $\boldsymbol{f_i}$, where $\mathbf{0} \le \boldsymbol{s_i} < \boldsymbol{f_i} < \infty$.
  - If selected, activity a_i takes place during **the half-open time interval** $[s_i, f_i)$.

START AND FINISH TIME

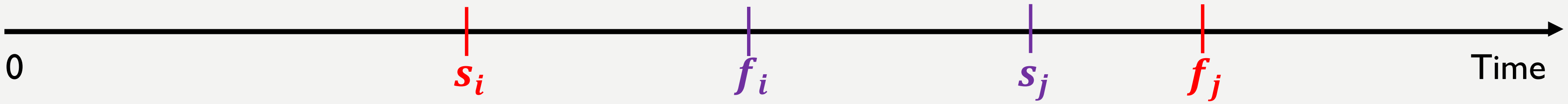
- Example
  - Given $S = \{a_1, a_2, a_3\}$ and their start and finish times
    - $s_1 = 2, f_1 = 6$
    - $\cancel{s_2 = 5, f_2 = 3}$ $s_2 = 3, f_1 = 5$
    - $s_3 = 5, f_3 = 7$
  - The start and finish time of activity $\underline{\quad \boldsymbol{a_2} \quad}$ is NOT plausible.
  - Flip the values of start and finish time of the activity mentioned above.
  - After the flip, activity $\underline{\quad \boldsymbol{a_2} \quad}$ and $\underline{\quad \boldsymbol{a_3} \quad}$ can be scheduled back-to-back.

COMPATIBLE ACTIVITIES

- Activities a_i and a_j are ***compatible*** if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap.
  - That is, a_i and a_j are compatible if
    - $s_i \geq f_j$, indicating that activity _____a_j_____ takes place before _____a_i_____.



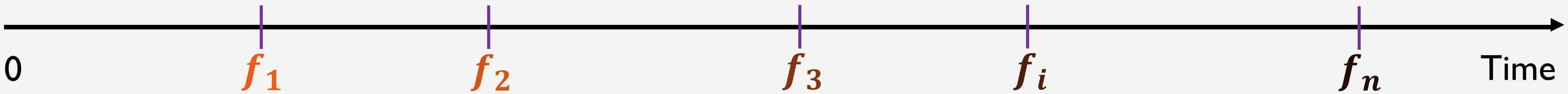
    - $s_j \geq f_i$, indicating that activity _____a_i_____ takes place before _____a_j_____.



THE COMPLETE DEFINITION OF THE ACTIVITY-SELECTION PROBLEM

- Given a set $\boldsymbol{S = \{a_1, a_2, \dots, a_n\}}$ of n proposed activities that wish to **use a resource**. We wish to select a ***maximum-size*** subset of mutually compatible activities.
  - By ***mutually compatible activities***, we mean no two activities in the subset overlap.

- We assume that the activities are ***sorted*** in monotonically ***increasing*** order of finish time, i.e., $f_1 \le f_2 \le f_3 \le \cdots \le f_{n-1} \le f_n$.



THE PROBLEM DEFINITION
PRACTICE #1

- Consider the activities shown in the table.
  - Rearrange (Sort) the activities such that they qualify for the definition of the activity-selection problem.
    - Do not move the indexes when sorting the activities.
    - If two activities finish at the same time, order them by their start times.

i	1	2	3	4	5
s_i	6	2	1	3	4
f_i	9	10	6	9	5

$\boldsymbol{i}$	**1**	**2**	**3**	**4**	**5**
$\boldsymbol{s_i}$					
$\boldsymbol{f_i}$					

THE PROBLEM DEFINITION
PRACTICE #2

i	1	2	3	4	5
s_i	4	1	3	6	2
f_i	5	6	9	9	10

- Consider the activities after the rearrangement.
  - List all ***pairs*** of activities that are compatible. Write the qualifying pairs ***in the form of*** $< a_i, a_j >$.
    - $\boldsymbol{< a_1, a_4 >}$
    - $\boldsymbol{< a_2, a_4 >}$
    - ____________
    - ____________

AN INSTANCE

- Given a list of activities and their respective start and finish time.

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

The activities must be sorted in monotonically ***increasing*** order of their ***finish*** times.

- Reorder the activities such that they can be an instance of the activity-selection problem.
  - The indices must remain where they are.
  - If two activities have the same finish time, order them in increasing order of their start times.

AN INSTANCE REORDERING

- Before

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

- After

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

SOLVING THE PROBLEM FOR THE GIVEN INSTANCE STEP #1

- Goal
  - We wish to find a maximum-size ***subset*** of mutually compatible activities.

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

- To schedule as many activities as possible, we want to first select activity a_1.
- The remaining compatible activities are $a_4, a_6, a_7, a_8, a_9, a_{11}$.
- The next step is finding a maximum-size ***subset*** of mutually compatible activities of $a_4, a_6, a_7, a_8, a_9, a_{11}$

SOLVING THE PROBLEM FOR THE GIVEN INSTANCE STEP #2

- Goal of step #2
  - We wish to find a maximum-size ***subset*** of mutually compatible activities of the remaining activities.

i	1	4	6	7	8	9	11
s_i	1	5	5	6	8	8	12
f_i	4	7	9	10	11	12	16

- To schedule as many activities as possible, we want to first select activity a_4.
- The remaining compatible activities are a_8, a_9, a_{11}.
- The next step is finding a maximum-size ***subset*** of mutually compatible activities of a_8, a_9, a_{11}

SOLVING THE PROBLEM FOR THE GIVEN INSTANCE STEP #3

- Goal of step #3
  - We wish to find a maximum-size ***subset*** of mutually compatible activities of the remaining activities.

i	1	4	8	9	11
s_i	1	5	8	8	12
f_i	4	7	11	12	16

- To schedule as many activities as possible, we want to first select activity ___ a_8 ___.
- The remaining compatible activities are ___ a_{11} ___.
- The next step is finding a maximum-size ***subset*** of mutually compatible activities of a_{11}

SOLVING THE PROBLEM FOR THE GIVEN INSTANCE STEP #4

- Final step
  - Conclude: the maximum-size ***subset*** of mutually compatible activities of $\boldsymbol{S = \{a_1, a_2, \dots, a_{11}\}}$ is

i	1	4	8	11
s_i	1	5	8	12
f_i	4	7	11	16

- Generally
  - We pick the activity with the earliest finish time.
  - Solve the rising subproblem on the remaining activities that are compatible with the chosen one.

ATTEMPTING TO SOLVE BY DP

- Here is a checklist of the qualifications of a DP problem.
  - ❑ Optimization problem
    - ❑ Goal is to find a ***maximum***-size ***subset*** of mutually compatible activities
  - ❑ Two key ingredients
    - ❑ Optimal substructure
    - ❑ Overlapping subproblems

DISCOVERING THE OPTIMAL SUBSTRUCTURE

- **General steps**
  - **Step 1**: A solution to the problem consists of making a choice.
  - **Step 2**: Suppose that for a given problem, you are given the choice that leads to an optimal solution.
  - **Step 3**: Given this choice, you determine which subproblems ensue and how to best characterize the resulting space of subproblems.
  - **Step 4**: Show the solutions to the subproblems used within an optimal solution to the problem must themselves be optimal by using a "cut-and-paste" technique.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 1

- **Step 1**: A solution to the problem consists of making a choice.
- The activity-selection problem can be solved by making a choice.

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

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 2

- **Step 2**: Suppose that for a given problem, you are given the choice that leads to an optimal solution.
  - At this point, you do not concern yourself with how to determine this choice.

- Previously, we selected activity a_1. Based off the selection, we proceeded to solve the problem.

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

  - We **DO NOT** know if picking a_1 leads to an optimal solution.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 2 (CONT'D)

- **Step 2**: Suppose that for a given problem, you are given the choice that leads to an optimal solution.

- Apply abstraction
  - Denote by $\boldsymbol{S_{ij}}$ **the set of activities** that **start after** activity a_i finishes and that ***finish before*** activity a_j starts.
    - The activities included in $\boldsymbol{S_{ij}}$ are highlighted in the table below.

i	i	$i+1$	…	$k-1$	k	$k+1$	…	$j-1$	j
s_i	s_i	s_{i+1}	…	s_{k-1}	s_k	s_{k+1}	…	s_{j-1}	s_j
f_i	f_i	f_{i+1}	…	f_{k-1}	f_k	f_{k+1}	…	f_{j-1}	f_j

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 2 (CONT'D)

- **Step 2**: Suppose that for a given problem, you are given the choice that leads to an optimal solution.

- Apply abstraction
  - In general, for a given set of activities $\boldsymbol{S_{ij}}$. We suppose that we are given a choice $\boldsymbol{a_k}$, $\boldsymbol{i} \le \boldsymbol{k} \le \boldsymbol{j}$, that leads to an optimal solution.

i	i	$i+1$	…	$k-1$	k	$k+1$	…	$j-1$	j
s_i	s_i	s_{i+1}	…	s_{k-1}	s_k	s_{k+1}	…	s_{j-1}	s_j
f_i	f_i	f_{i+1}	…	f_{k-1}	f_k	f_{k+1}	…	f_{j-1}	f_j

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 3

- **Step 3**: Given this choice, you determine which subproblems ensue and how to best characterize the resulting space of subproblems.

- After we include activity a_k in an optimal solution, ________ subproblems arise.
  - # 1: Finding the ***maximum***-size ***subset*** of mutually compatible activities of S_{ik}.
  - # 2: Finding the ***maximum***-size ***subset*** of mutually compatible activities of S_{kj}.

i	i	$i+1$	…	$k-1$	k	$k+1$	…	$j-1$	j
s_i	s_i	s_{i+1}	…	s_{k-1}	s_k	s_{k+1}	…	s_{j-1}	s_j
f_i	f_i	f_{i+1}	…	f_{k-1}	f_k	f_{k+1}	…	f_{j-1}	f_j

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4

- **Step 4**: Show the solutions to the subproblem used within an optimal solution to the problem must themselves be optimal by using a "cut-and-paste" technique.

- We want to show that the ***maximum***-size ***subset*** of mutually compatible activities of S_{ij} includes within itself the following.
  - # 1: The ***maximum***-size ***subset*** of mutually compatible activities of $\underline{\quad \boldsymbol{S_{ik}} \quad}$.
  - # 2: The ***maximum***-size ***subset*** of mutually compatible activities of $\underline{\quad \boldsymbol{S_{kj}} \quad}$.

i	i	$i+1$	...	$k-1$	k	$k+1$	...	$j-1$	j
s_i	s_i	s_{i+1}	...	s_{k-1}	s_k	s_{k+1}	...	s_{j-1}	s_j
f_i	f_i	f_{i+1}	...	f_{k-1}	f_k	f_{k+1}	...	f_{j-1}	f_j

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4 (CONT'D)

- **Step 4**: Show the solutions to the subproblem used within an optimal solution to the problem must themselves be optimal by using a "cut-and-paste" technique.

- Repeatedly saying the ***maximum***-size ***subset*** of mutually compatible activities of $\boldsymbol{S_{ij}}$ can be a hassle. Let us denote such a ***maximum***-size ***subset*** by $\boldsymbol{A_{ij}}$.
  - In other words, $\boldsymbol{A_{ij}}$ is an optimal solution to S_{ij}.

i	i	$i+1$	…	$k-1$	$\boldsymbol{k}$	$k+1$	…	$j-1$	j
s_i	s_i	s_{i+1}	…	s_{k-1}	$s_{\boldsymbol{k}}$	s_{k+1}	…	s_{j-1}	s_j
f_i	f_i	f_{i+1}	…	f_{k-1}	$f_{\boldsymbol{k}}$	f_{k+1}	…	f_{j-1}	f_j

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4 (CONT'D)

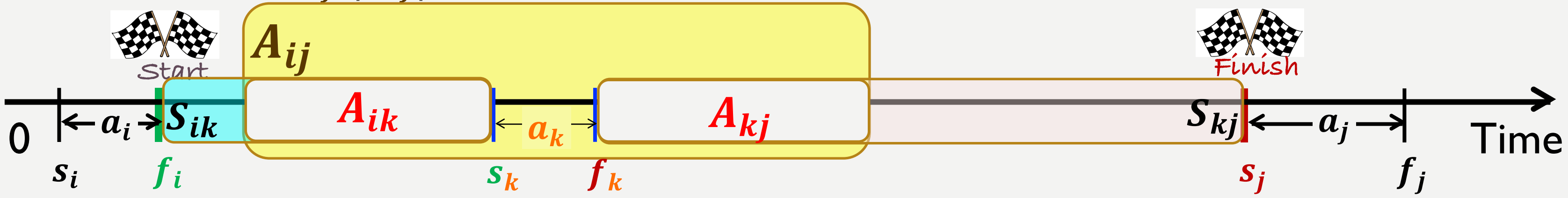
- **Step 4**: Notations.

- The goal of step 4 can be rephrased as showing that ~~the ***maximum*** size ***subset*** of mutually compatible activities of S_{ij}~~ A_{ij} includes within itself the following.
  - # 1: ~~The ***maximum*** size ***subset*** of mutually compatible activities of S_{ik}~~. A_{ik}
  - # 2: ~~The ***maximum*** size ***subset*** of mutually compatible activities of S_{kj}~~. A_{kj}
- In other words, we want to prove that $A_{ij} = \underline{A_{ik}} \cup \underline{\{a_k\}} \cup \underline{A_{kj}}$.
  - The ∪ operator indicates these are set operations.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4 (CONT'D)

- **Step 4**: Proof of optimality of sub-solutions using a "cut-and-paste" technique.

- $\boldsymbol{A_{ij}} = \boldsymbol{A_{ik}} \cup \{\boldsymbol{a_k}\} \cup \boldsymbol{A_{kj}}$
- The goal is to prove that $\underline{\quad A_{ik} \quad}$ and $\underline{\quad A_{kj} \quad}$ are the ***maximum*-size *subset*** of mutually compatible of $\underline{\quad S_{ik} \quad}$ and $\underline{\quad S_{kj} \quad}$, respectively.
  i. **Assume** that A_{ik} is **NOT** an optimal solution in S_{ik}, and that A_{kj} is **NOT** an optimal solution in S_{kj}
     a. Instead, let $A_{ik} = S_{ik} \cap A_{ij}$ and $A_{kj} = S_{kj} \cap A_{ij}$, yielding $A_{ij} = \underline{\quad A_{ik} \quad} \cup \{a_k\} \cup \underline{\quad A_{kj} \quad}$.
     b. The cardinality of A_{ij}, $|A_{ij}| = \underline{\quad |A_{ik}| \quad} + \underline{\quad 1 \quad} + \underline{\quad |A_{kj}| \quad}$.



DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4 (CONT'D)

- **Step 4**: Proof of optimality of sub-solutions using a "cut-and-paste" technique. (Continued)
- The goal is to prove that __ A_{ik} __ and __ A_{kj} __ are the ***maximum***-size ***subset*** of mutually compatible of __ S_{ik} __ and __ S_{kj} __, respectively.

   ii. **There exist an optimal solution**, denoted by A^*_{ik} in S_{ik}, and A^*_{kj} in S_{kj}.

      a. The following relationships hold __ $|A^*_{ik}| > |A_{ik}|$ __ and __ $|A^*_{kj}| > |A_{kj}|$ __.

DISCOVERING THE OPTIMAL SUBSTRUCTURE STEP 4 (CONT'D)

- **Step 4**: Proof of optimality of sub-solutions using a "cut-and-paste" technique. (Continued)

- The goal is to prove that A_{ik} and A_{kj} are the ***maximum***-size ***subset*** of mutually compatible of S_{ik} and S_{kj}, respectively.

  iii. We can **construct a** subset A_{ij}^* as follows, $A_{ij}^* = A_{ik}^* \cup \{a_k\} \cup A_{kj}^*$

  iv. Obviously, A_{ij}^* contains the mutually compatible activities of set S_{ij}, and the cardinality of A_{ij}^*, $|A_{ij}^*| = |A_{ik}^*| + 1 + |A_{kj}^*| > |A_{ij}|$, **contradicting the supposition** that A_{ij} is the ***maximum***-size ***subset*** of mutually compatible activities of S_{ij}.

  v. **Therefore**, A_{ik} is an optimal solution in S_{ik} and A_{kj} is an optimal solution in S_{kj}.

DISCOVERING THE OPTIMAL SUBSTRUCTURE CONCLUSION

- Given a set $\boldsymbol{S} = \{\boldsymbol{a_1}, \boldsymbol{a_2}, \dots, \boldsymbol{a_n}\}$ of n proposed activities that wish to **use a resource**. We wish to select a ***maximum-size*** subset of mutually compatible activities.
- Denote by $\boldsymbol{S_{ij}}$ **the set of activities** that **start after** activity a_i finishes and that ***finish before*** activity a_j starts.
- Let $\boldsymbol{A_{ij}}$ denote the ***maximum***-size ***subset*** of mutually compatible activities of $\boldsymbol{S_{ij}}$.
- The ***optimal substructure***
  - The ***optimal*** solution A_{ij} must also include ***optimal*** solutions to the two subproblems for S_{ik} and S_{kj}, i.e., $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$.

ATTEMPTING TO SOLVE BY DP

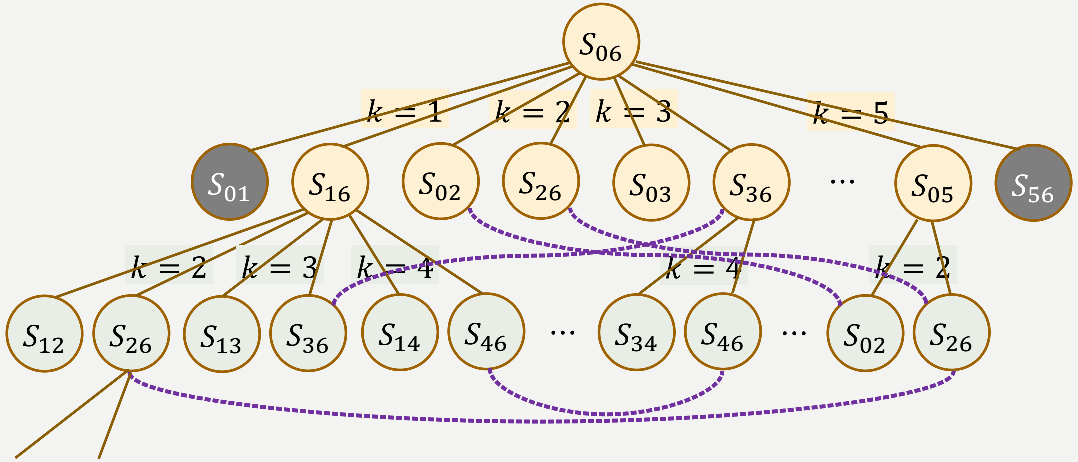
- Here is a checklist of the qualifications of a DP problem.
  - ☑ Optimization problem
    - ☑ Goal is to find a ***maximum***-size ***subset*** of mutually compatible activities
  - ❑ Two key ingredients
    - ☑ Optimal substructure
    - ❑ Overlapping subproblems

DISCOVER OVERLAPPING SUBPROBLEMS

- Draw the corresponding subproblem graph to the input instance

i	1	2	3	4	5
s_i	1	3	0	5	3
f_i	4	5	6	7	9

- Let a vertex S_{ij} represent the size of the (sub)problem.
  – Insert two dummy activities a_0 and a_6



ATTEMPTING TO SOLVE BY DP

- Here is a checklist of the qualifications of a DP problem.
  - ☑ Optimization problem
    - ☑ Goal is to find a ***maximum***-size ***subset*** of mutually compatible activities
  - ☑ Two key ingredients
    - ☑ Optimal substructure
    - ☑ Overlapping subproblems

MOVING FORWARD WITH DYNAMIC PROGRAMMING

- **General steps**
  - **Step 1**: Characterize the structure of an optimal solution
  - **Step 2**: Recursively define the ***value*** of an optimization.
    - The ***value*** of an optimization in the activity-selection problem is the size of an optimal solution for the set S_{ij}. We will denote the size by $c[i, j]$, then we would have the following recurrence

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

  - **Step 3**: Compute the ***value*** of an optimal solution. (Top-down or Bottom-up with ***memo***ization)
  - **Step 4**: Construct the optimal solution from the computed information.

SEEKING A FASTER WAY

- From previous analysis, we see that a DP solution can be developed.
- However, we would be overlooking another important characteristic of the activity-selection problem.

- Consider the input activities.
  - They are ordered in ***monotonically increasing order*** of their finish times.

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

SEEKING A FASTER WAY
THE BEST CHOICE

- Consider the input activities ordered in ***monotonically increasing order*** of their finish times.

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

- Which of the activity would be ***the best*** activity to be ***scheduled first***?
  - We want to choose an activity that leaves the resource **available for as many other activities as possible** as all activities share the same resource.
    - If we pick a_1 (finishes the earliest), we will have a longer period of time available.
- Which of the remaining to schedule after that?

MAKING A GREEDY CHOICE

- In discovering the ***optimal substructure***, we suppose that ***we are given a choice*** that leads to an optimal solution.
- If picking the activity that finishes the earliest is included in the ***optimal*** solution, then we no longer need such a supposition.
- We need to prove the optimality of the greedy choice.

THE GREEDY CHOICE NOTATIONS

- Let us use $S_k = \{a_i \in S: s_i \geq f_k\}$ to denoted the activities that start after a_k finishes.
- Let a_m be an activity in S_k with the ***earliest finish time***.

- Consider the instance below

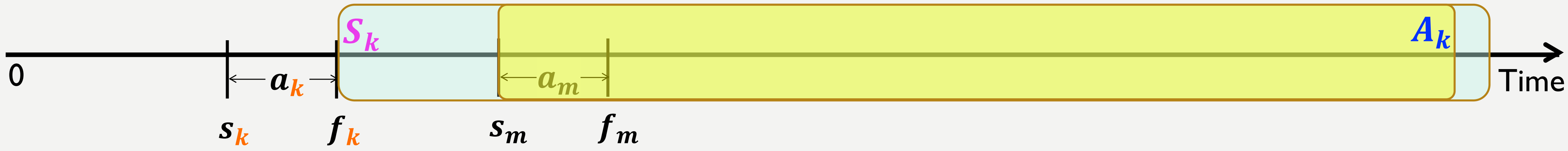
i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

  – $S_1 = \{a_4, a_6, a_7, a_8, a_9, a_{11}\}$, $a_m = a_4$
  – $S_3 = \{a_7, a_8, a_9, a_{11}\}$, $a_m = a_7$
  – $S_{10} = \emptyset$

THE GREEDY CHOICE THEOREM

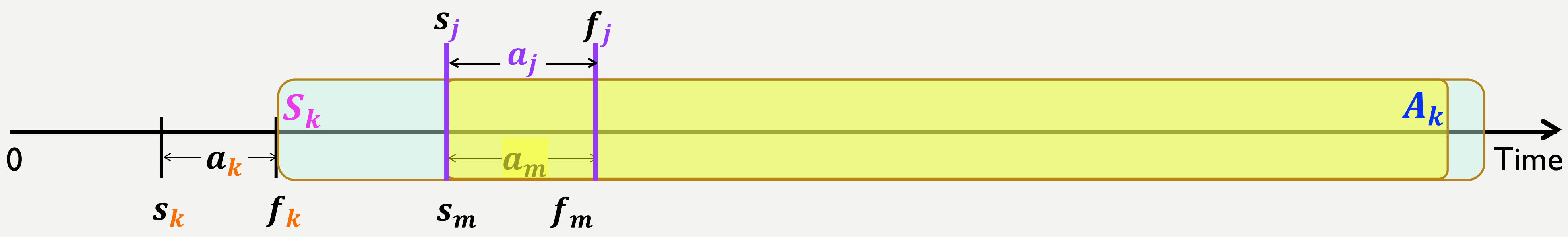
- **Theorem 16.1**
  - Consider any nonempty subproblem S_k, and let $\boldsymbol{a_m}$ *be* an activity in S_k with the ***earliest finish time***. Then $\boldsymbol{a_m}$ ***is included*** in some ***maximum-size subset*** of ***mutually compatible*** activities of S_k.

- Visualization
  - Let $\boldsymbol{A_k}$ **be a maximum-size** subset of mutually compatible activities in S_k.
    - Then we have $\boldsymbol{a_m} \in \boldsymbol{A_k}$. We shall prove this theorem by the "cut-and-paste" technique.



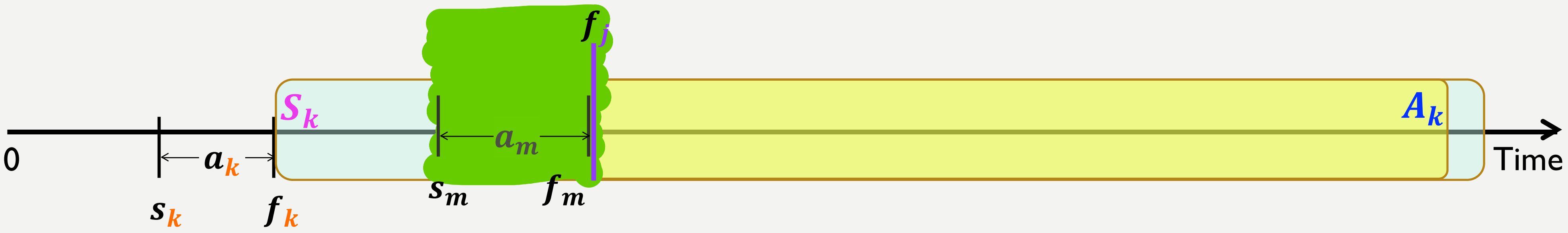
PROOF OF THE GREEDY CHOICE
CASE #1

- Our goal is to show that a_m is included in the optimal solution for S_k.
- Let a_j be the activity **in** A_k with the ***earliest finish time***. Two cases arise
  - Case #1: If $a_j = a_m$ (activity a_j and a_m overlap), done $a_m \in A_k$.



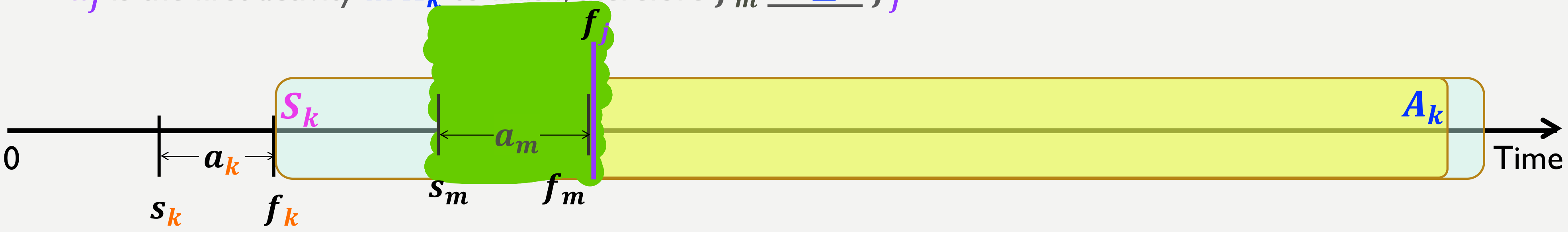
PROOF OF THE GREEDY CHOICE
CASE #2

- Our goal is to show that a_m is included in the optimal solution for S_k.
- Let a_j be the activity **in A_k** with the ***earliest finish time***. Two cases arise.
  - Case #2: Else $a_j \neq a_m$. Let us **construct a new** set of activities $A'_k = (A_k - \{a_j\}) \cup \{a_m\}$.
    - Cut $\{a_j\}$ out of A_k and paste a_m, i.e., substitute a_m for a_j in A_k.



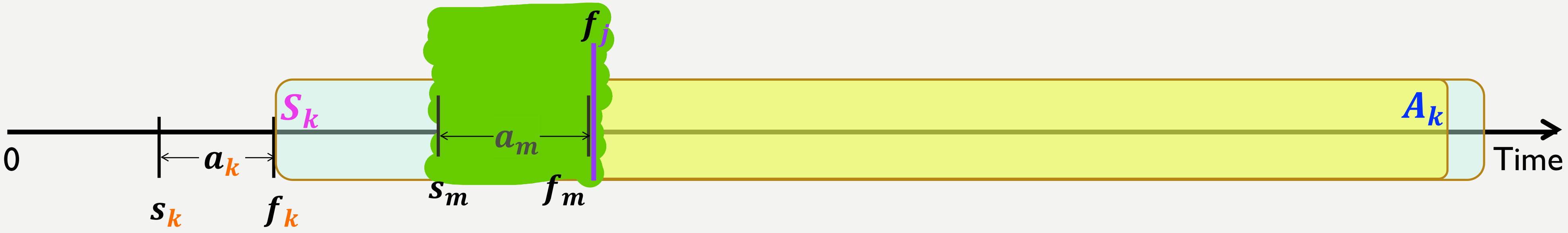
PROOF OF THE GREEDY CHOICE
CASE #2 (CONT'D)

- Our goal is to show that a_m is included in the optimal solution for S_k.
- Let a_j be the activity **in** A_k with the ***earliest finish time***. Two cases arise.
  - Case #2: Else $a_j \neq a_m$. After constructing $A'_k = (A_k - \{a_j\}) \cup \{a_m\}$.
    - The activities in A'_k are disjoint as the activities in A_k are disjoint.
      - Activities in a set being disjoint means that no two activities overlap.
    - a_j is the first activity **in** A_k to finish, therefore f_m __$\leq$__ f_j



PROOF OF THE GREEDY CHOICE
CASE #2 (CONT'D)

- Our goal is to show that a_m is included in the optimal solution for S_k.
- Let a_j be the activity **in** A_k with the ***earliest finish time***. Two cases arise.
  - Case #2: Else $a_j \neq a_m$. After constructing $A'_k = \left(A_k - \{a_j\}\right) \cup \{a_m\}$.
    - $|A'_k| = \underline{\quad |A_k| - 1 + 1 \quad} = \underline{\quad |A_k| \quad}$
    - Conclude case #2, A'_k is a ***maximum-size subset*** of ***mutually compatible activities*** of S_k, and it includes a_m.



CONTINUE TO SOLVE THE PROBLEM

- Design an algorithm that makes the greedy choice every time.
  - Unlike a normal DP solution where the algorithms can be in ***bottom-up*** fashion, a greedy algorithm ***does not need to work bottom-up***.

- Generally, the algorithm can work as follows
  - ***make a greedy choice*** to put into the optimal solution and then
  - ***solve the subproblem*** of choosing activities from those that are compatible with those already chosen.

RECURSIVE-ACTIVITY-SELECTOR

- Input
  - An array $\boldsymbol{s}$ that stores the start times of the activities
  - An array $\boldsymbol{f}$ that stores the finish times of the activities
  - An index $\boldsymbol{k}$ that defines the subproblem $S_{\boldsymbol{k}}$ it is to solve. Use 0 for initial call.
  - The size $\boldsymbol{n}$ of the original problem
- Example: RECURSIVE-ACTIVITY-SELECTOR $(\boldsymbol{s}, \boldsymbol{f}, \mathbf{0}, \mathbf{5})$
  - $\boldsymbol{s} = \{1, 3, 0, 5, 3\}, \boldsymbol{f} = \{4, 5, 6, 7, 9\}$

RECURSIVE-ACTIVITY-SELECTOR $(\boldsymbol{s}, \boldsymbol{f}, \boldsymbol{k}, \boldsymbol{n})$	
1	$m = k + 1$
2	**while** $m \leq n$ **and** $s[m] < f[k]$ // find the first activity in S_k to finish
3	$m = m + 1$
4	**if** $m \leq n$
5	**return** $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n)
6	**else return** $\emptyset$

i	1	2	3	4	5
s_i	1	3	0	5	3
f_i	4	5	6	7	9

RECURSIVE-ACTIVITY-SELECTOR THE USE OF FICTITIOUS ACTIVITY a_0

- The initial call to solve a given problem is RECURSIVE-ACTIVITY-SELECTOR $(\boldsymbol{s}, \boldsymbol{f}, \mathbf{0}, \mathbf{5})$
  - Line 2 accesses $f[\boldsymbol{k}] = f[\mathbf{0}]$ to make a choice
- We add a **fictitious activity** $\boldsymbol{a_0}$ with $\boldsymbol{f_0 = 0}$ to avoid index-out-of-bound exception.

RECURSIVE-ACTIVITY-SELECTOR $(\boldsymbol{s}, f, k, n)$	
1	$m = k + 1$
2	**while** $m \leq n$ **and** $s[m] < f[\boldsymbol{k}]$ // find the first activity in S_k to finish
3	$m = m + 1$
4	**if** $m \leq n$
5	**return** $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n)
6	**else return** $\emptyset$

- The input activities must be sorted in non-decreasing order by their finish times
  - If the given input is NOT sorted, we can spend $O(n \lg n)$ time to sort it.

RECURSIVE-ACTIVITY-SELECTOR IN ACTION

- Run the algorithm on the input sequence.

Push

RECURSIVE-ACTIVITY-SELECTOR $(s, f, 11, 11)$
RECURSIVE-ACTIVITY-SELECTOR $(s, f, 8, 11)$
RECURSIVE-ACTIVITY-SELECTOR $(s, f, 4, 11)$
RECURSIVE-ACTIVITY-SELECTOR $(s, f, 1, 11)$
RECURSIVE-ACTIVITY-SELECTOR $(s, f, 0, 11)$

- Return: $\{\boldsymbol{a_1},\ \boldsymbol{a_4},\ \boldsymbol{a_8},\ \boldsymbol{a_{11}}\}$

RECURSIVE-ACTIVITY-SELECTOR $(\boldsymbol{s}, f, k, n)$

```
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

RECURSIVE-ACTIVITY-SELECTOR RUNNING TIME

- The algorithm examines each activity EXACTLY once.
- The running time function of RECURSIVE-ACTIVITY-SELECTOR algorithm is $T(n) = \underline{\;\Theta(n)\;}$.

RECURSIVE-ACTIVITY-SELECTOR $(\boldsymbol{s}, f, k, n)$
$m = k + 1$
while $m \le n$ **and** $s[m] < f[k]$ // find the first activity in S_k to finish
$\quad m = m + 1$
if $m \le n$
$\quad$ **return** $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n)
else return $\emptyset$

RECURSIVE-ACTIVITY-SELECTOR
TAIL RECURSIVE

- The algorithm is almost "tail recursive."
  - It ends with a recursive call to itself followed by a union operation.

- A ***tail-recursive*** procedure is easy to transform to an ***iterative*** form.

RECURSIVE-ACTIVITY-SELECTOR $(\boldsymbol{s}, f, k, n)$
$m = k + 1$
while $m \leq n$ **and** $s[m] < f[k]$ // find the first activity in S_k to finish
$\quad m = m + 1$
if $m \leq n$
$\quad$ **return** $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR (s, f, m, n)
else return $\emptyset$

ITERATIVE-ACTIVITY-SELECTOR

- Input
  - An array $\boldsymbol{s}$ that stores the start times of the activities
  - An array $\boldsymbol{f}$ that stores the finish times of the activities
- The algorithm takes advantage of the greedy property by including the first activity in set A.
  - The input must be pre-sorted.
  - This way activity a_1 is always the one **with the earliest finish time**.

GREEDY-ACTIVITY-SELECTOR $(\boldsymbol{s}, \boldsymbol{f})$
$n = s.length$
$A = \{a_1\}$
$k = 1$
for $m = 2$ **to** n
if $s[m] \geq f[k]$
$A = A \cup \{a_m\}$
$k = m$
return A

ITERATIVE-ACTIVITY-SELECTOR IN ACTION

- Run the algorithm on the input sequence.

		m									
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										

GREEDY-ACTIVITY-SELECTOR ($\boldsymbol{s}, \boldsymbol{f}$)

```
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
```

- $A = \{\ \boldsymbol{a_1}\ \} \cup \{\ \boldsymbol{a_4}\ \} \cup \{\ \boldsymbol{a_8}\ \} \cup \{\ \boldsymbol{a_{11}}\ \} = \{\boldsymbol{a_1, a_4, a_8, a_{11}}\ \}$

ITERATIVE-ACTIVITY-SELECTOR RUNNING TIME

- The code of the GREEDY-ACTIVITY-SELECTOR $(\boldsymbol{s}, f)$ algorithm is structured as follows.

  for $m = 2$ **to** n
      if $s[m] \geq f[k]$
          $Union$

- Therefore, the running time $T(n) =$ ________
  - Assuming the input activities are sorted and union operation takes $\Theta(1)$.

GREEDY-ACTIVITY-SELECTOR $(\boldsymbol{s}, f)$
$n = s.length$
$A = \{a_1\}$
$k = 1$
for $m = 2$ **to** n
if $s[m] \geq f[k]$
$A = A \cup \{a_m\}$
$k = m$
return A

NEXT UP
HUFFMAN CODES

REFERENCE

- https://www.netclipart.com/isee/hRwxRh_kids-clipart-nurse-cute-female-doctor-cartoon/
- https://listposts.com/lera-kiryakova-celebrities-cartoon-characters/