

LECTURE 09 — GREEDY ALGORITHMS (PART 1)

COMPSCI 308 — DESIGN AND ANALYSIS OF ALGORITHMS

Xing Shi Cai <https://newptcai.gitlab.io>

Duke Kunshan University, Kunshan, China

Jan-Mar 2026

SUMMARY

ITA 16 Greedy Algorithms

ITA 16.1 An Activity-Selection Problem

The Problem

A Dynamic Programming Solution

A Greedy Algorithm

ITA 16.2 Elements of the Greedy Strategy

Greedy Versus Dynamic Programming

ASSIGNMENTS¹



Practice makes perfect!

Introduction to Algorithms (ITA) 
Required Readings:

- Section 16.1.
- Section 16.2.

 Required Exercises:

- Exercises 16.1 – 1–5.
- Exercises 16.2 – 1–5.

¹ ⚪ Assignments will not be collected; however, quiz problems will be selected from them. (This includes both Readings and Exercises.)

ITA 16 GREEDY ALGORITHMS

GREEDY ALGORITHMS

A **greedy algorithm** picks the **best local choice** at each step.

It builds a **solution** by repeating this **choice**.

It is **fast**, but it can miss the **global optimum**.



Figure 1: Greed can be healthy!

ITA 16.1 AN ACTIVITY-SELECTION PROBLEM

ITA 16.1 An Activity-Selection Problem

THE PROBLEM

A FILM FESTIVAL PROBLEM

The screening schedule of six  at a film festival is shown below.

No overlaps

What is the maximum number of films that you can watch in their entirety?

18 - 20:30

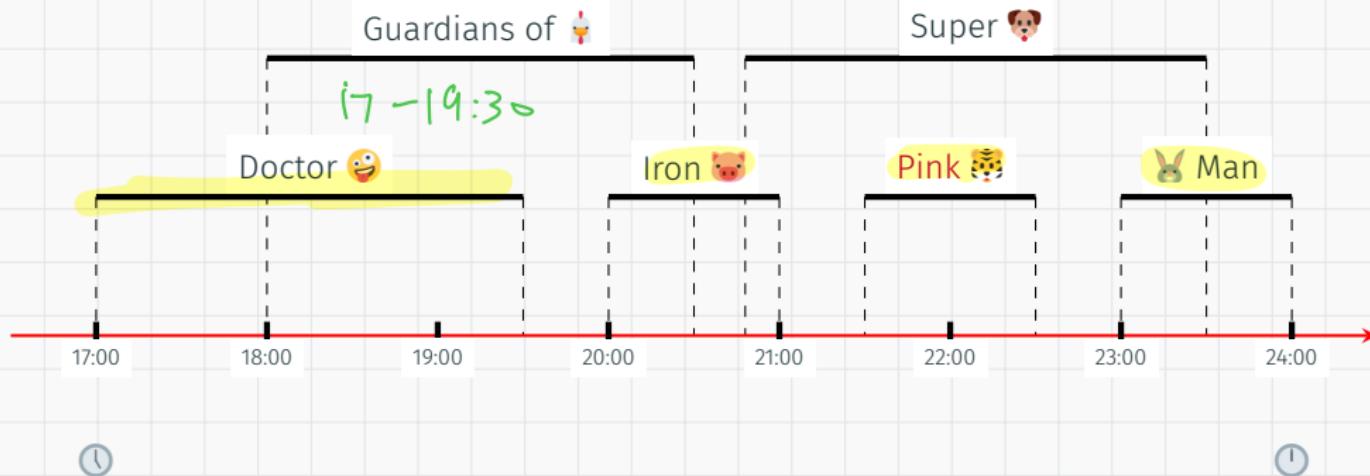
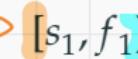


Figure 2: The screening schedule

ACTIVITY-SELECTION PROBLEM

Consider a sequence of n intervals

closed \rightarrow



$\{x : s_i \leq x < f_i\}$

open.

$[s_1, f_1), [s_2, f_2), \dots, [s_n, f_n)$,

finish.

↑
Start

where $0 < s_i < f_i < \infty$ for all $i = 1, \dots, n$.

ACTIVITY-SELECTION PROBLEM

Consider a sequence of n intervals

$$[s_1, f_1), [s_2, f_2), \dots, [s_n, f_n),$$

where $0 < s_i < f_i < \infty$ for all $i = 1, \dots, n$.

Assume also that —

$$f_1 \leq f_2 \leq \dots \leq f_{n-1} \leq f_n.$$

🎂 What should we do if this condition is not met?

We can always sort them by finishing time.

Preprocess

ACTIVITY-SELECTION PROBLEM

Consider a sequence of n intervals

$$[s_1, f_1), [s_2, f_2), \dots, [s_n, f_n),$$

where $0 < s_i < f_i < \infty$ for all $i = 1, \dots, n$.

Assume also that —

$$f_1 \leq f_2 \leq \dots \leq f_{n-1} \leq f_n.$$

- 🎂 What should we do if this condition is not met?
- 🎯 The objective is to find the maximum set of non-overlapping intervals in the sequence.

ITA 16.1 AN ACTIVITY-SELECTION PROBLEM

A DYNAMIC PROGRAMMING SOLUTION

SOME NOTATION



Let

$$S_{i,j} = \{[s_k, f_k) : f_i \leq s_k \text{ and } f_k \leq s_j\}.$$

We treat i and j as sentinel boundaries, so $S_{i,j}$ only includes intervals strictly between them.

Thus, we typically add two sentinel endpoints with $f_0 = 0$ and $s_{n+1} = \infty$.

🎂 Which intervals are in $S_{2,2}$? = \emptyset

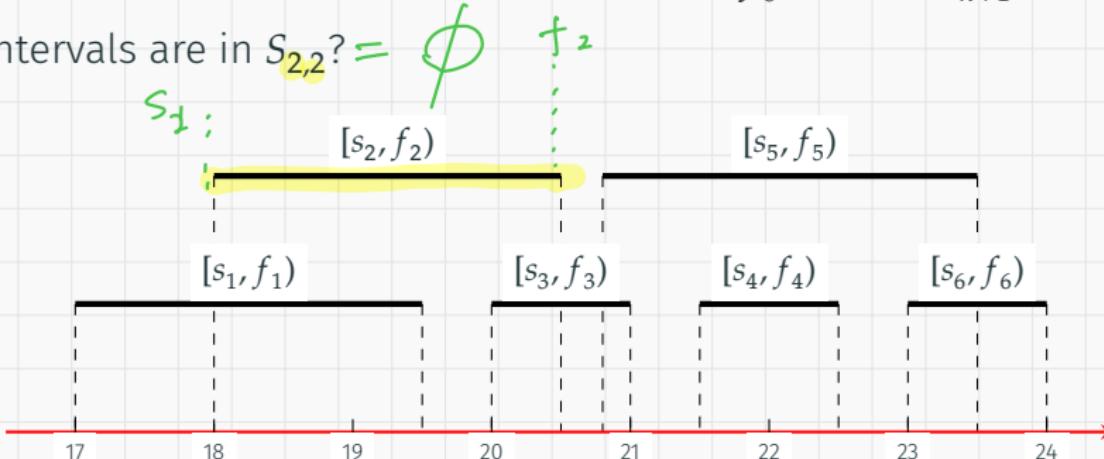


Figure 3: Six intervals

SOME NOTATION

Let

$$S_{i,j} = \{[s_k, f_k) : f_i \leq s_k \text{ and } f_k \leq s_j\}.$$

We treat i and j as sentinel boundaries, so $S_{i,j}$ only includes intervals strictly between them.

Thus, we typically add two sentinel endpoints with $f_0 = 0$ and $s_{n+1} = \infty$.

🎂 Which intervals are in $S_{2,3}$?

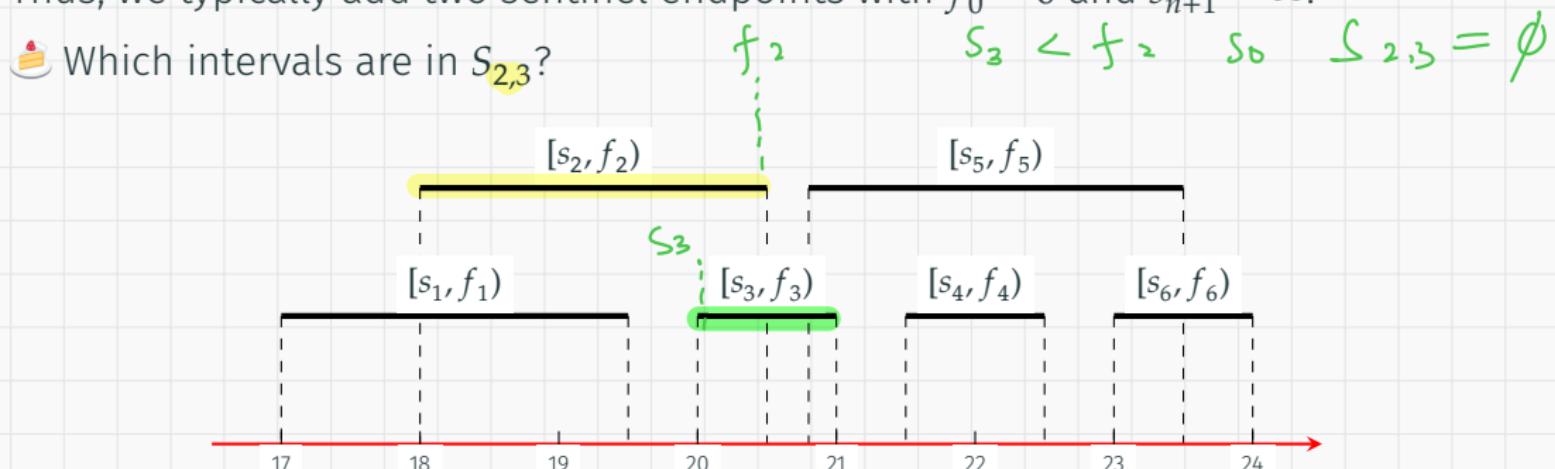


Figure 3: Six intervals

SOME NOTATION

Let

— Note that $(s_i, f_i) \notin S_{ij}$
 $(s_j, f_j) \notin S_{ij}$.

$$S_{i,j} = \{(s_k, f_k) : f_i \leq s_k \text{ and } f_k \leq s_j\}.$$

We treat i and j as sentinel boundaries, so $S_{i,j}$ only includes intervals strictly between them.

Thus, we typically add two sentinel endpoints with $f_0 = 0$ and $s_{n+1} = \infty$.

🎂 Which intervals are in $S_{1,6}$? $= \{[s_3, f_3), [s_4, f_4)\}$

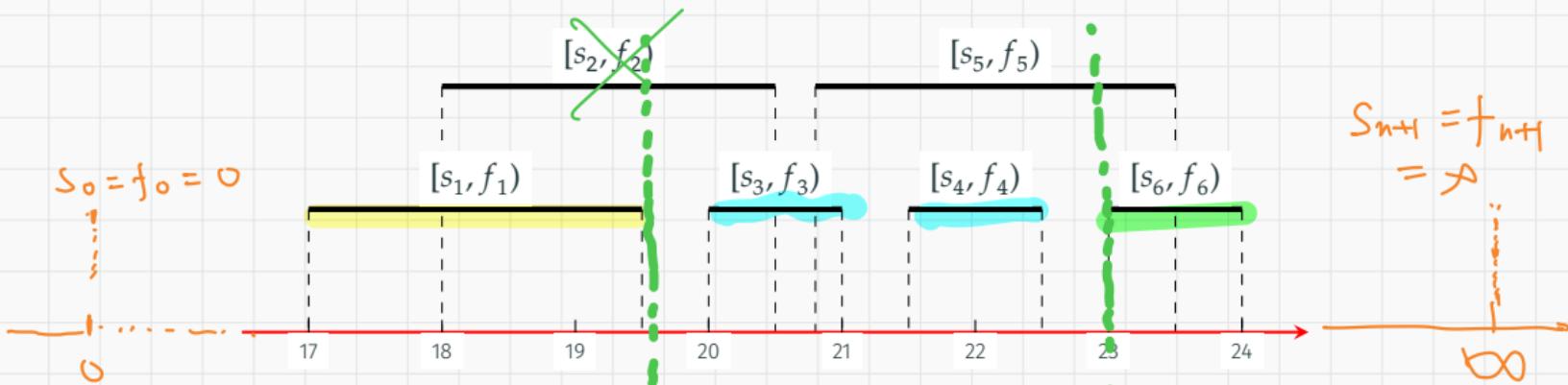


Figure 3: Six intervals

What $s_0, f_7 =$ all of the six

SOME NOTATION

Let

$$S_{i,j} = \{[s_k, f_k) : f_i \leq s_k \text{ and } f_k \leq s_j\}.$$

We treat i and j as sentinel boundaries, so $S_{i,j}$ only includes intervals strictly between them.

Thus, we typically add two sentinel endpoints with $f_0 = 0$ and $s_{n+1} = \infty$.

🎂 Which intervals are in $S_{2,6}$? = { $[s_4, f_4)$ }.

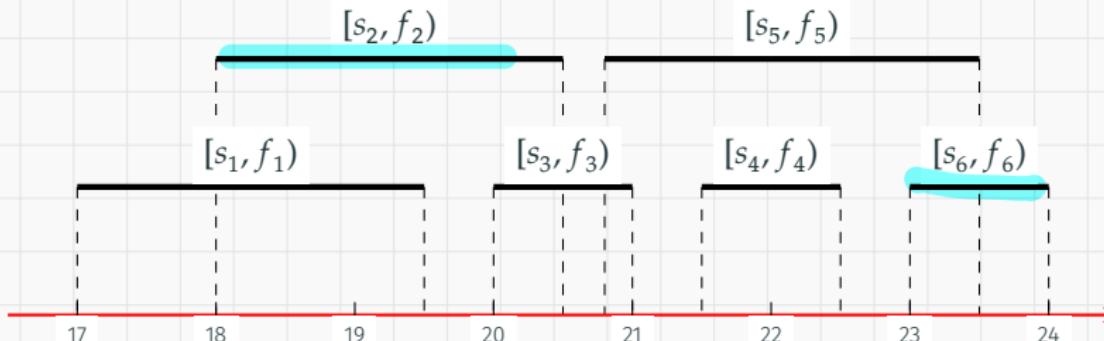


Figure 3: Six intervals

A DYNAMIC PROGRAMMING SOLUTION

Let $A_{i,j}$ be the maximum subset of non-overlapping intervals in $S_{i,j}$.

💡 Key observation — If $A_{i,j}$ includes $[s_k, f_k)$, then

$$A_{i,j} = A_{i,k} \cup \{[s_k, f_k)\} \cup A_{k,j},$$

where the union is over three disjoint sets.

🎂 Does $[s_i, f_i)$ or $[s_j, f_j)$ belong to $A_{i,j}$?



A DYNAMIC PROGRAMMING SOLUTION

Let $A_{i,j}$ be the maximum subset of non-overlapping intervals in $S_{i,j}$.

💡 Key observation — If $A_{i,j}$ includes $[s_k, f_k)$, then

$$A_{i,j} = A_{i,k} \cup \{[s_k, f_k)\} \cup A_{k,j},$$

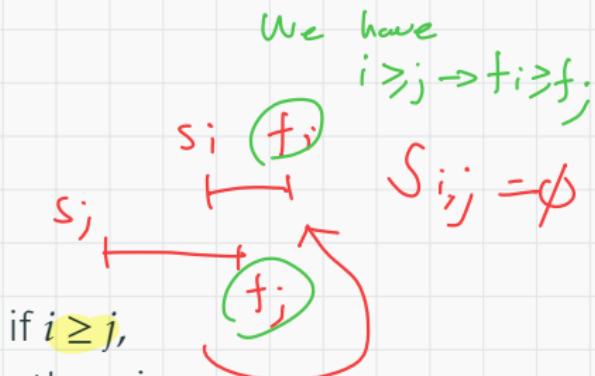
where the union is over three disjoint sets.

Thus, letting $c[i,j] = |A_{i,j}|$, we have the recursion —

$$c[i,j] = \begin{cases} 0 & S_{i,j} \\ \max_{[s_k, f_k) \in S_{i,j}} c[i,k] + 1 + c[k,j] & \downarrow \text{if } i \geq j, \\ & \text{otherwise.} \end{cases}$$

This naturally leads to a *dynamic programming* algorithm.

The answer to our original question: $c[0, n]$.



$c[0, n+1]$

~~$c[1, n]$~~

ITA 16.1 An Activity-Selection Problem

A GREEDY ALGORITHM

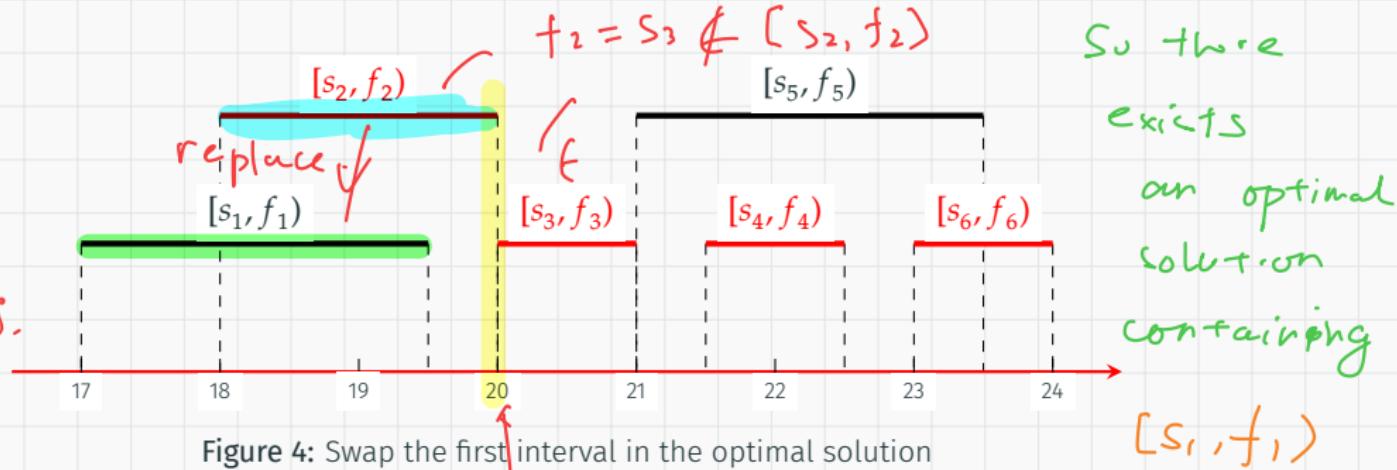
A KEY OBSERVATION

In the maximum subset of non-overlapping intervals, there exists one interval that finishes first.

Note that we can replace it by $[s_1, f_1)$ while still having an optimal solution.

Reason:

$f_1 \leq f_i$
for any
 $i \in \{1, \dots, n\}$.



No overlapping -

A KEY OBSERVATION

In the maximum subset of non-overlapping intervals, there exists one interval that finishes first.

Note that we can replace it by $[s_1, f_1)$ while still having an optimal solution.

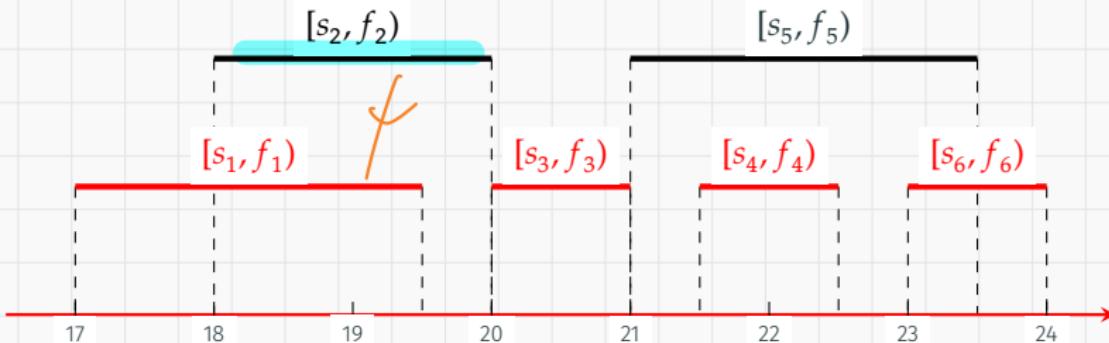


Figure 4: Swap the first interval in the optimal solution

THEOREM 16.1

Let $f_0 = 0$ and

$$S_k = \{[s_i, f_i) : s_i \geq f_k\}, \quad \text{for all } k = 0, \dots, n.$$

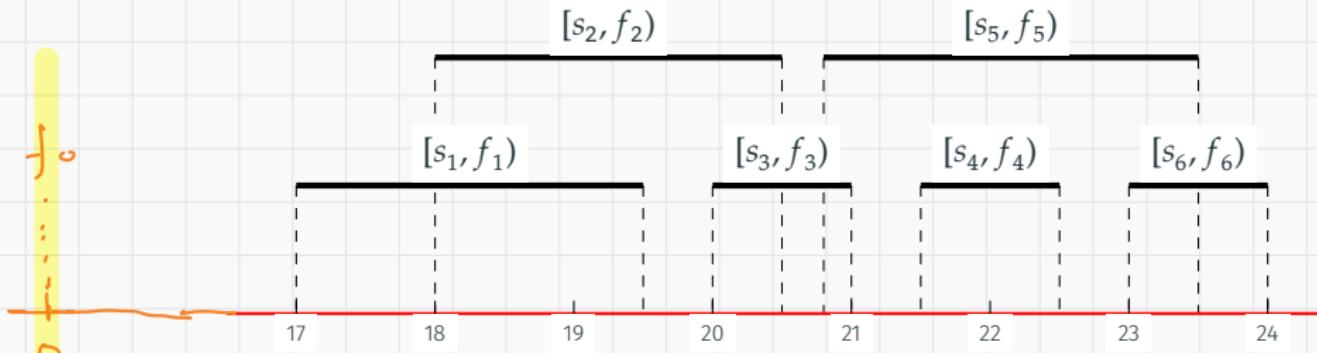


Figure 5: 🎂 What is S_0 ?

↑
All six intervals.

THEOREM 16.1

Let $f_0 = 0$ and

$$S_k = \{[s_i, f_i) : s_i \geq f_k\}, \quad \text{for all } k = 0, \dots, n.$$

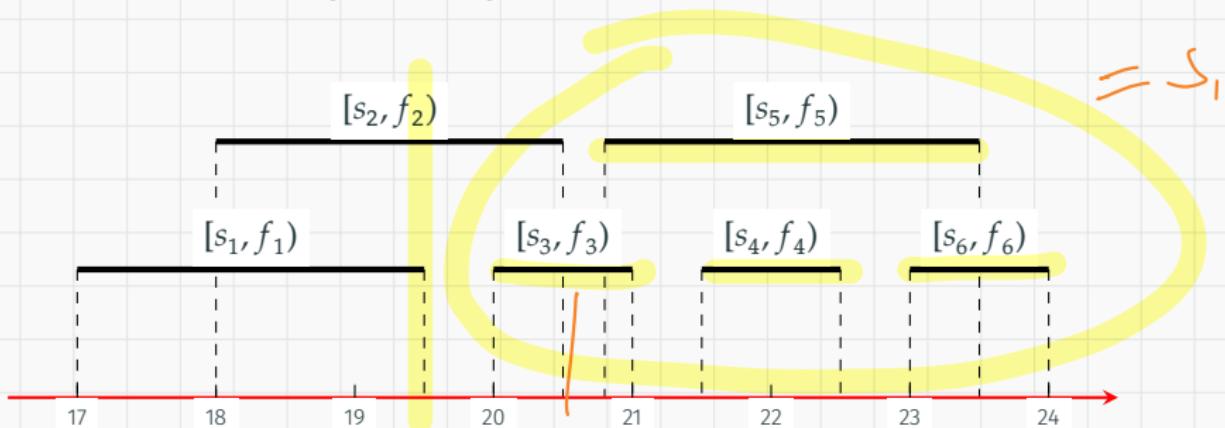


Figure 5: 🎂 What is S_1 ?

THEOREM 16.1

Let $f_0 = 0$ and

$$S_k = \{[s_i, f_i) : s_i \geq f_k\}, \quad \text{for all } k = 0, \dots, n.$$

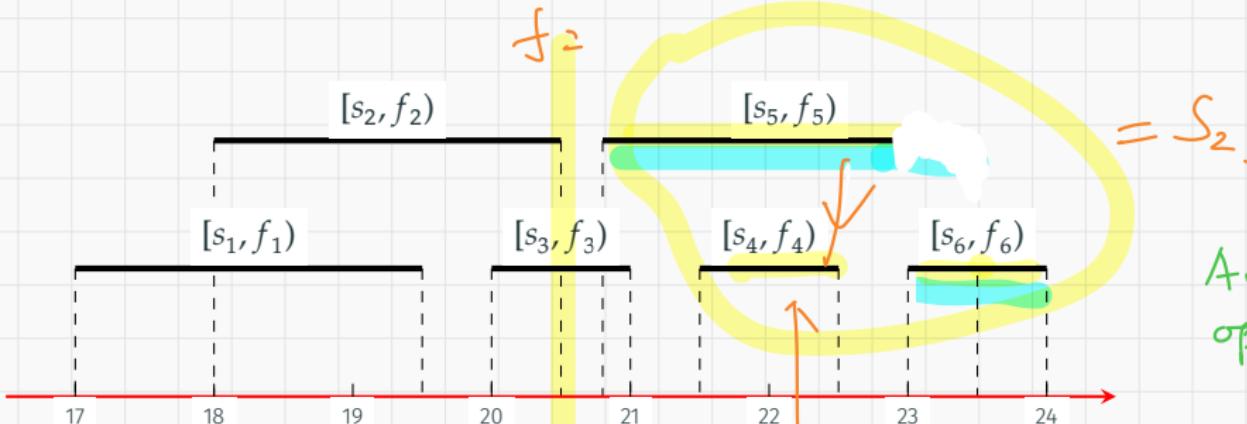


Figure 5: 🎂 What is S_2 ?

$[s_m, f_m)$.

Both (s_5, f_5) ; (s_6, f_6)

(s_4, f_4) , (s_6, f_6)

and



THEOREM 16.1

Let $f_0 = 0$ and

$$S_k = \{[s_i, f_i) : s_i \geq f_k\}, \quad \text{for all } k = 0, \dots, n.$$

Let $[s_m, f_m)$ be the interval that finishes first in S_k .

🎂 Is it true that we always have $[s_m, f_m) = [s_{k+1}, f_{k+1})$?

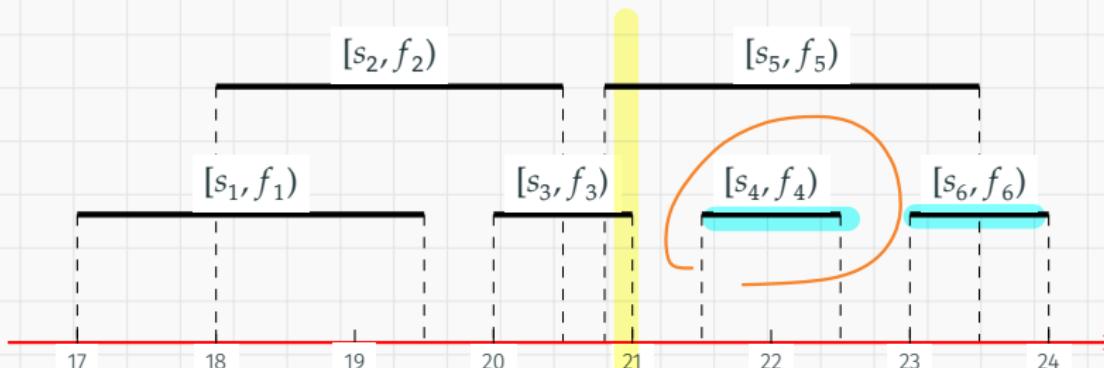


Figure 5: 🎂 What is $[s_m, f_m)$ for S_3 ?

For S_3 ,
 $[s_m, f_m) = [s_4, f_4)$

THEOREM 16.1

Let $f_0 = 0$ and

$$S_k = \{[s_i, f_i) : s_i \geq f_k\}, \quad \text{for all } k = 0, \dots, n.$$

Let $[s_m, f_m)$ be the interval that finishes first in S_k .

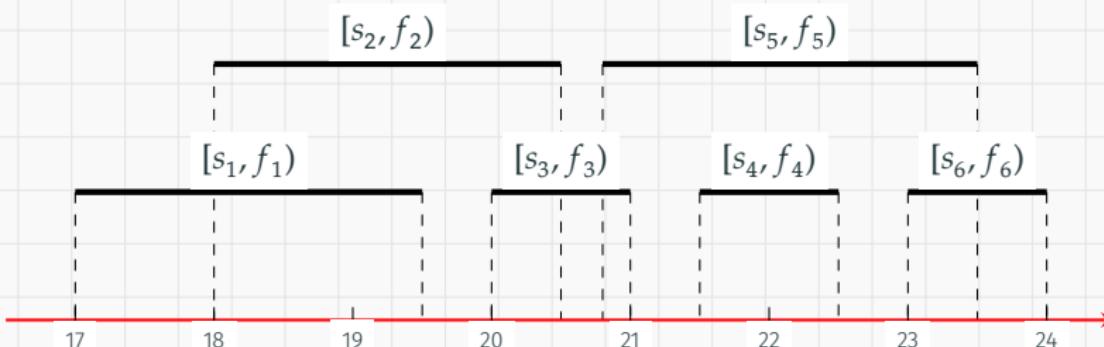


Figure 5: 🎂 What is $[s_m, f_m)$ for S_4 ?

THEOREM 16.1

Let $f_0 = 0$ and

$$S_k = \{[s_i, f_i) : s_i \geq f_k\}, \quad \text{for all } k = 0, \dots, n.$$

Let $[s_m, f_m)$ be the interval that finishes first in S_k .

Then *one of* the maximum subsets of S_k containing non-overlapping intervals includes $[s_m, f_m)$.

Proof Sketch — Let A_k be the maximum subset of S_k containing non-overlapping intervals.

Replace the earliest-finish interval in A_k by $[s_m, f_m)$.

Then we get another maximum subset of non-overlapping intervals.

A GREEDY ALGORITHM

Recall that the activities are sorted by finish time, with $f_0 = 0$ as a sentinel.

Theorem 16.1 leads to a greedy algorithm –

```
1: Recursive-Selector( $s, f, k, n$ )
2:  $m \leftarrow k + 1$ 
3: while  $m \leq n$  &  $s[m] < f[k]$  do
4:    $m \leftarrow m + 1$ 
5: if  $m \leq n$  then
6:   return  $\{[s_m, f_m]\} \cup \text{RECURSIVE-SELECTOR}(s, f, m, n)$ 
7: else
8:   return  $\emptyset$ 
```

🎂 What is the ⏳ complexity of $\text{RECURSIVE-SELECTOR}(s, f, 0, n)$?

AN ITERATIVE IMPLEMENTATION

$$A = \{ [s_1, f_1], [s_3, f_3], [s_4, f_4], [s_6, f_6] \}$$

We can implement the same rule iteratively —

Greedy-Selector(s, f)

$n \leftarrow \text{length}(s)$

$A \leftarrow \{[s_1, f_1]\}$ *Take interval. 1*

$k \leftarrow 1$

for $m \leftarrow 2$ to n do

if $s[m] \geq f[k]$ then

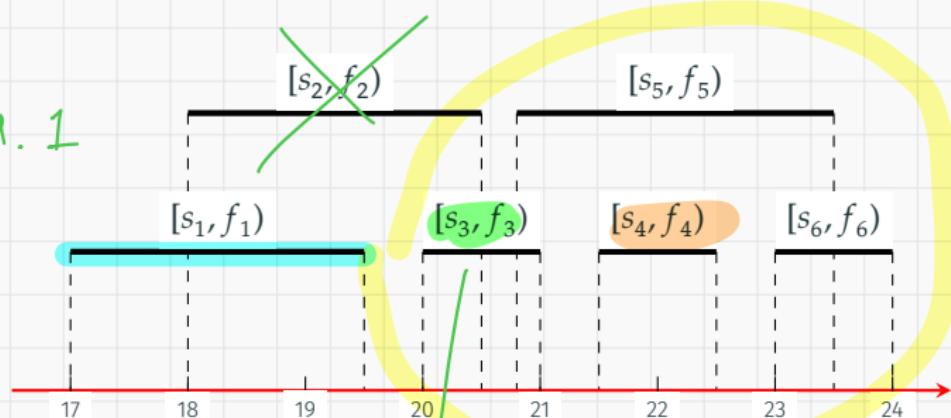
$A \leftarrow A \cup \{[s_m, f_m]\}$

$k \leftarrow m$

return A

Run-time $\Theta(n)$

Let us try it out —

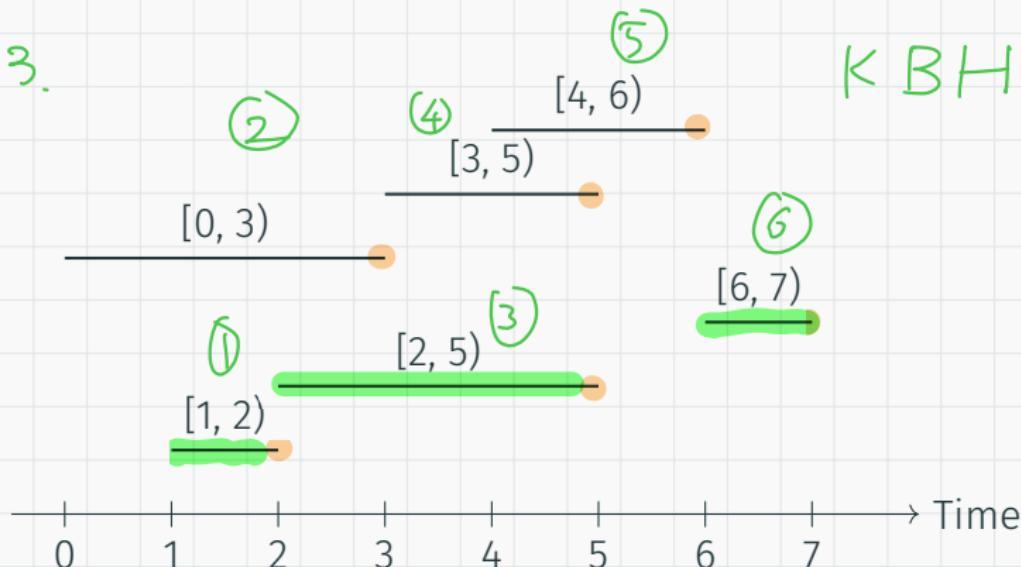


(s_3, f_3) is the $[s_m, f_m]$ for s_1

What is the maximum number of non-overlapping intervals among the following 6 intervals?

Answer: 3.

K BH SMS



TIME FOR TOILET

Suppose that we also require that there is at least **15** minutes between two screenings that we select.

🍰 How should we modify our greedy algorithm?



Figure 5: Rushing to 🚽 between films

ITA 16.2 ELEMENTS OF THE GREEDY STRATEGY



GREEDY ALGORITHMS

General recipe for a greedy algorithm –

1. Formulate the problem so one **greedy choice** leaves a single **subproblem**.
2. Prove optimal substructure: **greedy choice + optimal subproblem solution is optimal.**



GREEDY ALGORITHMS

General recipe for a greedy algorithm —

1. Formulate the problem so one greedy choice leaves a single subproblem.
2. Prove optimal substructure: greedy choice + optimal subproblem solution is optimal.

Such problems have —

- Greedy-choice Property — A locally optimal (greedy) choice leads towards a globally optimal solution.
- Optimal Substructure — An optimal solution contains optimal solutions to subproblems.

Also for dynamic programming

ITA 16.2 ELEMENTS OF THE GREEDY STRATEGY

GREEDY VERSUS DYNAMIC PROGRAMMING

◆ THE 0-1 KNAPSACK PROBLEM

A 🐰 who is robbing a store finds n items.

The i -th item is worth v_i 💵 and weighs w_i kg, where v_i and w_i are integers.

The 🐰 can carry at most W kg in his 🦑.

How can the 🐰 maximize his revenue?



Figure 6: A 🐰 robbing a store

🧀 THE FRACTIONAL KNAPSACK PROBLEM

In the fractional knapsack problem, we allow the 🐰 to take fractions of the items.

💡 Think of the 0-1 version as robbing a 💎 store, and the fractional version as robbing a 🧀 store.



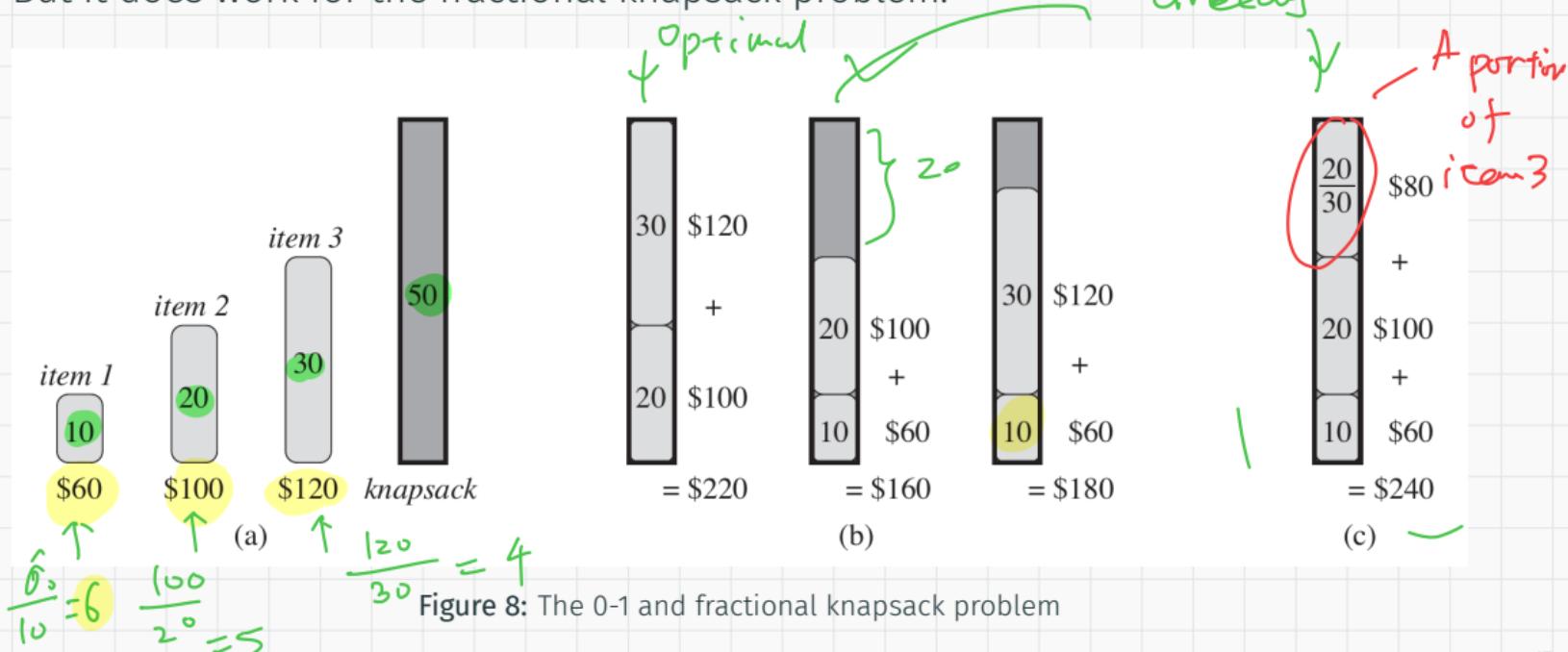
Figure 7: A 🐰 robbing a store



GREEDY ALGORITHM OR DYNAMIC PROGRAMMING?

As the example below shows, the algorithm does not always work for the 0-1 knapsack problem.

But it does work for the fractional knapsack problem.



SOLUTION FOR THE 0-1 KNAPSACK PROBLEM

Let us use dynamic programming to solve the 0-1 knapsack problem –

Knapsack(v, w, n, W)

Initialize $K[0 \dots n][0 \dots W]$

for $i \leftarrow 0$ to n do

 for $j \leftarrow 0$ to W do

 if $i = 0$ or $j = 0$ then

$K[i][j] \leftarrow 0$

 else if $w[i] \leq j$ then

$K[i][j] \leftarrow \max \{ v[i] + K[i-1][j-w[i]], K[i-1][j] \}$, and capacity

 else

$K[i][j] \leftarrow K[i-1][j]$.

return $K[n][W]$

$K[i][w]$ is the maximum value we can get
No item or No capacity given the first i items.

Can we take item i

$K[i-1][j]$

↑
Not taking item i

SOLUTION FOR THE FRACTIONAL KNAPSACK PROBLEM

But for the fractional knapsack problem, we can use a  algorithm –

Knapsack(v, w, n, W)

$V \leftarrow 0$

▷ Total value initialized to 0

SORTBYRATIO(v, w)

▷ Sort by value-to-weight ratio in decreasing order

for $i \leftarrow 1$ to n **do**

we can take all of item i

if $w[i] \leq W$ **then**

$V \leftarrow V + v[i]$ *— take i*

▷ Take the whole item

$W \leftarrow W - w[i]$ *— reduce capacity* **▷ Decrease available weight**

else

$V \leftarrow V + \frac{W}{w_i} \cdot \frac{v_i}{w_i}$

▷ Take fraction of the item

break



▷ Return the total value

return V

Capacity left +

Take a fraction of item i

How should we modify **Knapsack** (fractional version), if item i has only $u_i \leq w_i$ weight available?

- 💡 Treat item i as having weight u_i and value $v_i \cdot (u_i/w_i)$, then run greedy as usual.

Knapsack(v, w, n, W)

$V \leftarrow 0$

SORTBYRATIO(v, w)

for $i \leftarrow 1$ to n do

 if $w[i] \leq W$ then

$V \leftarrow V + v[i]$

$W \leftarrow W - w[i]$

 else

$V \leftarrow V + v[i] \cdot W/w[i]$ ▷ Take fraction

$W/w[i]$

 break

return V

Knapsack-Cap(v, w, u, n, W)

for $i \leftarrow 1$ to n do

$v[i] \leftarrow v[i] \cdot \underline{\hspace{1cm}}$ ▷ Scale

 value

$w[i] \leftarrow \underline{\hspace{1cm}}$ ▷ Cap weight

return KNAPSACK(v, w, n, W)

 EXERCISE

How should we modify Knapsack (fractional version), if we may take from at most m items (fractions allowed)?

💡 Sort by v_i/w_i , but stop after taking from m items.

Knapsack(v, w, n, W)

$V \leftarrow 0$

SORTBYRATIO(v, w)

for $i \leftarrow 1$ to n do

 if $w[i] \leq W$ then

$V \leftarrow V + v[i]$

$W \leftarrow W - w[i]$

 else

$V \leftarrow V + v[i] \cdot W/w[i]$ ▷ Take fraction

$W/w[i]$

 break

return V

Knapsack-At-Most-M(v, w, n, W, m)

$V \leftarrow 0$

$used \leftarrow 0$

SORTBYRATIO(v, w)

for $i \leftarrow 1$ to n do

 if _____ then

 return V

 ▷ Stop

$x \leftarrow _____$

$V \leftarrow _____$

$W \leftarrow W - x$

$used \leftarrow used + 1$

return V

💬 GREEDY ALGORITHM FOR THE KNAPSACK PROBLEM?

We have seen that being 💰 in the fractional version of the knapsack problem is **OK**.

… Can you think of some other **variations** of the **knapsack problem** for which being 💰 is *also* **OK**?



Figure 9: It is **OK** to be 💰 when the 🐉 is 🤑



THE CHANGE-MAKING PROBLEM

We have infinitely many with denominations in $D = \{d_1, \dots, d_n\}$ where

$$d_1 < d_2 \dots < d_n.$$

We want to make a specified amount with these coins while minimizing the number of .

Example – $A = 89$ and

$$D = \{2, 5, 11, 23, 50\},$$

can a greedy algorithm work?

$$89 = \underline{\underline{50 + 23 + 11 +}} \boxed{5}$$



THE CHANGE-MAKING PROBLEM

We have infinitely many with denominations in $D = \{d_1, \dots, d_n\}$ where

$$d_1 < d_2 \dots < d_n.$$

We want to make a specified amount A with these coins while minimizing the number of .

If we take the largest coin first at each step,

$$89 = \underline{\hspace{2cm}}$$

This uses _____ coins (and is optimal). (Why?)



THE CHANGE-MAKING PROBLEM

We have infinitely many



with denominations in $D = \{d_1, \dots, d_n\}$ where

$$d_1 < d_2 \dots < d_n.$$

We want to make a specified amount A with these coins while minimizing the number of

What if $A = 51$ with the same denominations?

$$51 = \underline{50 + ?}$$

↑

Does greedy still work?

Cannot finish.

Prove that if it works, then the solution is optimal.

Greedy algorithm for this problem does not always solve it.

WHAT ARE YOUR MAIN TAKEAWAYS TODAY? ANY QUESTIONS?

