Greedy Algorithms: Interval Scheduling

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 7

Greedy Algorithms

- Need to make a sequence of choices to achieve a global optimum

Greedy Algorithms

- Need to make a sequence of choices to achieve a global optimum
- At each stage, make the next choice based on some local criterion

Greedy Algorithms

- Need to make a sequence of choices to achieve a global optimum
- At each stage, make the next choice based on some local criterion
- Never go back and revise an earlier decision

Greedy Algorithms

- Need to make a sequence of choices to achieve a global optimum
- At each stage, make the next choice based on some local criterion
- Never go back and revise an earlier decision
- Drastically reduces space to search for solutions

Greedy Algorithms

- Need to make a sequence of choices to achieve a global optimum
- At each stage, make the next choice based on some local criterion
- Never go back and revise an earlier decision
- Drastically reduces space to search for solutions
- How to prove that local choices achieve global optimum?

Greedy Algorithms

- Need to make a sequence of choices to achieve a global optimum
- At each stage, make the next choice based on some local criterion
- Never go back and revise an earlier decision
- Drastically reduces space to search for solutions
- How to prove that local choices achieve global optimum?

Examples

- Dijkstra's algorithm
  - Local rule: freeze the distance to nearest unvisited vertex
  - Global optimum: distance assigned to each vertex is shortest distance from source

Greedy Algorithms

- Need to make a sequence of choices to achieve a global optimum
- At each stage, make the next choice based on some local criterion
- Never go back and revise an earlier decision
- Drastically reduces space to search for solutions
- How to prove that local choices achieve global optimum?

Examples

- Prim's algorithm
  - Local rule: add to the spanning tree nearest non-tree vertex
  - Global optimum: final spanning tree is minimum cost spanning tree

Greedy Algorithms

- Need to make a sequence of choices to achieve a global optimum
- At each stage, make the next choice based on some local criterion
- Never go back and revise an earlier decision
- Drastically reduces space to search for solutions
- How to prove that local choices achieve global optimum?

Examples

- Kruskal's algorithm
  - Local rule: add to the current set of edges the smallest edge that does not form a cycle
  - Global optimum: final spanning tree is minimum cost spanning tree

Interval scheduling

- IIT Madras has a special video classroom for delivering online lectures

Interval scheduling

- IIT Madras has a special video classroom for delivering online lectures
- Different teachers want to book the classroom

Interval scheduling

- IIT Madras has a special video classroom for delivering online lectures
- Different teachers want to book the classroom
- Slot for instructor i starts at $s(i)$ and finishes at $f(i)$

Interval scheduling

- IIT Madras has a special video classroom for delivering online lectures
- Different teachers want to book the classroom
- Slot for instructor i starts at $s(i)$ and finishes at $f(i)$
- Slots may overlap, so not all bookings can be honoured

Interval scheduling

- IIT Madras has a special video classroom for delivering online lectures
- Different teachers want to book the classroom
- Slot for instructor i starts at $s(i)$ and finishes at $f(i)$
- Slots may overlap, so not all bookings can be honoured
- Choose a subset of bookings to maximize the number of teachers who get to use the room

Interval scheduling

- IIT Madras has a special video classroom for delivering online lectures
- Different teachers want to book the classroom
- Slot for instructor i starts at $s(i)$ and finishes at $f(i)$
- Slots may overlap, so not all bookings can be honoured
- Choose a subset of bookings to maximize the number of teachers who get to use the room

Greedy approach

- Pick the next booking to allot based on a local strategy

Interval scheduling

- IIT Madras has a special video classroom for delivering online lectures
- Different teachers want to book the classroom
- Slot for instructor i starts at $s(i)$ and finishes at $f(i)$
- Slots may overlap, so not all bookings can be honoured
- Choose a subset of bookings to maximize the number of teachers who get to use the room

Greedy approach

- Pick the next booking to allot based on a local strategy
- Remove all bookings that overlap with the chosen slot

Interval scheduling

- IIT Madras has a special video classroom for delivering online lectures
- Different teachers want to book the classroom
- Slot for instructor i starts at $s(i)$ and finishes at $f(i)$
- Slots may overlap, so not all bookings can be honoured
- Choose a subset of bookings to maximize the number of teachers who get to use the room

Greedy approach

- Pick the next booking to allot based on a local strategy
- Remove all bookings that overlap with the chosen slot
- Argue that this sequence of bookings will maximize the number of teachers who get to use the room

Interval scheduling

- IIT Madras has a special video classroom for delivering online lectures
- Different teachers want to book the classroom
- Slot for instructor i starts at $s(i)$ and finishes at $f(i)$
- Slots may overlap, so not all bookings can be honoured
- Choose a subset of bookings to maximize the number of teachers who get to use the room

Greedy approach

- Pick the next booking to allot based on a local strategy
- Remove all bookings that overlap with the chosen slot
- Argue that this sequence of bookings will maximize the number of teachers who get to use the room
- What is a sound local strategy?

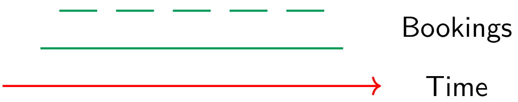
Greedy strategies for interval scheduling

- Strategy 1
  Choose the booking whose starting time is earliest

Greedy strategies for interval scheduling

- **Strategy 1**
  Choose the booking whose starting time is earliest

Counterexample



Greedy strategies for interval scheduling

- Strategy 1
  Choose the booking whose starting time is earliest

- Strategy 2
  Choose the booking spanning the shortest interval

Greedy strategies for interval scheduling

- Strategy 1
  Choose the booking whose starting time is earliest
- Strategy 2
  Choose the booking spanning the shortest interval

Counterexample

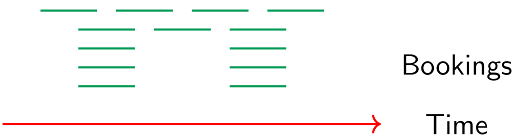
Greedy strategies for interval scheduling

- Strategy 1
  Choose the booking whose starting time is earliest
- Strategy 2
  Choose the booking spanning the shortest interval
- Strategy 3
  Choose the booking that overlaps with minimum number of other bookings

Greedy strategies for interval scheduling

- Strategy 1
  Choose the booking whose starting time is earliest
- Strategy 2
  Choose the booking spanning the shortest interval
- Strategy 3
  Choose the booking that overlaps with minimum number of other bookings

Counterexample



Greedy strategies for interval scheduling

- Strategy 1
  Choose the booking whose starting time is earliest
- Strategy 2
  Choose the booking spanning the shortest interval
- Strategy 3
  Choose the booking that overlaps with minimum number of other bookings
- Strategy 4
  Choose the booking whose finish time is the earliest

Greedy strategies for interval scheduling

- Strategy 1
  Choose the booking whose starting time is earliest
- Strategy 2
  Choose the booking spanning the shortest interval
- Strategy 3
  Choose the booking that overlaps with minimum number of other bookings
- Strategy 4
  Choose the booking whose finish time is the earliest
  - Counterexample? Proof of correctness?

Greedy algorithm for interval scheduling

- B is the set of bookings

Greedy algorithm for interval scheduling

- B is the set of bookings
- A is the set of accepted bookings
  - Initially, A is empty

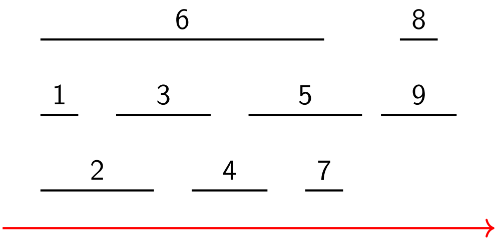
Greedy algorithm for interval scheduling

- B is the set of bookings
- A is the set of accepted bookings
  - Initially, A is empty
- While B is not empty
  - Pick $b \in B$ with earliest finishing time
  - Add b to A
  - Remove from B all bookings that overlap with b

Greedy algorithm for interval scheduling

- B is the set of bookings
- A is the set of accepted bookings
  - Initially, A is empty
- While B is not empty
  - Pick $b \in B$ with earliest finishing time
  - Add b to A
  - Remove from B all bookings that overlap with b

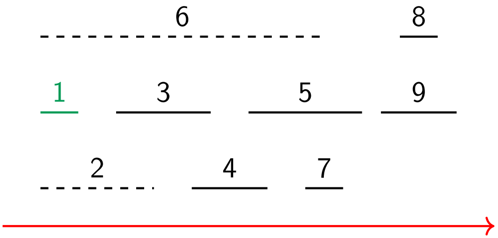
The algorithm in action



Greedy algorithm for interval scheduling

- B is the set of bookings
- A is the set of accepted bookings
  - Initially, A is empty
- While B is not empty
  - Pick $b \in B$ with earliest finishing time
  - Add b to A
  - Remove from B all bookings that overlap with b

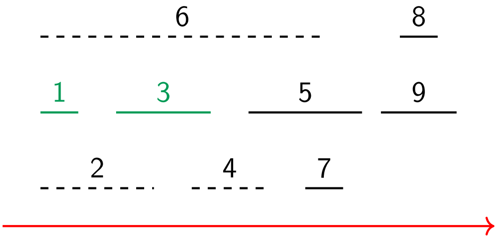
The algorithm in action



Greedy algorithm for interval scheduling

- B is the set of bookings
- A is the set of accepted bookings
  - Initially, A is empty
- While B is not empty
  - Pick $b \in B$ with earliest finishing time
  - Add b to A
  - Remove from B all bookings that overlap with b

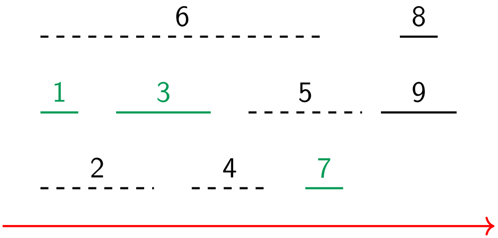
The algorithm in action



Greedy algorithm for interval scheduling

- B is the set of bookings
- A is the set of accepted bookings
  - Initially, A is empty
- While B is not empty
  - Pick $b \in B$ with earliest finishing time
  - Add b to A
  - Remove from B all bookings that overlap with b

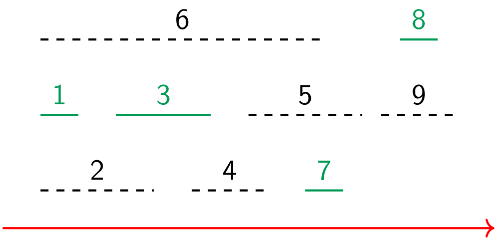
The algorithm in action



Greedy algorithm for interval scheduling

- B is the set of bookings
- A is the set of accepted bookings
  - Initially, A is empty
- While B is not empty
  - Pick $b \in B$ with earliest finishing time
  - Add b to A
  - Remove from B all bookings that overlap with b

The algorithm in action



Correctness

- Our algorithm produces a solution A

Correctness

- Our algorithm produces a solution A
- Let O be an optimal set of accepted bookings

Correctness

- Our algorithm produces a solution A
- Let O be an optimal set of accepted bookings
- A and O need not be identical
  - Could have multiple allocations of the same size

Correctness

- Our algorithm produces a solution A
- Let O be an optimal set of accepted bookings
- A and O need not be identical
  - Could have multiple allocations of the same size
- Just show that $|A| = |O|$
  - Both sets of bookings are of the same size

Correctness

- Our algorithm produces a solution A
- Let O be an optimal set of accepted bookings
- A and O need not be identical
  - Could have multiple allocations of the same size
- Just show that $|A| = |O|$
  - Both sets of bookings are of the same size
- Let $A = \{i_1, i_2, \ldots, i_k\}$
  - Assume sorted
  - $f(i_1) \leq s(i_2)$, $f(i_2) \leq s(i_3)$, ...

Correctness

- Our algorithm produces a solution A
- Let O be an optimal set of accepted bookings
- A and O need not be identical
  - Could have multiple allocations of the same size
- Just show that $|A| = |O|$
  - Both sets of bookings are of the same size

- Let $A = \{i_1, i_2, \ldots, i_k\}$
  - Assume sorted
  - $f(i_1) \leq s(i_2)$, $f(i_2) \leq s(i_3)$, ...
- Let $O = \{j_1, j_2, \ldots, j_m\}$
  - Also sorted
  - $f(j_1) \leq s(j_2)$, $f(j_2) \leq s(j_3)$, ...

Correctness

- Our algorithm produces a solution A
- Let O be an optimal set of accepted bookings
- A and O need not be identical
  - Could have multiple allocations of the same size
- Just show that $|A| = |O|$
  - Both sets of bookings are of the same size
- Let $A = \{i_1, i_2, \ldots, i_k\}$
  - Assume sorted
  - $f(i_1) \leq s(i_2)$, $f(i_2) \leq s(i_3)$, ...
- Let $O = \{j_1, j_2, \ldots, j_m\}$
  - Also sorted
  - $f(j_1) \leq s(j_2)$, $f(j_2) \leq s(j_3)$, ...
- Our goal is to show that $k = m$

Greedy algorithm stays ahead

- $A = \{i_1, i_2, \ldots, i_k\}$
- $O = \{j_1, j_2, \ldots, j_m\}$

Greedy algorithm stays ahead

- $A = \{i_1, i_2, \ldots, i_k\}$
- $O = \{j_1, j_2, \ldots, j_m\}$
- Claim For each $\ell \leq k$, $f(i_\ell) \leq f(j_\ell)$

Greedy algorithm stays ahead

- $A = \{i_1, i_2, \ldots, i_k\}$
- $O = \{j_1, j_2, \ldots, j_m\}$
- Claim For each $\ell \leq k$, $f(i_\ell) \leq f(j_\ell)$
- Proof By induction of ℓ

Greedy algorithm stays ahead

- $A = \{i_1, i_2, \ldots, i_k\}$
- $O = \{j_1, j_2, \ldots, j_m\}$
- Claim For each $\ell \leq k$, $f(i_\ell) \leq f(j_\ell)$
- Proof By induction of ℓ
- Base case: $\ell = 1$

  By greedy strategy, i_1 has earliest overall finish time

Greedy algorithm stays ahead

- $A = \{i_1, i_2, \dots, i_k\}$
- $O = \{j_1, j_2, \dots, j_m\}$
- Claim For each $\ell \leq k$, $f(i_\ell) \leq f(j_\ell)$
- Proof By induction of ℓ
- Base case: $\ell = 1$

  By greedy strategy, i_1 has earliest overall finish time
- Induction step: Assume $f(i_{\ell-1}) \leq f(j_{\ell-1})$

Greedy algorithm stays ahead

- $A = \{i_1, i_2, \ldots, i_k\}$
- $O = \{j_1, j_2, \ldots, j_m\}$
- Claim For each $\ell \leq k$, $f(i_\ell) \leq f(j_\ell)$
- Proof By induction of ℓ
- Base case: $\ell = 1$

  By greedy strategy, i_1 has earliest overall finish time
- Induction step: Assume $f(i_{\ell-1}) \leq f(j_{\ell-1})$
  - $f(i_{\ell-1}) \leq f(j_{\ell-1}) \leq s(j_\ell)$

Greedy algorithm stays ahead

- $A = \{i_1, i_2, \ldots, i_k\}$
- $O = \{j_1, j_2, \ldots, j_m\}$
- Claim For each $\ell \leq k$, $f(i_\ell) \leq f(j_\ell)$
- Proof By induction of ℓ
- Base case: $\ell = 1$

  By greedy strategy, i_1 has earliest overall finish time
- Induction step: Assume $f(i_{\ell-1}) \leq f(j_{\ell-1})$
  - $f(i_{\ell-1}) \leq f(j_{\ell-1}) \leq s(j_\ell)$
  - If $f(j_\ell) < f(i_\ell)$, greedy strategy would pick j_ℓ

Greedy algorithm stays ahead

- $A = \{i_1, i_2, \ldots, i_k\}$
- $O = \{j_1, j_2, \ldots, j_m\}$
- Claim For each $\ell \leq k$, $f(i_\ell) \leq f(j_\ell)$
- Proof By induction of ℓ
- Base case: $\ell = 1$

  By greedy strategy, i_1 has earliest overall finish time
- Induction step: Assume $f(i_{\ell-1}) \leq f(j_{\ell-1})$
  - $f(i_{\ell-1}) \leq f(j_{\ell-1}) \leq s(j_\ell)$
  - If $f(j_\ell) < f(i_\ell)$, greedy strategy would pick j_ℓ
  - We must have $f(i_\ell) \leq f(j_\ell)$

Greedy strategy is optimal

- $A = \{i_1, i_2, \ldots, i_k\}$
- $O = \{j_1, j_2, \ldots, j_m\}$

Greedy strategy is optimal

- $A = \{i_1, i_2, \ldots, i_k\}$
- $O = \{j_1, j_2, \ldots, j_m\}$
- Suppose $m > k$

Greedy strategy is optimal

- $A = \{i_1, i_2, \ldots, i_k\}$
- $O = \{j_1, j_2, \ldots, j_m\}$
- Suppose $m > k$
- We know $f(i_k) \leq f(j_k)$

Greedy strategy is optimal

- $A = \{i_1, i_2, \ldots, i_k\}$
- $O = \{j_1, j_2, \ldots, j_m\}$
- Suppose $m > k$
- We know $f(i_k) \leq f(j_k)$
- Greedy strategy stops when B is empty

Greedy strategy is optimal

- $A = \{i_1, i_2, \ldots, i_k\}$
- $O = \{j_1, j_2, \ldots, j_m\}$
- Suppose $m > k$
- We know $f(i_k) \leq f(j_k)$
- Greedy strategy stops when B is empty
  - Consder request j_{k+1}
  - Since $f(i_k) \leq f(j_k) \leq s(j_{k+1})$, this request is compatible with A

Greedy strategy is optimal

- $A = \{i_1, i_2, \ldots, i_k\}$
- $O = \{j_1, j_2, \ldots, j_m\}$
- Suppose $m > k$
- We know $f(i_k) \leq f(j_k)$
- Greedy strategy stops when B is empty
  - Consder request j_{k+1}
  - Since $f(i_k) \leq f(j_k) \leq s(j_{k+1})$, this request is compatible with A
  - j_{k+1} must still be in B

Greedy strategy is optimal

- $A = \{i_1, i_2, \ldots, i_k\}$
- $O = \{j_1, j_2, \ldots, j_m\}$
- Suppose $m > k$
- We know $f(i_k) \leq f(j_k)$
- Greedy strategy stops when B is empty
  - Consder request j_{k+1}
  - Since $f(i_k) \leq f(j_k) \leq s(j_{k+1})$, this request is compatible with A
  - j_{k+1} must still be in B
  - B is not empty after choosing A, contradiction!

Implementation

- Initially, sort n bookings by finish time — $O(n \log n)$
  - Renumber bookings to reflect this sorted order

Implementation

- Initially, sort n bookings by finish time — $O(n \log n)$
  - Renumber bookings to reflect this sorted order
- Record start and finish times in array/dictionary
  - `S[i]` is starting time of request `i`
  - `F[i]` is finish time of request `i`

Implementation

- Initially, sort n bookings by finish time — $O(n \log n)$
  - Renumber bookings to reflect this sorted order
- Record start and finish times in array/dictionary
  - `S[i]` is starting time of request `i`
  - `F[i]` is finish time of request `i`
- Add first booking to A

Implementation

- Initially, sort n bookings by finish time — $O(n \log n)$
  - Renumber bookings to reflect this sorted order
- Record start and finish times in array/dictionary
  - `S[i]` is starting time of request `i`
  - `F[i]` is finish time of request `i`
- Add first booking to A
- In general, after adding booking `j` to A, Find the smallest `r` with `S[r] > F[j]`
  - Single scan, $O(n)$ overall

Summary

- A greedy algorithm makes a sequence of locally optimal choices to achieve a global optimum

Summary

- A greedy algorithm makes a sequence of locally optimal choices to achieve a global optimum
- The algorithm never goes back and reconsiders on a choice

Summary

- A greedy algorithm makes a sequence of locally optimal choices to achieve a global optimum
- The algorithm never goes back and reconsiders on a choice
- Drastically reduces space to search for solutions, but need to show prove that local strategy is globally optimal

Summary

- A greedy algorithm makes a sequence of locally optimal choices to achieve a global optimum
- The algorithm never goes back and reconsiders on a choice
- Drastically reduces space to search for solutions, but need to show prove that local strategy is globally optimal
- Interval scheduling — many "natural" greedy strategies

Summary

- A greedy algorithm makes a sequence of locally optimal choices to achieve a global optimum
- The algorithm never goes back and reconsiders on a choice
- Drastically reduces space to search for solutions, but need to show prove that local strategy is globally optimal
- Interval scheduling — many “natural” greedy strategies
- Most of them are wrong!

Summary

- A greedy algorithm makes a sequence of locally optimal choices to achieve a global optimum
- The algorithm never goes back and reconsiders on a choice
- Drastically reduces space to search for solutions, but need to show prove that local strategy is globally optimal
- Interval scheduling — many "natural" greedy strategies
- Most of them are wrong!
- Correct strategy needs a proof

Summary

- A greedy algorithm makes a sequence of locally optimal choices to achieve a global optimum
- The algorithm never goes back and reconsiders on a choice
- Drastically reduces space to search for solutions, but need to show prove that local strategy is globally optimal
- Interval scheduling — many "natural" greedy strategies
- Most of them are wrong!
- Correct strategy needs a proof
- One way is to show that greedy solution "stays ahead", step by step, of any optimal solutions