CS3230
eric_han@nus.edu.sg
https://eric-han.com
Computer Science

T09 – Week 10

Greedy Algorithms

CS3230 – Design and Analysis of Algorithms

Admin

1. Midterm scripts returned via Softmark and the time for script review had passed.
2. No Monday Week 11 tutorials (TG01-09, TG19) due to Hari Raya Puasa. Please watch the pre-recorded tutorial instead, available from Monday, 9am on Canvas > Videos/Panopto > Tutorials. All Tuesday tutorials will proceed as scheduled.
3. Free attendance is granted for all Week 11 Monday sessions. (Tuesday sessions had received their free attendance in Week 3.)

Lecture Review

Greedy Algorithms solve problems by making a **greedy choice** and solving the **remaining subproblem**.

1. **Prove**[1] that an optimal solution always makes the greedy choice (often using an exchange argument).
2. Use **optimal substructure** to show that combining the **greedy choice** with an optimal subproblem solution yields an optimal solution (often by contradiction).

This tutorial covers more greedy algorithm examples beyond those in lectures.

[1] In theory courses like CS3230, this is crucial. In programming competitions, ‘faith’ in their ‘greedy choice’, implement it, and hope it passes.

Burning CDs

Bob wants to burn music files onto CDs[2]:

- Each CD has **100 MB** capacity.
- At most **two** files per CD[3].
- Files **cannot** be split across CDs.
- Given a set A of file sizes ($<$ 100 MB), define $MinCD(A)$ as the minimum number of CDs needed.

Bob wants to use the least number of CDs to fit his music collection.

[2] If unfamiliar, see this guide.
[3] A necessary constraint for a greedy solution.

Question 1 [G]/[P1]

Which of the following correctly describes an optimal substructure property of the problem, assuming that **at least one pair of files fit into a CD**?

a. For *any pair* of files f_1, f_2 in A,

b. For any pair of files f_1, f_2 in A that belong on a single CD in an optimal solution,

c. If f_1 and f_2 are the **largest** and **smallest** files in A,

$$MinCD(A) = 1 + MinCD(A \backslash \{f_1, f_2\}) \tag{1}$$

Answer 1a

This approach fails—we **cannot** simply pair *any two* files and expect an optimal solution.

Counterexample: Given $A = \{10, 20, 80, 90\}$,

- **Optimal solution:** $\{10, 90\}, \{20, 80\} \rightarrow$ **2 CDs**
- Using eq. 1 with **wrong pairing** $(f_1 = 10,\ f_2 = 20)$ gives:

$$1 + MinCD(\{80, 90\}) = 1 + 2 = 3,$$

which is incorrect since $\{80, 90\}$ needs **2 CDs**.

Answer 1b

This is correct. In any optimal solution:

1. Remove any pair stored in the same CD in the optimal solution (say *first* CD).
2. The remaining files must be stored optimally—otherwise, a better allocation exists, contradicting optimality.

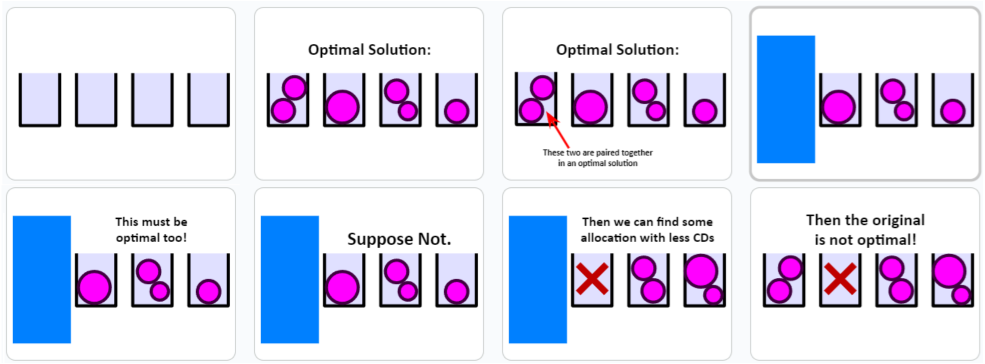
Optimal Solution:
Optimal Solution:
These two are paired together in an optimal solution
This must be optimal too!
Suppose Not.
Then we can find some allocation with less CDs
Then the original is not optimal!

Figure 1: Optimal substructure illustration

Answer 1c

This fails because the largest file (f_1) and smallest file (f_2) may exceed **100 MB** and cannot fit on a single CD.

Counterexample: $A = \{10, 20, 30, 95, 99\}$,

- **Optimal solution:** $\{10, 20\}, \{30\}, \{95\}, \{99\} \rightarrow$ **4 CDs** (cannot be improved).
- Using eq. 1 with $f_1 = 99$, $f_2 = 10$ gives:

$$1 + MinCD(\{20, 30, 95\}) = 1 + 2 = 3,$$

which is **impossible** since f_1 and f_2 do not fit on a single CD.

Question 2 [G]/[P2]

Assume any optimal solution contains a pair on a CD.
Select all **True** statements:

a. The smallest file f must be paired in some optimal solution.

b. The pair $\{f_1, f_2\}$, where f_1 is the **smallest** file and f_2 is the **largest file that fits with f_1 on one CD**, must appear in some optimal solution.

c. The pair $\{f_1, f_2\}$, where f_1 is the **smallest** file and f_2 is the **largest** file, must appear in some optimal solution.

Answer 2a

True. Some optimal solution includes the smallest file f (**if a pair exists**):

- If the smallest file f is already in a pair, we are done.
- Otherwise, swap f with any file a from an optimal pair $\{a, b\}$ (exchange argument).
  - Since $f \leq a$, if $a + b \leq 100$ MB, then $f + b \leq 100$ MB.
  - The total number of CDs used remains unchanged.

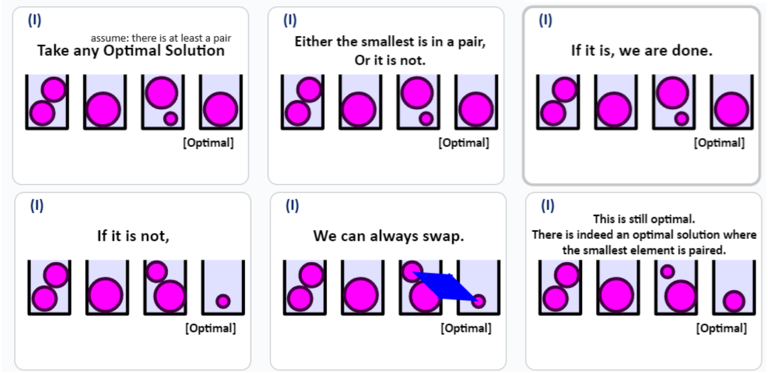
(I)
assume: there is at least a pair
Take any Optimal Solution
[Optimal]
(I)
Either the smallest is in a pair,
Or it is not.
[Optimal]
(I)
If it is, we are done.
[Optimal]
(I)
If it is not,
[Optimal]
(I)
We can always swap.
[Optimal]
(I)
This is still optimal.
There is indeed an optimal solution where
the smallest element is paired.
[Optimal]

Figure 2: Smallest file in a pair

Answer 2b

True. From (2a), smallest file f_1 that is paired with some file (**if a pair exists**):

- If f_1 (smallest) is paired with f_2 (largest that fits), we are done.
- Otherwise, if paired with f_3 (some other file), swap it with f_2 (exchange argument).
  1. If f_2 was unpaired, the solution remains optimal.
  2. If f_2 was paired with f_4 ($\{f_1, f_3\}$ and $\{f_2, f_4\}$ fit in 2 CDs), the swap is valid since $f_2 > f_3$ ($\{f_1, f_2\}$ and $\{f_3, f_4\}$ also fit in 2 CDs).

Thus, an optimal solution exists where f_2 is paired with the smallest file f_1.

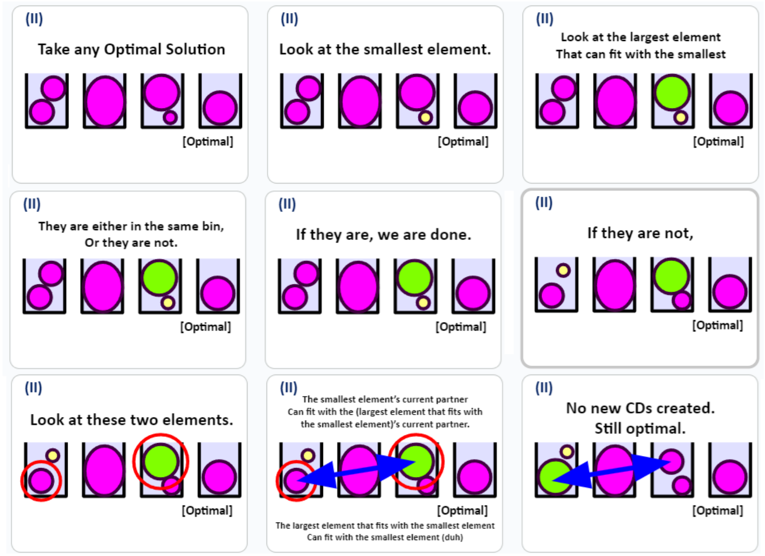
(II)
Take any Optimal Solution
[Optimal]
(II)
Look at the smallest element.
[Optimal]
(II)
Look at the largest element
That can fit with the smallest
[Optimal]
(II)
They are either in the same bin,
Or they are not.
[Optimal]
(II)
If they are, we are done.
[Optimal]
(II)
If they are not,
[Optimal]
(II)
Look at these two elements.
[Optimal]
(II)
The smallest element's current partner
Can fit with the (largest element that fits with
the smallest element)'s current partner.
[Optimal]
The largest element that fits with the smallest element
Can fit with the smallest element (duh)
(II)
No new CDs created.
Still optimal.
[Optimal]

Figure 3: Smallest and largest (that fits) fit in a pair

Answer 2c

False. Same argument as 1c. The problem is the “largest file”.

Question 3 [G]/[P2]

Derive the greedy algorithm for obtaining the minimum number of CDs that Bob needs to burn his music files and apply it to this array of file sizes $A = \{89, 74, 81, 12, 7, 91\}$, what is the output of $MinCD(A)$ for this test case?

Answer

1. **Sort the file sizes**
   - Use a sorting algorithm ($O(n \log n)$) or Counting Sort ($O(n)$).
2. **Initialize two pointers**
   - Let `forward` point to the **smallest file** (A_1),
   - and `backward` point to the **largest file** (A_n).
   - Set `num_cds = 0` to track the CDs used.
3. **Pair smallest and largest files greedily**
   - If $A_{\text{forward}} + A_{\text{backward}} \leq 100$ MB, store them together and move **both** pointers.
   - Otherwise, store A_{backward} alone and move only the **right** pointer.
4. **Repeat until pointers meet**
   - Continue advancing `forward` and retreating `backward` until all files are stored.
5. **Return total CDs used**

Remarks

Interested students can try implementing a similar greedy algorithm with these problems:

- UVa 410 - Station Balance

For $A = \{89, 74, 81, 12, 7, 91\}$, we have the following trace:

	Remaining Files $[A_{\text{forward}}, \cdots, A_{\text{backward}}]$	Pair	CDs Used
0	[7, 12, 74, 81, 89, 91]	$\{$91, 7$\}$	1
1	[12, 74, 81, 89]	$\{$89$\}$	2
2	[12, 74, 81]	$\{$81, 12$\}$	3
3	[74]	$\{$74$\}$	4

Note: Pairings are not required but can easily be tracked.

We illustrate using a larger set $A = \{89, 59, 32, 74, 81, 12, 7, 49, 43, 51, 62, 91, 27\}$:

	Remaining Files $[A_{\text{forward}}, \cdots, A_{\text{backward}}]$	Pair	CDs Used
0	[7, 12, 27, 32, 43, 49, 51, 59, 62, 74, 81, 89, 91]	{91, 7}	1
1	[12, 27, 32, 43, 49, 51, 59, 62, 74, 81, 89]	{89}	2
2	[12, 27, 32, 43, 49, 51, 59, 62, 74, 81]	{81, 12}	3
3	[27, 32, 43, 49, 51, 59, 62, 74]	{74}	4
4	[27, 32, 43, 49, 51, 59, 62]	{27, 62}	5
5	[32, 43, 49, 51, 59]	{32, 59}	6
6	[43, 49, 51]	{51, 43}	7
7	[49]	{49}	8

Activity Selection

Your task is to find the largest subset of mutually compatible activities:

- Given a set of activities $S = \{a_1, a_2, \dots, a_n\}$.
- Each activity occurs within $[s_i, f_i)$ (start time inclusive, finish time exclusive).
- Two activities a_i and a_j are **compatible** if their intervals do not overlap:
  - $s_i \geq f_j$ (i.e., a_i starts after a_j finishes), or
  - $s_j \geq f_i$ (i.e., a_j starts after a_i finishes).

Example

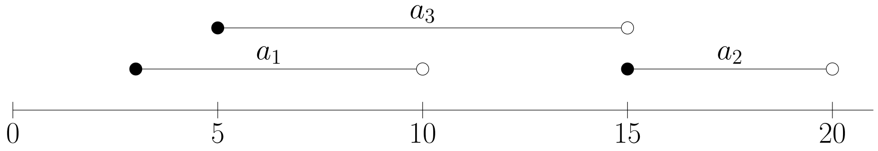

Figure 4: Activity intervals for $a_1 = [3, 10)$, $a_2 = [15, 20)$, and $a_3 = [5, 15)$.

- Compatible sets: $\{a_1, a_2\}$ or $\{a_2, a_3\}$ (both optimal).
- Incompatible set: $\{a_1, a_3\}$.

Question 4 [G]

Which of these greedy strategy (or strategies) work for the activity selection problem?

a. Choose the activity a that **starts last**, remove conflicting activities, and recurse.

b. Choose the activity a that **ends last**, remove conflicting activities, and recurse.

c. Choose the **shortest** activity a, remove conflicting activities, and recurse.

Answer 4a

Correct, see the details in the next question!

Answer 4a

Correct, see the details in the next question!

Answer 4b

Wrong, counterexample: $a_1 = [1, 10)$, $a_2 = [1, 5)$, $a_3 = [6, 9)$.

- Choosing a_1 (which ends last) blocks all other choices, leading to a suboptimal solution.
- **Optimal solution:** $\{a_2, a_3\}$.

Answer 4a

Correct, see the details in the next question!

Answer 4b

Wrong, counterexample: $a_1 = [1, 10)$, $a_2 = [1, 5)$, $a_3 = [6, 9)$.

- Choosing a_1 (which ends last) blocks all other choices, leading to a suboptimal solution.
- **Optimal solution:** $\{a_2, a_3\}$.

Answer 4c

Wrong. Counterexample: $a_1 = [1, 4)$, $a_2 = [3, 5)$, $a_3 = [4, 10)$.

- Choosing a_2 (the shortest activity) blocks all other choices, leading to a suboptimal solution.
- **Optimal solution:** $\{a_1, a_3\}$.

Question 5

Derive the greedy algorithm for the Activity Selection Problem.

Answer

Optimal Substructure

Suppose an optimal schedule S includes activity a_j:

- $Before_j = \{a_i \mid f_i \leq s_j\}$ (activities finishing before a_j starts).
- $After_j = \{a_i \mid s_i \geq f_j\}$ (activities starting after a_j ends).

Then, S must also contain an optimal schedule for $Before_j$ and an optimal schedule for $After_j$. This can be proven by contradiction.

Greedy Choice Property

Greedily selecting the activity a that **starts last** works. Any optimal solution can be modified (using an exchange argument) to include the activity that starts last:

- If an optimal schedule S originally contained some latest-starting activity a, it can always be **replaced** with a^* while maintaining compatibility and the same number of selected activities.
- This guarantees that selecting a^* is optimal.

Greedy Algorithm

1. **Sort the activities by start time** in non-decreasing order (if not already sorted).
   - Sorting takes $\Theta(n \log n)$, but once sorted, the algorithm runs in $O(n)$.
2. **Initialize an empty schedule** S.
3. **Iterate through the activities in reverse order** (starting from the last activity):
   - Select the latest-starting activity a^* that is compatible with the current schedule S.
   - Add a^* to S.
4. **Return the final schedule** S as the largest set of compatible activities.

Remarks

Interested students can try implementing a similar greedy algorithm with these problems:

- Kattis - Classrooms
- LeetCode - Non-Overlapping Intervals

Practical [Optional]

LeetCode - Non-Overlapping Intervals is highly similar to our Activity Selection problem.

1. Solve this LeetCode problem!
2. See others Greedy - LeetCode