CS 577 - Greedy

Manolis Vlatakis

Department of Computer Sciences
University of Wisconsin – Madison

Fall 2024

Intro

The algorithms class is interesting, but even when I find the solution,
I don't know how to prove it's correct.

The algorithms class is interesting, but even when I find the solution,
I don't know how to prove it's correct.

Let's fix that!
Welcome to
Greedy Algorithms!

Greedy Algorithms

A Personal Confession

Student Perspective

As a student, I knew what the correct solution was, but I couldn't prove it.

Greedy Algorithms

A Personal Confession

Professor Perspective

As a professor, I don't even know what the correct solution is anymore.

Warm-up: The Solution is Simple… But the Proof is Stronger

"Greed is Good" - Michael Douglas in Wall Street

- A greedy algorithm always makes the choice that looks best at the moment
- Greedy algorithms do not always lead to optimal solutions, but for many problems they do

Warm-up: Maximizing a Linear Function

Definition

Maximize the value of $f(x) = c \cdot x$, where $x \in S$ such as $S = \{-1, 2, 5, 8, -3, 7\}$ or any unordered set S of distinct integers.

🤔What is the algorithm that you will choose to solve this problem?

Warm-up: Maximizing a Linear Function

Definition

Maximize the value of $f(x) = c \cdot x$, where $x \in S$ such as $S = \{-1, 2, 5, 8, -3, 7\}$ or any unordered set S of distinct integers.

🤔What is the algorithm that you will choose to solve this problem?

Solution To maximize $f(x) = c \cdot x$, we clearly want to pick the largest value of x from the set S if $c \geq 0$, otherwise the smallest.

Select $x = \max(S)$ because it gives the largest value for $c \cdot x$.

Why is the solution correct?

Proof

- Suppose the optimal solution, denoted as $\mathcal{O}(f, S)$, is different from the solution provided by our algorithm, $\mathcal{A}(f, S)$.

$$\mathcal{O}(f, S) \neq \mathcal{A}(f, S)$$

Why is the solution correct?

Proof

- Suppose the optimal solution, denoted as $\mathcal{O}(f, S)$, is different from the solution provided by our algorithm, $\mathcal{A}(f, S)$.

$$\mathcal{O}(f, S) \neq \mathcal{A}(f, S)$$

- We aim to show that $\mathcal{A}(f, S) \leftarrow \mathcal{O}(f, S)$ is only improving!!!

Why is the solution correct?

Proof

- Suppose the optimal solution, denoted as $\mathcal{O}(f, S)$, is different from the solution provided by our algorithm, $\mathcal{A}(f, S)$.

$$\mathcal{O}(f, S) \neq \mathcal{A}(f, S)$$

- We aim to show that $\mathcal{A}(f, S) \leftarrow \mathcal{O}(f, S)$ is only improving!!!

🤔Why is it obvious that $\mathcal{O}(f, S) \geq \mathcal{A}(f, S)$ for any algorithm $\mathcal{A}$?

- $\mathcal{O}(f, S)$ represents the optimal solution.
- $\mathcal{A}(f, S)$ is the solution generated by our algorithm.

Why is the solution correct?

Proof

- Suppose the optimal solution, denoted as $\mathcal{O}(f,S)$, is different from the solution provided by our algorithm, $\mathcal{A}(f,S)$.

$$\mathcal{O}(f,S) \neq \mathcal{A}(f,S)$$

- We aim to show that $\mathcal{A}(f,S) \leftarrow \mathcal{O}(f,S)$ is only improving!!!

🤔Why is it obvious that $\mathcal{O}(f,S) \geq \mathcal{A}(f,S)$ for any algorithm $\mathcal{A}$?

- $\mathcal{O}(f,S)$ represents the optimal solution.
- $\mathcal{A}(f,S)$ is the solution generated by our algorithm.

By definition, $\mathcal{O}(f,S)$ provides the best possible solution, so no algorithm's solution can exceed it.

Why is the solution correct?

Proof

- We aim to show that $\mathcal{A}(f, S) \leftarrow \mathcal{O}(f, S)$ is only improving!!!

Why is the solution correct?

Proof

- We aim to show that $\mathcal{A}(f,S) \leftarrow \mathcal{O}(f,S)$ is only improving!!!

We will do the proof of arrogance:

We are better than any other best ☺ !!!

Why is the solution correct?

Proof

- We aim to show that $\mathcal{A}(f, S) \leftarrow \mathcal{O}(f, S)$ is only improving!!!

We will do the proof of arrogance:

We are better than any other best ☺ !!!

Proof by Contradiction

- Now, let's assume that the optimal solution chooses a value $x' \in S$, where $x' \neq x^* = \max(S)$.

Why is the solution correct?

Proof

- We aim to show that $\mathcal{A}(f, S) \leftarrow \mathcal{O}(f, S)$ is only improving!!!

We will do the proof of arrogance:

We are better than any other best ☺ !!!

Proof by Contradiction

- Now, let's assume that the optimal solution chooses a value $x' \in S$, where $x' \neq x^* = \max(S)$.
- Clearly, $x' < \max(S)$. Therefore, the value of $c \cdot x'$ would be less than $c \cdot \max(S)$.

Why is the solution correct?

Proof

- We aim to show that $\mathcal{A}(f, S) \leftarrow \mathcal{O}(f, S)$ is only improving!!!

We will do the proof of arrogance:

We are better than any other best ☺ !!!

Proof by Contradiction

- Now, let's assume that the optimal solution chooses a value $x' \in S$, where $x' \neq x^* = \max(S)$.
- Clearly, $x' < \max(S)$. Therefore, the value of $c \cdot x'$ would be less than $c \cdot \max(S)$.
- **Thus, we can transform the optimal solution to match our algorithm's choice of $x^* = \max(S)$ without loss.**

Why is the solution correct?

Proof

- We aim to show that $\mathcal{A}(f, S) \leftarrow \mathcal{O}(f, S)$ is only improving!!!

We will do the proof of arrogance:

We are better than any other best ☺ !!!

Proof by Contradiction

- Now, let's assume that the optimal solution chooses a value $x' \in S$, where $x' \neq x^* = \max(S)$.
- Clearly, $x' < \max(S)$. Therefore, the value of $c \cdot x'$ would be less than $c \cdot \max(S)$.
- **Thus, we can transform the optimal solution to match our algorithm's choice of $x^* = \max(S)$ without loss.**
- By choosing $x^* = \max(S)$, the value $c \cdot x^*$ provides only a better result than $c \cdot x'$, which completes the proof.

Maximizing Your Profit in the Magic World: A Harry Potter Story

Imagine you are Harry Potter, and you run a magical shop in Diagon Alley. You have a variety of magical items for sale, and just like in the muggle world, some items make you money, but some have a negative price (they cost you money to keep!).

- **Item A**: Earns you 12 Galleons for each one sold.
- **Item B**: Earns you 3 Galleons for each one sold.

Your goal is to choose two different items to maximize your profit!!!

🤔Too Easy !!! Any constraints???

Maximizing Your Profit in the Magic World: A Harry Potter Story

- However, you can only select one item for the 12 Galleon promotion and one item for the 3 Galleon promotion.
- Here's a list of magical items you have in stock, with their quantities: S={-1, 2, 5, 8, -3, and 7}

☺: *In the magical world, some items are cursed, which is why they have a negative price—every time you sell them, you actually lose money!*

🤔What is the algorithm that you will choose to solve this problem?

Maximizing a Linear Function with Two Variables

Let's simplify using math:

Definition

Maximize the value of $f(x, y) = 12 \cdot x + 3 \cdot y$, where $x, y \in S$, $x \neq y$, and $S = \{-1, 2, 5, 8, -3, 7\}$.

🤔What is the algorithm that you will choose to solve this problem?

Maximizing a Linear Function with Two Variables

To maximize $f(x,y) = 12 \cdot x + 3 \cdot y$, we need to choose the two largest values in the set S, assigning the larger coefficient to the larger value.

$$\text{Select } x^* = \max(S) \text{ and } y^* = \max(S \setminus \{x^*\}),$$

Why is the two-variable solution correct?

The Proof of Arrogance Again

We're about to prove that our solution is better than any other —because why settle for anything less? — ☺

I know, I know... it seems ridiculous to have to prove this, but let's just imagine there's a non-believer out there who doesn't think it's obvious!!

Why is the two-variable solution correct?

The Proof of Arrogance Again

What would a non-believer try to claim?

$$\max_{x,y \in \{-1,2,5,8,-3,7\}} 12 \cdot x + 3 \cdot y$$

Proof by Contradiction

- Suppose the optimal solution chosen by a non-believer uses values x' and y', where

$$x', y' \in S \quad \text{and} \quad x' \neq x^* = \max(S) \quad \text{or} \quad y' \neq y^* = \max(S \setminus \{x'\}).$$

Why is the two-variable solution correct?

The Proof of Arrogance Again

What would a non-believer try to claim?

$$\max_{x,y \in \{-1,2,5,8,-3,7\}} 12 \cdot x + 3 \cdot y$$

Proof by Contradiction

- Suppose the optimal solution chosen by a non-believer uses values x' and y', where

$$x', y' \in S \quad \text{and} \quad x' \neq x^* = \max(S) \quad \text{or} \quad y' \neq y^* = \max(S \setminus \{x'\}).$$

- Clearly, $x' < x^* = \max(S)$ or $y' < y^* = \max(S \setminus \{x'\})$.

Why is the two-variable solution correct?

The Proof of Arrogance Again

Proof b

- Su ... ver uses
  val

  x ... $x(S \setminus \{x'\})$.

- Cle

🤔Coul

Why is the two-variable solution correct?

The Proof of Arrogance Again

What would a non-believer try to claim?

$$\max_{x,y \in \{-1,2,5,8,-3,7\}} 12 \cdot x + 3 \cdot y$$

Proof by Contradiction

- Clearly, $x' < x^* = \max(S)$ or $y' < y^* = \max(S \setminus \{x'\})$.

Could the non-believer's solution $x' < y'$ be optimal?

No! We can improve the non-believer's solution by simply swapping $x' \leftrightarrow y'$.

$$\text{Example}: f(5,12) = 12 \times 5 + 3 \times 7 \leq 12 \times 7 + 3 \times 5 = f(12,5).$$

WHY IS THE TWO-VARIABLE SOLUTION CORRECT?

THE PROOF OF ARROGANCE AGAIN

What would a non-believer try to claim?

$$\max_{x,y \in \{-1,2,5,8,-3,7\}} 12 \cdot x + 3 \cdot y$$

Proof by Contradiction

- Clearly, $x' < x^* = \max(S)$ or $y' < y^* = \max(S \setminus \{x'\})$.

If $x' = \max(S)$ and $y' < y^*$, it is trivial— Why?? .

Why is the two-variable solution correct?

The Proof of Arrogance Again

What would a non-believer try to claim?

$$\max_{x,y \in \{-1,2,5,8,-3,7\}} 12 \cdot x + 3 \cdot y$$

Proof by Contradiction

- Clearly, $x' < x^* = \max(S)$ or $y' < y^* = \max(S \setminus \{x'\})$.

If $x' = \max(S)$ and $y' < y^*$, it is trivial— just exchange $y' \leftrightarrow y^*$.

Why is the two-variable solution correct?

The Proof of Arrogance Again

What would a non-believer try to claim?

$$\max_{x,y \in \{-1,2,5,8,-3,7\}} 12 \cdot x + 3 \cdot y$$

Proof by Contradiction

- Clearly, $x' < x^* = \max(S)$ or $y' < y^* = \max(S \setminus \{x'\})$.

🤔Could $x' > x^*$ be the case?

Why is the two-variable solution correct?

The Proof of Arrogance Again

What would a non-believer try to claim?

$$\max_{x,y \in \{-1,2,5,8,-3,7\}} 12 \cdot x + 3 \cdot y$$

Proof by Contradiction

- Clearly, $x' < x^* = \max(S)$ or $y' < y^* = \max(S \setminus \{x'\})$.

Could $x' > x^*$ be the case?

No, because we chose x^* to be the maximum of the distinct integers in S.

Why is the two-variable solution correct?

The Proof of Arrogance Again

What would a non-believer try to claim?

$$\max_{x,y \in \{-1,2,5,8,-3,7\}} 12 \cdot x + 3 \cdot y$$

Proof by Contradiction

- Clearly, $x' < x^* = \max(S)$ or $y' < y^* = \max(S \setminus \{x'\})$.

No, because we chose x^* to be the maximum of the distinct integers in S.

So the only remaining case is: $x^* > x'$

WHY IS THE TWO-VARIABLE SOLUTION CORRECT?

THE PROOF OF ARROGANCE AGAIN

What would a non-believer try to claim?

$$\max_{x,y \in \{-1,2,5,8,-3,7\}} 12 \cdot x + 3 \cdot y$$

Proof by Contradiction

- Clearly, $x' < x^* = \max(S)$ or $y' < y^* = \max(S \setminus \{x'\})$.

🤔Could $x^* > x' > y' > y^*$ be the case?

Why is the two-variable solution correct?

The Proof of Arrogance Again

What would a non-believer try to claim?

$$\max_{x,y \in \{-1,2,5,8,-3,7\}} 12 \cdot x + 3 \cdot y$$

Proof by Contradiction

- Clearly, $x' < x^* = \max(S)$ or $y' < y^* = \max(S \setminus \{x'\})$.

🤔Could $x^* > x' > y' > y^*$ be the case?

No, because <u>we</u> chose y^* to be the maximum after x^*.

Why is the two-variable solution correct?

The Proof of Arrogance Again

What would a non-believer try to claim?

$$\max_{x,y\in\{-1,2,5,8,-3,7\}} 12\cdot x+3\cdot y$$

Proof by Contradiction

- Clearly, $x' < x^* = \max(S)$ or $y' < y^* = \max(S \setminus \{x'\})$.

The only possible case is $x^* > y^* > x' > y'$....What time is now?

Why is T?

The Proof of

Proof b

- Cle

The only ... ow?

Why is the two-variable solution correct?

The Proof of Arrogance Again

What would a non-believer try to claim?

$$\max_{x,y \in \{-1,2,5,8,-3,7\}} 12 \cdot x + 3 \cdot y$$

Proof by Contradiction

- Clearly, $x' < x^* = \max(S)$ or $y' < y^* = \max(S \setminus \{x'\})$.

The only possible case is $x^* > y^* > x' > y'$....What time is now?

Let's exchange $x' \leftrightarrow x^*$ and $y' \leftrightarrow y^*$. Now, the non-believer has our solution and seems happier ☺.

Moment of Truth

Aesop's Moral

In both problems, finding the solution was straightforward!

Moment of Truth

Aesop's Moral

In both problems, finding the solution was straightforward!

Aesop's Moral

However, proving that the solution is optimal required more effort!

However !!!

The Importance of Proof

1. Proof as Validation

The proof is the only way to be certain
about the heuristic we have chosen.
After our proof, no one can question our solution.

The Importance of Proof

1. Proof as Validation

The proof is the only way to be certain
about the heuristic we have chosen.
After our proof, no one can question our solution.

2. The Structure of Proofs

The proof may seem strange at first,
but it always follows a specific pattern.

Greedy

Greedy Algorithms

What is a Greedy Algorithm (greedy)?

- Typically, thought of as a heuristic that is locally optimal.

Were our algorithms locally optimal??

Greedy Algorithms

What is a Greedy Algorithm (greedy)?

- Typically, thought of as a heuristic that is locally optimal.
- Is greedy always the best?

Greedy Algorithms

What is a Greedy Algorithm (greedy)?

- Typically, thought of as a heuristic that is locally optimal.
- Is greedy always the best? No, but a good place to start!!!

Greedy Algorithms

What is a Greedy Algorithm (greedy)?

- Typically, thought of as a heuristic that is locally optimal.
- Is greedy always the best? No, but a good place to start!!!
- This notion has yet to be fully formalized, and it often problem specific.

GREEDY ALGORITHMS

What is a Greedy Algorithm (GREEDY)?

- Typically, thought of as a heuristic that is locally optimal.
- Is GREEDY always the best? No, but a good place to start!!!
- This notion has yet to be fully formalized, and it often problem specific.

Definition from Priority Algorithms

A greedy algorithm is an algorithm that processes the input in a specified order. For each request in the input, the greedy algorithm processes it so as to minimize (resp. maximize) the objective, assuming that the request is the last request.

Greedy Algorithms

What is a Greedy Algorithm (greedy)?

- Typically, thought of as a heuristic that is locally optimal.
- Is greedy always the best? No, but a good place to start!!!
- This notion has yet to be fully formalized, and it often problem specific.

Definition from Priority Algorithms

A greedy algorithm is an algorithm that processes the input in a specified order. For each request in the input, the greedy algorithm processes it so as to minimize (resp. maximize) the objective, assuming that the request is the last request.

For a given problem, there may be many greedy algorithms.

Historical Break

Who Named Them?

History of the Term

The exact origin of the term "greedy algorithm" is not attributed to one person. However:

- It became popular in the 1950s and 1960s during the study of algorithm theory.
- The term likely evolved from the behavior of these algorithms—they make decisions that seem "greedy" by choosing the best local option at every step.
- Notable contributors: **Edsgar Dijkstra**, **Richard Karp**, and others, helped formalize greedy approaches.

Is greedy Optimal?

Not always: Packing Items into Boxes

- You have boxes of a fixed size (e.g., each box can hold 10kg).

Is greedy Optimal?

Not always: Packing Items into Boxes

- You have boxes of a fixed size (e.g., each box can hold 10kg).
- You have several items of varying weights (e.g., 2.5 kg to 9 kg).

Is greedy Optimal?

Not always: Packing Items into Boxes

- You have boxes of a fixed size (e.g., each box can hold 10kg).
- You have several items of varying weights (e.g., 2.5 kg to 9 kg).
- Objective: Fit all items into the minimum number of boxes.

IS GREEDY OPTIMAL?

Not always: Packing Items into Boxes

- You have boxes of a fixed size (e.g., each box can hold 10kg).
- You have several items of varying weights (e.g., 2.5 kg to 9 kg).
- Objective: Fit all items into the minimum number of boxes.
- Greedy heuristic: FIRST FIT INCREASING (FFI)—place each item in the first box that has enough space.

Is greedy Optimal?

Not always: Packing Items into Boxes

- You have boxes of a fixed size (e.g., each box can hold 10kg).
- You have several items of varying weights (e.g., 2.5 kg to 9 kg).
- Objective: Fit all items into the minimum number of boxes.
- Greedy heuristic: First Fit Increasing (ffi)—place each item in the first box that has enough space.

Non-optimal example:

Is greedy Optimal?

Not always: Packing Items into Boxes

- You have boxes of a fixed size (e.g., each box can hold 10kg).
- You have several items of varying weights (e.g., 2.5 kg to 9 kg).
- Objective: Fit all items into the minimum number of boxes.
- Greedy heuristic: First Fit Increasing (ffi)—place each item in the first box that has enough space.

Non-optimal example:

- Items: $\sigma = \langle 4.9, 4.9, 5.1, 5.1 \rangle$

Is greedy Optimal?

Not always: Packing Items into Boxes

- You have boxes of a fixed size (e.g., each box can hold 10kg).
- You have several items of varying weights (e.g., 2.5 kg to 9 kg).
- Objective: Fit all items into the minimum number of boxes.
- Greedy heuristic: First Fit Increasing (ffi)—place each item in the first box that has enough space.

Non-optimal example:

- Items: $\sigma = \langle 4.9, 4.9, 5.1, 5.1 \rangle$
- ffi:

Is greedy Optimal?

Not always: Packing Items into Boxes

- You have boxes of a fixed size (e.g., each box can hold 10kg).
- You have several items of varying weights (e.g., 2.5 kg to 9 kg).
- Objective: Fit all items into the minimum number of boxes.
- Greedy heuristic: First Fit Increasing (ffi)—place each item in the first box that has enough space.

Non-optimal example:

- Items: $\sigma = \langle 4.9, 4.9, 5.1, 5.1 \rangle$
- ffi: requires 3 boxes.

Is greedy Optimal?

Not always: Packing Items into Boxes

- You have boxes of a fixed size (e.g., each box can hold 10kg).
- You have several items of varying weights (e.g., 2.5 kg to 9 kg).
- Objective: Fit all items into the minimum number of boxes.
- Greedy heuristic: First Fit Increasing (ffi)—place each item in the first box that has enough space.

Non-optimal example:

- Items: $\sigma = \langle 4.9, 4.9, 5.1, 5.1 \rangle$
- ffi: requires 3 boxes.
- Optimal packing:

Is greedy Optimal?

Not always: Packing Items into Boxes

- You have boxes of a fixed size (e.g., each box can hold 10kg).
- You have several items of varying weights (e.g., $2.5\,kg$ to $9\,kg$).
- Objective: Fit all items into the minimum number of boxes.
- Greedy heuristic: First Fit Increasing (ffi)—place each item in the first box that has enough space.

Non-optimal example:

- Items: $\sigma = \langle 4.9, 4.9, 5.1, 5.1 \rangle$
- ffi: requires 3 boxes.
- Optimal packing: only needs 2 boxes.

Is greedy Optimal?

Not always: Bin Packing Problem

- Bins of size 1, and requests of size $(0, 1]$.

Is greedy Optimal?

Not always: Bin Packing Problem

- Bins of size 1, and requests of size $(0, 1]$.
- Objective: Pack the items in the minimum number of bins.

Is greedy Optimal?

Not always: Bin Packing Problem

- Bins of size 1, and requests of size $(0, 1]$.
- Objective: Pack the items in the minimum number of bins.
- Greedy heuristic: First Fit Increasing (ffi)

Is greedy Optimal?

Not always: Bin Packing Problem

- Bins of size 1, and requests of size $(0, 1]$.
- Objective: Pack the items in the minimum number of bins.
- Greedy heuristic: First Fit Increasing (ffi)

Non-optimal example:

Is greedy Optimal?

Not always: Bin Packing Problem

- Bins of size 1, and requests of size $(0, 1]$.
- Objective: Pack the items in the minimum number of bins.
- Greedy heuristic: First Fit Increasing (ffi)

Non-optimal example:

- $\sigma = \langle 1/2 - \varepsilon, 1/2 - \varepsilon, 1/2 + \varepsilon, 1/2 + \varepsilon \rangle$

IS GREEDY OPTIMAL?

Not always: Bin Packing Problem

- Bins of size 1, and requests of size $(0, 1]$.
- Objective: Pack the items in the minimum number of bins.
- Greedy heuristic: FIRST FIT INCREASING (FFI)

Non-optimal example:

- $\sigma = \langle 1/2 - \varepsilon, 1/2 - \varepsilon, 1/2 + \varepsilon, 1/2 + \varepsilon \rangle$
- FFI:

Is greedy Optimal?

Not always: Bin Packing Problem

- Bins of size 1, and requests of size $(0, 1]$.
- Objective: Pack the items in the minimum number of bins.
- Greedy heuristic: First Fit Increasing (ffi)

Non-optimal example:

- $\sigma = \langle 1/2 - \varepsilon, 1/2 - \varepsilon, 1/2 + \varepsilon, 1/2 + \varepsilon \rangle$
- ffi: 3 bins

Is greedy Optimal?

Not always: Bin Packing Problem

- Bins of size 1, and requests of size $(0, 1]$.
- Objective: Pack the items in the minimum number of bins.
- Greedy heuristic: First Fit Increasing (ffi)

Non-optimal example:

- $\sigma = \langle 1/2 - \varepsilon, 1/2 - \varepsilon, 1/2 + \varepsilon, 1/2 + \varepsilon \rangle$
- ffi: 3 bins
- opt:

Is greedy Optimal?

Not always: Bin Packing Problem

- Bins of size 1, and requests of size $(0, 1]$.
- Objective: Pack the items in the minimum number of bins.
- Greedy heuristic: First Fit Increasing (ffi)

Non-optimal example:

- $\sigma = \langle 1/2 - \varepsilon, 1/2 - \varepsilon, 1/2 + \varepsilon, 1/2 + \varepsilon \rangle$
- ffi: 3 bins
- opt: 2 bins

Is greedy Optimal?

Not always: Bin Packing Problem

- Bins of size 1, and requests of size $(0, 1]$.
- Objective: Pack the items in the minimum number of bins.
- Greedy heuristic: First Fit Increasing (FFI)

Non-optimal example:

- $\sigma = \langle 1/2 - \varepsilon, 1/2 - \varepsilon, 1/2 + \varepsilon, 1/2 + \varepsilon \rangle$
- FFI: 3 bins
- OPT: 2 bins

Techniques for showing that GREEDY is optimal:

- Always stays ahead
- Exchange argument

Coin Collection

The Cashier's Problem

- Imagine a cashier has to count various coins to provide change.

Cashier counting coins

The Cashier's Problem

- Imagine a cashier has to count various coins to provide change.
- How can the cashier efficiently count the minimum number of coins to give exact change?

Cashier counting coins

Apple Pay

- Now, with solutions like Apple Pay, we avoid counting coins!

Coin Collection

Definition

We are given an array of coin denominations used in the U.S. $(1¢, 5¢, 10¢, 25¢, 50¢, \$1)$. The goal is to determine the minimum number of coins needed to sum up to a given amount S.

- The input consists of the set of coins:

$$\{1¢, 5¢, 10¢, 25¢, 50¢, \$1\}$$

- The output is the minimum number of coins that sum up to S

What happens for $S = 87¢$?

- Can we solve the problem generally using a greedy approach?

The Greedy Idea

Greedy Strategy

The idea is to always select the largest coin denomination that does not exceed the remaining amount.

The Greedy Idea

Greedy Strategy

The idea is to always select the largest coin denomination that does not exceed the remaining amount.

Example: Solving for $S = 87¢$

- Step 1: Pick the largest coin 50¢ . Remaining = 87¢ − 50¢ = 37¢
- Step 2: Pick 25¢ . Remaining = 37¢ − 25¢ = 12¢
- Step 3: Pick 10¢ . Remaining = 12¢ − 10¢ = 2¢
- Step 4: Pick 1¢ . Remaining = 2¢ − 1¢ = 1¢
- Step 5: Pick 1¢ . Remaining = 1¢ − 1¢ = 0

The Greedy Idea

Greedy Strategy

The idea is to always select the largest coin denomination that does not exceed the remaining amount.

Example: Solving for $S = 87¢$

- Step 1: Pick the largest coin $50¢$. Remaining = $87¢ - 50¢ = 37¢$
- Step 2: Pick $25¢$. Remaining = $37¢ - 25¢ = 12¢$
- Step 3: Pick $10¢$. Remaining = $12¢ - 10¢ = 2¢$
- Step 4: Pick $1¢$. Remaining = $2¢ - 1¢ = 1¢$
- Step 5: Pick $1¢$. Remaining = $1¢ - 1¢ = 0$

The minimum number of coins for $S = 87¢$ is 5 coins:

$(50¢, 25¢, 10¢, 1¢, 1¢)$

Another Example with Greedy Approach

Greedy Strategy

The idea is to always select the largest coin denomination that does not exceed the remaining amount.

Another Example with Greedy Approach

Greedy Strategy

The idea is to always select the largest coin denomination that does not exceed the remaining amount.

Example: Solving for $S = 2.51\$$

- Step 1: Pick 1\$. Remaining = 2.51\$ – 1\$ = 1.51\$
- Step 2: Pick 1\$. Remaining = 1.51\$ – 1\$ = 0.51\$
- Step 3: Pick 25¢ . Remaining = 51¢ – 25¢ = 26¢
- Step 4: Pick 25¢ . Remaining = 26¢ – 25¢ = 1¢
- Step 5: Pick 1¢ . Remaining = 1¢ – 1¢ = 0

Another Example with Greedy Approach

Greedy Strategy

The idea is to always select the largest coin denomination that does not exceed the remaining amount.

Example: Solving for $S = 2.51\$$

- Step 1: Pick 1\$. Remaining = 2.51\$ – 1\$ = 1.51\$
- Step 2: Pick 1\$. Remaining = 1.51\$ – 1\$ = 0.51\$
- Step 3: Pick 25¢ . Remaining = 51¢ – 25¢ = 26¢
- Step 4: Pick 25¢ . Remaining = 26¢ – 25¢ = 1¢
- Step 5: Pick 1¢ . Remaining = 1¢ – 1¢ = 0

The minimum number of coins for $S = 2.51\$$ is 5 coins:

(1\$, 1\$, 25¢ , 25¢ , 1¢)

Proof of Optimality

Key Claim

The greedy algorithm provides the optimal solution because any deviation from the greedy choice leads to using more coins.

Proof

- Assume an optimal solution is different... What time is it now???

Proof of Optimality

Key Clai

The greed ecause
any devia ore coins.

Proof

- Assu me is it
  now?

Proof of Optimality

Key Claim

The greedy algorithm provides the optimal solution because any deviation from the greedy choice leads to using more coins.

Proof

- Assume an optimal solution uses a smaller denomination instead of the largest possible coin, e.g., replacing a 25¢ coin with five 5¢ coins.
- Replacing four 5¢ coins with a single 25¢ coin reduces the total number of coins, contradicting the assumption that the solution was optimal.

Interesting properties

Theorem

The greedy algorithm finds the minimum number of coins for any amount S using U.S. coin denominations.

Proof Structure

1. **Greedy Choice Property**: At each step, the largest coin denomination is selected.
2. **Optimal Substructure**: Once the largest coin is selected, the remaining amount can be solved optimally using the same strategy.
3. **Exchange Argument**: Replacing any coin in the greedy solution with smaller denominations results in more coins.

Federal Bank is Clever

What if...

What if the coins are: 1¢ , 5¢ , 12¢ , 20¢ ?

Federal Bank is Clever

What if...

What if the coins are: 1¢ , 5¢ , 12¢ , 20¢ ?

Greedy Solution: 5 coins

- Greedy chooses: 20¢ $\times$ 1 + 1¢ $\times$ 4

Federal Bank is Clever

What if...

What if the coins are: 1¢ , 5¢ , 12¢ , 20¢ ?

Greedy Solution: 5 coins

- Greedy chooses: 20¢ $\times$ 1 + 1¢ $\times$ 4

Optimal Solution: 2 coins

- Optimal chooses: 12¢ $\times$ 2

Federal Bank is Clever

What if...

What if the coins are: 1¢ , 5¢ , 12¢ , 20¢ ?

Greedy Solution: 5 coins

- Greedy chooses: 20¢ $\times$ 1 + 1¢ $\times$ 4

Optimal Solution: 2 coins

- Optimal chooses: 12¢ $\times$ 2

-How can we solve the general case???

Federal Bank is Clever

What if...

What if the coins are: 1¢ , 5¢ , 12¢ , 20¢ ?

Greedy Solution: 5 coins

- Greedy chooses: 20¢ × 1 + 1¢ × 4

Optimal Solution: 2 coins

- Optimal chooses: 12¢ × 2

-How can we solve the general case???
-Dynamic Programming Next Week!!!☺

How should I know that my idea is the correct greedy algorithm???

Seeking about two magic properties

1. **Greedy Choice Property**:
2. **Optimal Substructure**:

Seeking about two magic properties

1. **Greedy Choice Property**: At each step, you gave a very simple priority (the biggest/the smallest/the largest/the fastest....).
2. **Optimal Substructure**: Once the greedy choice has been done once, the remaining amount should be solved optimally using the same strategy.

Summary of Exchange Argument Method
My solution is better than yours ☺

Step 1: Label Your Solutions

Algorithm's Solution and Optimal Solution

Let's define two solutions:

- $A = \{a_1, a_2, \ldots, a_k\}$ is the solution generated by your algorithm.
- $O = \{o_1, o_2, \ldots, o_k\}$ is an optimal solution.

These will help us compare how close our algorithm's solution is to the best possible one.

Step 2: Compare Greedy with Optimal

Two Possibilities

Assume that your optimal solution is different from the solution given by the greedy algorithm. Then:

- There is an element in O that is not in A, and an element in A that is not in O, or
- Two consecutive elements in O are in a different order compared to how they appear in A (i.e., an inversion).

Step 3: Exchange

Making Greedy Optimal

Swap or exchange elements in O to make it more like A:

- Swap one element out and another in (in the first case).
- Swap the order of two elements (in the second case).

Each time we swap, the solution is no worse than before. Keep swapping until O and A are the same.

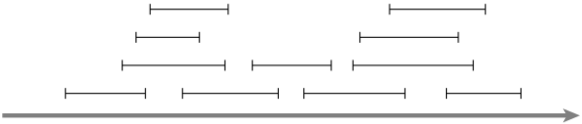
Conclusion: Greedy is Optimal

The Final Argument

After all the exchanges, we see that the greedy solution is just as good as any optimal solution, meaning the greedy algorithm is optimal.

Stays Ahead: Interval Scheduling

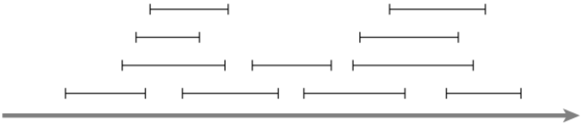
Interval Scheduling



Problem Definition

- Requests: $\sigma = \{r_1, \cdots, r_n\}$

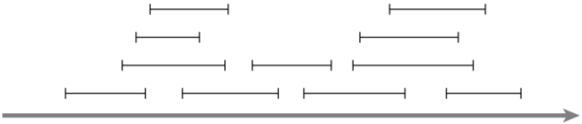
Interval Scheduling



Problem Definition

- Requests: $\sigma = \{r_1, \cdots, r_n\}$
- A request $r_i = (s_i, f_i)$, where s_i is the start time and f_i is the finish time.

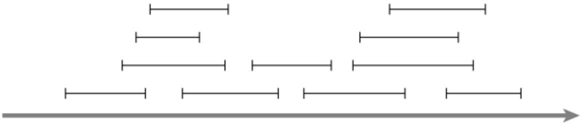
Interval Scheduling



Problem Definition

- Requests: $\sigma = \{r_1, \cdots, r_n\}$
- A request $r_i = (s_i, f_i)$, where s_i is the start time and f_i is the finish time.
- Objective: Produce a compatible schedule S that has maximum cardinality.

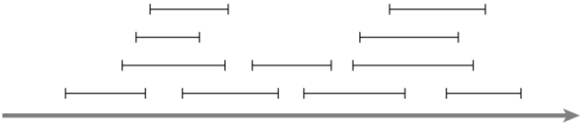
Interval Scheduling



Problem Definition

- Requests: $\sigma = \{r_1, \cdots, r_n\}$
- A request $r_i = (s_i, f_i)$, where s_i is the start time and f_i is the finish time.
- Objective: Produce a compatible schedule S that has maximum cardinality.
- Compatible schedule S: $\forall r_i, r_j \in S, f_i \leq s_j \vee f_j \leq s_i$.

Interval Scheduling



Problem Definition

- Requests: $\sigma = \{r_1, \cdots, r_n\}$
- A request $r_i = (s_i, f_i)$, where s_i is the start time and f_i is the finish time.
- Objective: Produce a compatible schedule S that has maximum cardinality.
- Compatible schedule S: $\forall r_i, r_j \in S, f_i \leq s_j \vee f_j \leq s_i$.

🤔What greedy heuristic might work?

Greedy Algorithms for Interval Scheduling

Heuristic 1: Earliest First

Schedule a compatible request with the earliest start time.

Greedy Algorithms for Interval Scheduling

Heuristic 1: Earliest First

Schedule a compatible request with the earliest start time.

Optimal?

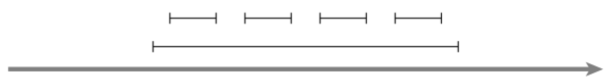
Greedy Algorithms for Interval Scheduling

Heuristic 1: Earliest First

Schedule a compatible request with the earliest start time.

Optimal?

Counter-example:



Greedy Algorithms for Interval Scheduling

Heuristic 2: Smallest Interval

Schedule a compatible request r_i with the smallest interval $(f_i - s_i)$.

Greedy Algorithms for Interval Scheduling

Heuristic 2: Smallest Interval

Schedule a compatible request r_i with the smallest interval $(f_i - s_i)$.

Optimal?

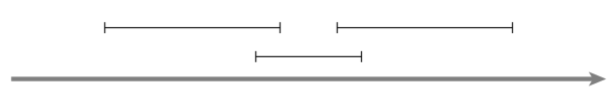
Greedy Algorithms for Interval Scheduling

Heuristic 2: Smallest Interval

Schedule a compatible request r_i with the smallest interval $(f_i - s_i)$.

Optimal?

Counter-example:



Greedy Algorithms for Interval Scheduling

Heuristic 3: Fewest Conflicts

Schedule a compatible request with the fewest remaining conflicts.

Greedy Algorithms for Interval Scheduling

Heuristic 3: Fewest Conflicts

Schedule a compatible request with the fewest remaining conflicts.

Optimal?

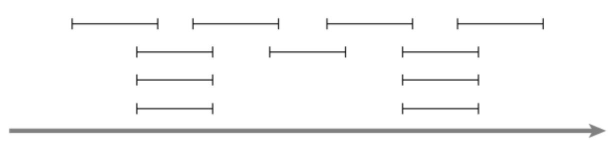
Greedy Algorithms for Interval Scheduling

Heuristic 3: Fewest Conflicts

Schedule a compatible request with the fewest remaining conflicts.

Optimal?

Counter-example:



Greedy Algorithms for Interval Scheduling

Heuristic 4: Finish First

Schedule a compatible request with the smallest finish time.

Greedy Algorithms for Interval Scheduling

Heuristic 4: Finish First

Schedule a compatible request with the smallest finish time.

Optimal?

Greedy Algorithms for Interval Scheduling

Heuristic 4: Finish First

Schedule a compatible request with the smallest finish time.

Optimal?

Counter-example? Let's try and prove it.

Exercise: Formalize the algorithm (pseudocode)

Heuristic 4: Finish First

Exercise: Formalize the algorithm (pseudocode)

Heuristic 4: Finish First

Algorithm: FinishFirst

Let S be an initially empty set.
while σ is not empty **do**
  Choose $r_i \in \sigma$ with the smallest finish time (break ties arbitrarily).
  Add r_i to S.
  Remove all incompatible requests in σ.
end
return S

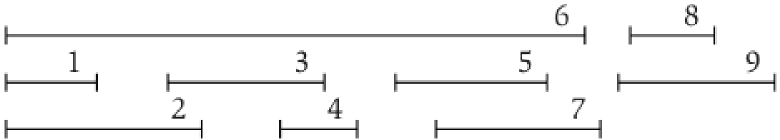
Exercise: Formalize the algorithm (pseudocode)

Heuristic 4: Finish First

Algorithm: FinishFirst

Let S be an initially empty set.
while σ is not empty **do**
    Choose $r_i \in \sigma$ with the smallest finish time (break ties arbitrarily).
    Add r_i to S.
    Remove all incompatible requests in σ.
end
return S

Sample Run (What is $|S|$?)



Analysis of FinishFirst

Observation 1

Immediate from the definition of FinishFirst, *S is compatible.*

Analysis of FinishFirst

Observation 1

Immediate from the definition of FinishFirst*, S is compatible.*

Showing Optimality

Let S^* be an optimal solution.

- We can show the strong claim that $S = S^*$.

Analysis of FinishFirst

Observation 1

Immediate from the definition of FinishFirst, S is compatible.

Showing Optimality

Let S^* be an optimal solution.

- We can show the strong claim that $S = S^*$.
- Can there be multiple S^*?

ANALYSIS OF FINISHFIRST

Observation 1

Immediate from the definition of FINISHFIRST, S is compatible.

Showing Optimality

Let S^* be an optimal solution.

- We can show the strong claim that $S = S^*$.
- Can there be multiple S^*? Yes.

Analysis of FinishFirst

Observation 1

Immediate from the definition of FinishFirst, S *is compatible.*

Showing Optimality

Let S^* be an optimal solution.

- We can show the strong claim that $S = S^*$.
- Can there be multiple S^*? Yes.
- Hence, we can show the weaker claim of $|S| = |S^*|$ for this problem.

Analysis of FinishFirst

Observation 1

Immediate from the definition of FinishFirst, *S is compatible.*

Showing Optimality

Let S^* be an optimal solution.

- We can show the strong claim that $S = S^*$.
- Can there be multiple S^*? Yes.
- Hence, we can show the weaker claim of $|S| = |S^*|$ for this problem.
- Technique: "Always stays ahead"

Analysis of FinishFirst

Observation 1

Immediate from the definition of FinishFirst, *S is compatible.*

Showing Optimality

Let S^* be an optimal solution.

- We can show the strong claim that $S = S^*$.
- Can there be multiple S^*? Yes.
- Hence, we can show the weaker claim of $|S| = |S^*|$ for this problem.
- Technique: "Always stays ahead"
  - At every time step i, $|S_i| \geq |S_i^*|$.

Stays Ahead Analysis

At every round, we are at least as any other solution

- Label $S = \langle i_1, \ldots, i_k \rangle$ such that $f_{i_u} < f_{i_v}$ for $u < v$.
- Label $S^* = \langle j_1, \ldots, j_m \rangle$ such that $f_{j_u} < f_{j_v}$ for $u < v$.

Stays Ahead Analysis

- Label $S = \langle i_1, \ldots, i_k \rangle$ such that $f_{i_u} < f_{i_v}$ for $u < v$.
- Label $S^* = \langle j_1, \ldots, j_m \rangle$ such that $f_{j_u} < f_{j_v}$ for $u < v$.

Lemma 1

For all i_r, j_r with $r \leq k$, we have $f_{i_r} \leq f_{j_r}$

Proof.

Stays Ahead Analysis

- Label $S = \langle i_1, \ldots, i_k \rangle$ such that $f_{i_u} < f_{i_v}$ for $u < v$.
- Label $S^* = \langle j_1, \ldots, j_m \rangle$ such that $f_{j_u} < f_{j_v}$ for $u < v$.

Lemma 1

For all i_r, j_r with $r \leq k$, we have $f_{i_r} \leq f_{j_r}$

Proof.

The proof is by induction.

- For $r = 1$, the claim is true as FinishFirst first selects the request with the earliest finish time.

Stays Ahead Analysis

- Label $S = \langle i_1, \ldots, i_k \rangle$ such that $f_{i_u} < f_{i_v}$ for $u < v$.
- Label $S^* = \langle j_1, \ldots, j_m \rangle$ such that $f_{j_u} < f_{j_v}$ for $u < v$.

Lemma 1

For all i_r, j_r with $r \leq k$, we have $f_{i_r} \leq f_{j_r}$

Proof.

The proof is by induction.

- For $r = 1$, the claim is true as FinishFirst first selects the request with the earliest finish time.
- Assume true for $r - 1$.
  - By the induction hypothesis, we have that $f_{i_{r-1}} \leq f_{j_{r-1}}$.
  - The only way for S to fall behind S^* would be for FinishFirst to choose a request q with $f_q > f_{j_r}$, but this is a contradiction.

$\square$

Stays Ahead Analysis

- Label $S = \langle i_1, \ldots, i_k \rangle$ such that $f_{i_u} < f_{i_v}$ for $u < v$.
- Label $S^* = \langle j_1, \ldots, j_m \rangle$ such that $f_{j_u} < f_{j_v}$ for $u < v$.

Lemma 1

For all i_r, j_r with $r \leq k$, we have $f_{i_r} \leq f_{j_r}$

The optimality of FinishFirst, essentially, follows immediately from Lemma 1.

FinishFirst is Optimal

- Label $S = \langle i_1, \ldots, i_k \rangle$ such that $f_{i_u} < f_{i_v}$ for $u < v$.
- Label $S^* = \langle j_1, \ldots, j_m \rangle$ such that $f_{j_u} < f_{j_v}$ for $u < v$.

Theorem 2

FinishFirst produces an optimal schedule.

Proof.

FinishFirst is Optimal

- Label $S = \langle i_1, \ldots, i_k \rangle$ such that $f_{i_u} < f_{i_v}$ for $u < v$.
- Label $S^* = \langle j_1, \ldots, j_m \rangle$ such that $f_{j_u} < f_{j_v}$ for $u < v$.

Theorem 2

FinishFirst produces an optimal schedule.

Proof.

By way of contradiction, assume that $|S^*| > |S|$. This implies that $m > k$. Lemma 1 shows that FinishFirst is ahead for all the k requests. That means it would be able to add the $(k+1)$-st item of S^*. As it did not, this contradicts the definition of FinishFirst. $\square$

Implementation and Running Time

Algorithm: FinishFirst

Let S be an initially empty set.
while σ is not empty **do**
  Choose $r_i \in \sigma$ with the smallest finish time (break ties arbitrarily).
  Add r_i to S.
  Remove all incompatible request in σ.
end
return S

Implementation Details

Implementation and Running Time

Algorithm: FinishFirst

Let S be an initially empty set.
while σ is not empty **do**
  Choose $r_i \in \sigma$ with the smallest finish time (break ties arbitrarily).
  Add r_i to S.
  Remove all incompatible request in σ.
end
return S

Implementation Details

- Choose request with smallest finish time:
- Remove incompatible requests:

Implementation and Running Time

Algorithm: FinishFirst

Let S be an initially empty set.
while σ is not empty **do**
  Choose $r_i \in \sigma$ with the smallest finish time (break ties arbitrarily).
  Add r_i to S.
  Remove all incompatible request in σ.
end
return S

Implementation Details

- Choose request with smallest finish time: Before processing, sort requests: $O(n \log n)$.
- Remove incompatible requests:

Implementation and Running Time

Algorithm: FinishFirst

Let S be an initially empty set.
while σ is not empty **do**
  Choose $r_i \in \sigma$ with the smallest finish time (break ties arbitrarily).
  Add r_i to S.
  Remove all incompatible request in σ.
end
return S

Implementation Details

- Choose request with smallest finish time: Before processing, sort requests: $O(n \log n)$.
- Remove incompatible requests: Advance in sorted order until a request with a compatible start time.

Implementation and Running Time

Algorithm: FinishFirst

Let S be an initially empty set.
while σ is not empty **do**
  Choose $r_i \in \sigma$ with the smallest finish time (break ties arbitrarily).
  Add r_i to S.
  Remove all incompatible request in σ.
end
return S

Implementation Details

- Choose request with smallest finish time: Before processing, sort requests: $O(n \log n)$.
- Remove incompatible requests: Advance in sorted order until a request with a compatible start time.

Overall:

$O(n \log n) + O(n) = O(n \log n)$

Interval Extensions

- Online variant: Requests are presented in a specific order to the algorithm. At request i, the algorithm does not know n nor $r_{i+1}, \ldots, r_n$.

Interval Extensions

- Online variant: Requests are presented in a specific order to the algorithm. At request i, the algorithm does not know n nor $r_{i+1}, \ldots, r_n$.
- Add a value to the intervals (online/offline). Now objective is to maximize the total value of scheduled intervals.

Interval Extensions

- Online variant: Requests are presented in a specific order to the algorithm. At request i, the algorithm does not know n nor $r_{i+1}, \ldots, r_n$.
- Add a value to the intervals (online/offline). Now objective is to maximize the total value of scheduled intervals.
- Scheduling all intervals: Interval Colouring Problem.

Interval Extensions

- Online variant: Requests are presented in a specific order to the algorithm. At request i, the algorithm does not know n nor $r_{i+1}, \ldots, r_n$.
- Add a value to the intervals (online/offline). Now objective is to maximize the total value of scheduled intervals.
- Scheduling all intervals: Interval Colouring Problem.
  - Unlimited resources and the algorithm must produce multiple compatible schedules that cover all the requests (without duplicates between the schedules).

Interval Extensions

- Online variant: Requests are presented in a specific order to the algorithm. At request i, the algorithm does not know n nor $r_{i+1}, \ldots, r_n$.
- Add a value to the intervals (online/offline). Now objective is to maximize the total value of scheduled intervals.
- Scheduling all intervals: Interval Colouring Problem.
  - Unlimited resources and the algorithm must produce multiple compatible schedules that cover all the requests (without duplicates between the schedules).
  - Objective: Minimize the number of schedules.

What time is now???


IT'S
MORPHIN
TIME!
MIGHTY MORPHIN
POWER RANGERS
ONCE & ALWAYS
© 2023 HASBRO

Appendix

References

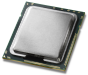
IMAGE SOURCES I

https://www.cse.unsw.edu.au/~cs1521/17s2/lecs/notices/slide068.html

http://mediablogrueil.blogspot.fr/2012/11/one-page-design-effet-de-mode-ou-reel.html

http://www.culturizame.es/articulo/nuestro-pequeno-diccionario-de-tecnologia



http://computer-help-tips.blogspot.fr/2011/04/different-types-of-computer-processors.html

Image Sources II

WISCONSIN UNIVERSITY OF WISCONSIN–MADISON https://brand.wisc.edu/web/logos/