CMP_SC 3050: Introduction to Greedy Algorithms

Rohit Chadha

September 25, 2014

Used for optimization problems (i.e. when we have to optimize some objective)

Idea:

- When we have a choice to make, make the one that looks best right now.

Used for optimization problems (i.e. when we have to optimize some objective)

Idea:

- When we have a choice to make, make the one that looks best right now.
- Make a locally optimal choice in hope of getting a globally optimal solution.

Used for optimization problems (i.e. when we have to optimize some objective)

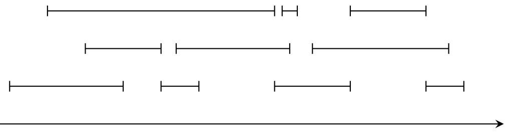
Idea:

- When we have a choice to make, make the one that looks best right now.
- Make a locally optimal choice in hope of getting a globally optimal solution.
- Warning: Greedy algorithms dont always yield an optimal solution, but many times they do

Activity selection

Input: A set $\{a_1, a_2, \ldots, a_n\}$ of activities with start and finish times that require exclusive use of a common resource. $s(i)$ is the start time of activity i and $f(i)$ is the finish time of activity i.

Output: Schedule as many activities as possible

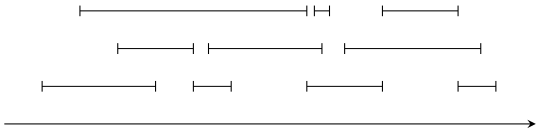


Activity selection

Input: A set $\{a_1, a_2, \ldots, a_n\}$ of activities with start and finish times that require exclusive use of a common resource. $s(i)$ is the start time of activity i and $f(i)$ is the finish time of activity i.

Output: Schedule as many activities as possible

- Two activities with overlapping intervals cannot both be scheduled!

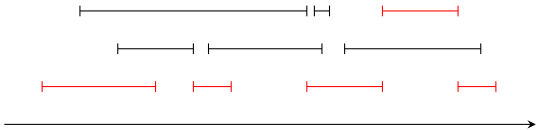


Activity selection

Input: A set $\{a_1, a_2, \ldots, a_n\}$ of activities with start and finish times that require exclusive use of a common resource. $s(i)$ is the start time of activity i and $f(i)$ is the finish time of activity i.

Output: Schedule as many activities as possible

- Two activities with overlapping intervals cannot both be scheduled!



Greedy Template

```
Initially R is the set of all activities
A is empty (* A will store all the activities that will be scheduled *)
while R is not empty
    choose i ∈ R
    add i to A
    remove from R all activities that overlap with i
return the set A
```

Greedy Template

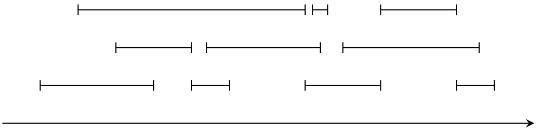
```
Initially R is the set of all activities
A is empty (* A will store all the activities that will be scheduled *)
while R is not empty
    choose i ∈ R
    add i to A
    remove from R all activities that overlap with i
return the set A
```

Main task: Decide the order in which to process activities in R

ES SP FC EF

Earliest Start Time

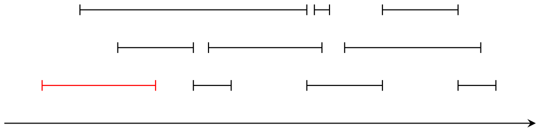
Process activities in the order of their starting times, beginning with those that start earliest.



Back Counter

Earliest Start Time

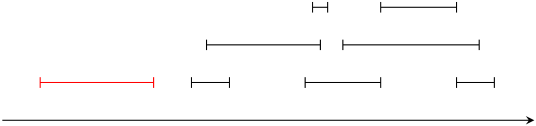
Process activities in the order of their starting times, beginning with those that start earliest.



Back Counter

Earliest Start Time

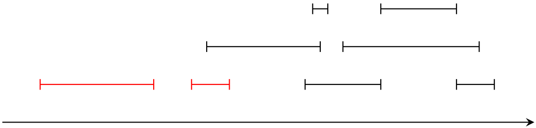
Process activities in the order of their starting times, beginning with those that start earliest.



Back Counter

Earliest Start Time

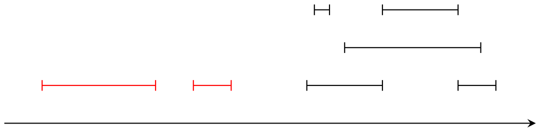
Process activities in the order of their starting times, beginning with those that start earliest.



Back Counter

Earliest Start Time

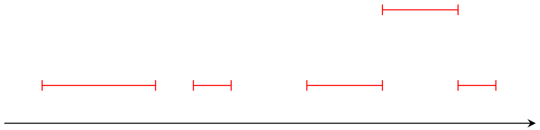
Process activities in the order of their starting times, beginning with those that start earliest.



Back Counter

Earliest Start Time

Process activities in the order of their starting times, beginning with those that start earliest.



Back Counter

Earliest Start Time

Process activities in the order of their starting times, beginning with those that start earliest.

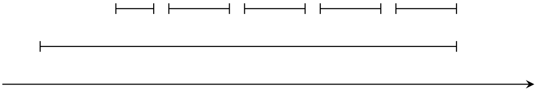


Figure : Counter example for earliest start time

Back Counter

Earliest Start Time

Process activities in the order of their starting times, beginning with those that start earliest.

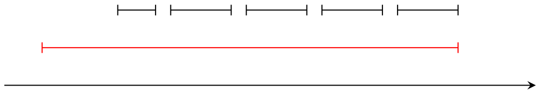


Figure : Counter example for earliest start time

Back Counter

Earliest Start Time

Process activities in the order of their starting times, beginning with those that start earliest.

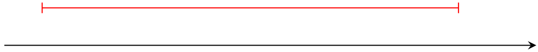
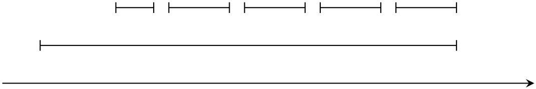


Figure : Counter example for earliest start time

Back Counter

Smallest Processing Time

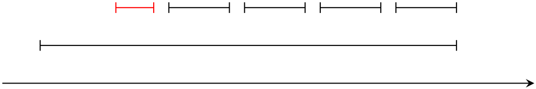
Process activities in the order of processing time, starting with activities that require the shortest processing.



Back Counter

Smallest Processing Time

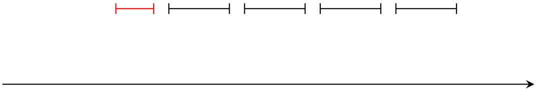
Process activities in the order of processing time, starting with activities that require the shortest processing.



Back Counter

Smallest Processing Time

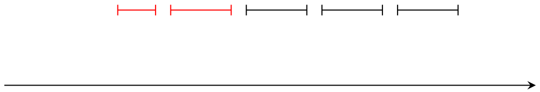
Process activities in the order of processing time, starting with activities that require the shortest processing.



Back Counter

Smallest Processing Time

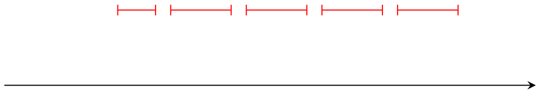
Process activities in the order of processing time, starting with activities that require the shortest processing.



Back Counter

Smallest Processing Time

Process activities in the order of processing time, starting with activities that require the shortest processing.



Back Counter

Smallest Processing Time

Process activities in the order of processing time, starting with activities that require the shortest processing.

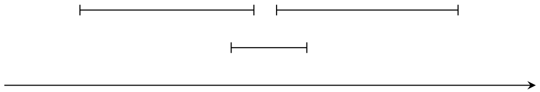


Figure : Counter example for smallest processing time

Back Counter

Smallest Processing Time

Process activities in the order of processing time, starting with activities that require the shortest processing.

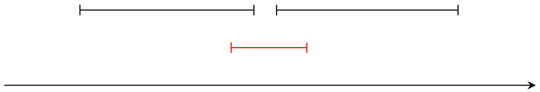


Figure : Counter example for smallest processing time

Back Counter

Smallest Processing Time

Process activities in the order of processing time, starting with activities that require the shortest processing.

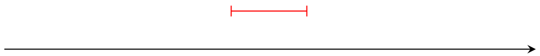
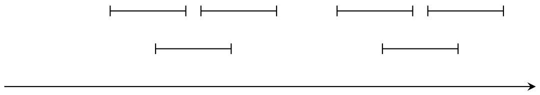


Figure : Counter example for smallest processing time

Back Counter

Fewest Conflicts

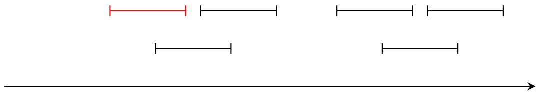
Process activities in that have the fewest “conflicts” first.



Back Counter

Fewest Conflicts

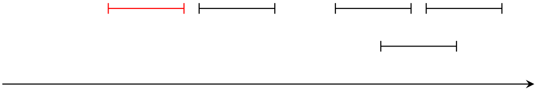
Process activities in that have the fewest "conflicts" first.



Back Counter

Fewest Conflicts

Process activities in that have the fewest “conflicts” first.



Back Counter

Fewest Conflicts

Process activities in that have the fewest “conflicts” first.

Back Counter

Fewest Conflicts

Process activities in that have the fewest “conflicts” first.

Back Counter

Fewest Conflicts

Process activities in that have the fewest “conflicts” first.

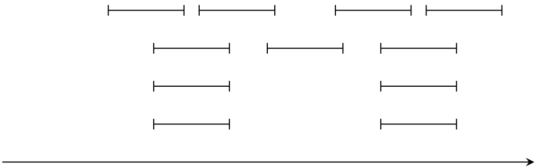


Figure : Counter example for fewest conflicts

Back Counter

Fewest Conflicts

Process activities in that have the fewest "conflicts" first.

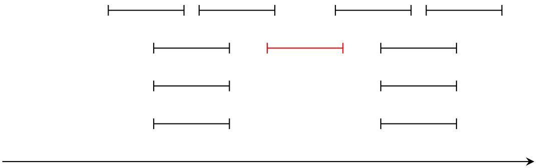


Figure : Counter example for fewest conflicts

Back Counter

Fewest Conflicts

Process activities in that have the fewest "conflicts" first.

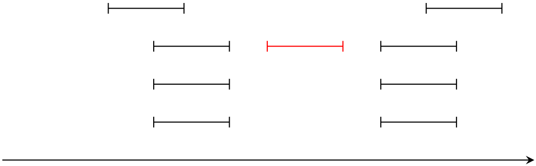


Figure : Counter example for fewest conflicts

Back Counter

Fewest Conflicts

Process activities in that have the fewest “conflicts” first.

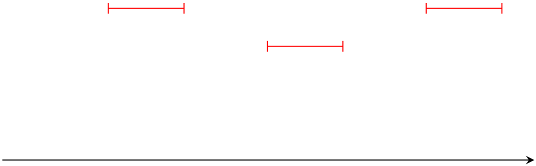
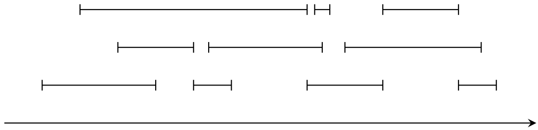


Figure : Counter example for fewest conflicts

Back Counter

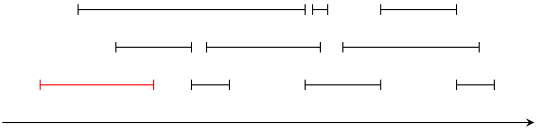
Earliest Finish Time

Process activities in the order of their finishing times, beginning with those that finish earliest.



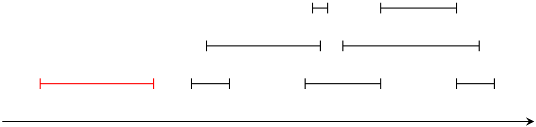
Earliest Finish Time

Process activities in the order of their finishing times, beginning with those that finish earliest.



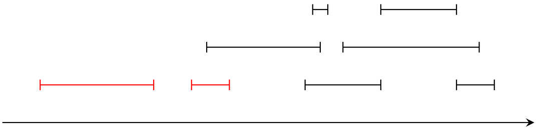
Earliest Finish Time

Process activities in the order of their finishing times, beginning with those that finish earliest.



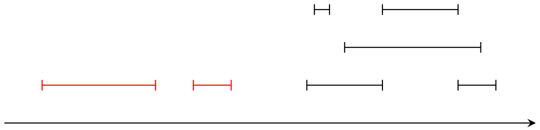
Earliest Finish Time

Process activities in the order of their finishing times, beginning with those that finish earliest.



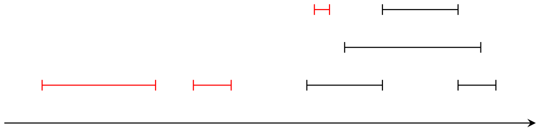
Earliest Finish Time

Process activities in the order of their finishing times, beginning with those that finish earliest.



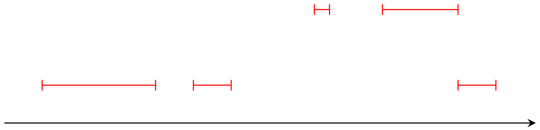
Earliest Finish Time

Process activities in the order of their finishing times, beginning with those that finish earliest.



Earliest Finish Time

Process activities in the order of their finishing times, beginning with those that finish earliest.



Optimal Greedy Algorithm

```
Initially R is the set of all activities
A is empty (* A will store all the activities that will be scheduled *)
while R is not empty
    choose i ∈ R such that finishing time of i is least
    add i to A
    remove from R all activities that overlap with i
return the set A
```

The greedy algorithm that picks activities in the order of their finishing times is optimal.

Why is the greedy algorithm correct and optimal?

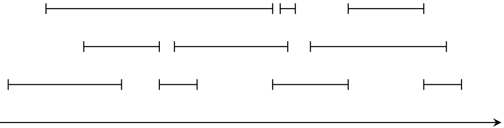
- Clearly the algorithm returns a set of activities that does not have any conflicts

Why is the greedy algorithm correct and optimal?

- Clearly the algorithm returns a set of activities that does not have any conflicts
- For a set of activities R, let O be the optimal set and let A be the set returned by the greedy algorithm. Then $O = A$

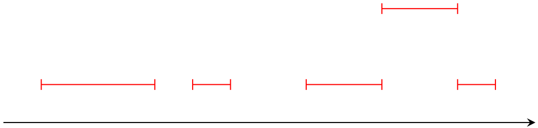
Why is the greedy algorithm correct and optimal?

- Clearly the algorithm returns a set of activities that does not have any conflicts
- For a set of activities R, let O be the optimal set and let A be the set returned by the greedy algorithm. Then $O = A$? There are many optimal solutions!



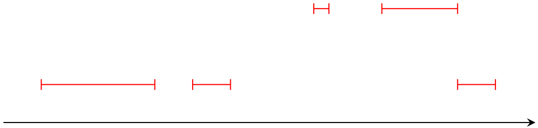
Why is the greedy algorithm correct and optimal?

- Clearly the algorithm returns a set of activities that does not have any conflicts
- For a set of activities R, let O be the optimal set and let A be the set returned by the greedy algorithm. Then $O = A$? There are many optimal solutions!



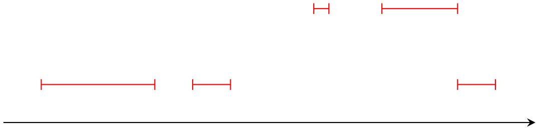
Why is the greedy algorithm correct and optimal?

- Clearly the algorithm returns a set of activities that does not have any conflicts
- For a set of activities R, let O be the optimal set and let A be the set returned by the greedy algorithm. Then $O = A$? There are many optimal solutions!



Why is the greedy algorithm correct and optimal?

- Clearly the algorithm returns a set of activities that does not have any conflicts
- For a set of activities R, let O be the optimal set and let A be the set returned by the greedy algorithm. Then $O = A$? There are many optimal solutions!



Instead we argue that O and A have the same number of elements

Why is the greedy algorithm correct?

- Let $i_1, i_2, \ldots i_k$ be the activities in A, listed in the order they were added

Why is the greedy algorithm correct?

- Let $i_1, i_2, \ldots i_k$ be the activities in A, listed in the order they were added
- Let $j_1, j_2, \ldots j_m$ be the activities in O, ordered from left to right according to the start and finish times

Why is the greedy algorithm correct?

- Let $i_1, i_2, \ldots i_k$ be the activities in A, listed in the order they were added
- Let $j_1, j_2, \ldots j_m$ be the activities in O, ordered from left to right according to the start and finish times
- Goal: To argue $k = m$
- Observation: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$

Why is the greedy algorithm correct?

- Let $i_1, i_2, \ldots i_k$ be the activities in A, listed in the order they were added
- Let $j_1, j_2, \ldots j_m$ be the activities in O, ordered from left to right according to the start and finish times
- Goal: To argue $k = m$
- Observation: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$
- Suppose $m > k$. The observation shows that $f(i_k) \leq f(j_k)$

Why is the greedy algorithm correct?

- Let $i_1, i_2, \ldots i_k$ be the activities in A, listed in the order they were added
- Let $j_1, j_2, \ldots j_m$ be the activities in O, ordered from left to right according to the start and finish times
- Goal: To argue $k = m$
- Observation: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$
- Suppose $m > k$. The observation shows that $f(i_k) \leq f(j_k)$
- Now $s(j_{k+1}) \geq f(j_k)$, and so after removing all activities in conflict with $i_1, \ldots i_k$, j_{k+1} remains

Why is the greedy algorithm correct?

- Let $i_1, i_2, \ldots i_k$ be the activities in A, listed in the order they were added
- Let $j_1, j_2, \ldots j_m$ be the activities in O, ordered from left to right according to the start and finish times
- Goal: To argue $k = m$
- Observation: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$
- Suppose $m > k$. The observation shows that $f(i_k) \leq f(j_k)$
- Now $s(j_{k+1}) \geq f(j_k)$, and so after removing all activities in conflict with $i_1, \ldots i_k$, j_{k+1} remains
- Greedy algorithm can still pick more activities!

Why is the observation correct?

Observation: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$

Why is the observation correct?

Observation: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$

- $f(i_1) \leq f(j_1)$ (we picked the activity with the finish time first)

Why is the observation correct?

Observation: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$

- $f(i_1) \leq f(j_1)$ (we picked the activity with the finish time first)
- Now consider the second activity in O and A !

Why is the observation correct?

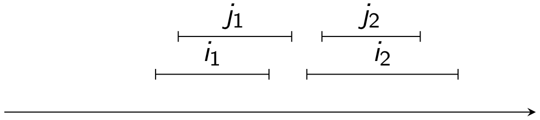
Observation: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$

- $f(i_1) \leq f(j_1)$ (we picked the activity with the finish time first)
- Now consider the second activity in O and A !

Why is the observation correct?

Observation: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$

- $f(i_1) \leq f(j_1)$ (we picked the activity with the finish time first)
- Now consider the second activity in O and A !
- Is this a problem?



Why is the observation correct?

Observation: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$

- $f(i_1) \leq f(j_1)$ (we picked the activity with the finish time first)
- Now consider the second activity in O and A !
- Since, $f(j_1) \leq s(j_2)$, j_2 is in R after removing the activities conflicting with i_1
- Hence $f(i_2) \leq f(j_2)$

Why is the observation correct?

Observation: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$

- $f(i_1) \leq f(j_1)$ (we picked the activity with the finish time first)
- Now consider the second activity in O and A !
- Since, $f(j_1) \leq s(j_2)$, j_2 is in R after removing the activities conflicting with i_1
- Hence $f(i_2) \leq f(j_2)$
- We can continue the same argument for the 3rd, 4th, ... activities

Implementation and Running Time

```
Initially R is the set of all activities
A is empty (* A will store all the activities that will be scheduled *)
while R is not empty
    choose i ∈ R such that finishing time of i is least
    add i to A
    remove from R all activities that overlap with i
return the set A
```

Implementation and Running Time

```
Initially R is the set of all activities
A is empty (* A will store all the activities that will be scheduled *)
while R is not empty
    choose i ∈ R such that finishing time of i is least
    if i does not overlap with activities in A
        add i to A
return the set A
```

Implementation and Running Time

```
Initially R is the set of all activities
A is empty (* A will store all the activities that will be scheduled *)
while R is not empty
    choose i ∈ R such that finishing time of i is least
    if i does not overlap with activities in A
        add i to A
return the set A
```

- Pre-sort all activities based on finishing time. Takes $O(n \log n)$ time
- Now choosing least finishing time is $O(1)$

Implementation and Running Time

```
Initially R is the set of all activities
A is empty (* A will store all the activities that will be scheduled *)
while R is not empty
    choose i ∈ R such that finishing time of i is least
    if i does not overlap with activities in A
        add i to A
return the set A
```

- Pre-sort all activities based on finishing time. Takes $O(n \log n)$ time
- Now choosing least finishing time is $O(1)$
- Keep track of the finishing time of the last activity added to A (which can be implemented as a list). Then check if starting time of i later than that
- Thus, checking non-overlapping is $O(1)$

Implementation and Running Time

```
Initially R is the set of all activities
A is empty (* A will store all the activities that will be scheduled *)
while R is not empty
    choose i ∈ R such that finishing time of i is least
    if i does not overlap with activities in A
        add i to A
return the set A
```

- Pre-sort all activities based on finishing time. Takes $O(n \log n)$ time
- Now choosing least finishing time is $O(1)$
- Keep track of the finishing time of the last activity added to A (which can be implemented as a list). Then check if starting time of i later than that
- Thus, checking non-overlapping is $O(1)$
- Total time $O(n \log n + n) = O(n \log n)$

Extensions

- All activities need not be known at the beginning. Such *online* algorithms are a subject of research

Extensions

- All activities need not be known at the beginning. Such *online* algorithms are a subject of research
- Instead of maximizing the total number of activities, associate *value* with each activity that is scheduled. Try to schedule activities to maximize total value. Will be seen later in this course

Scheduling all Activities

Input A set of lectures, with start and end times

Output Find the minimum number of classrooms, needed to schedule all the lectures such two lectures do not occur at the same time in the same room.

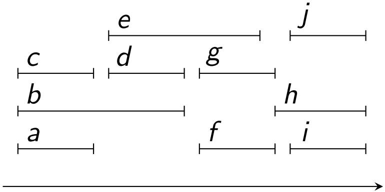


Figure : A schedule that uses 4 classrooms

Scheduling all Activities

Input A set of lectures, with start and end times

Output Find the minimum number of classrooms, needed to schedule all the lectures such two lectures do not occur at the same time in the same room.

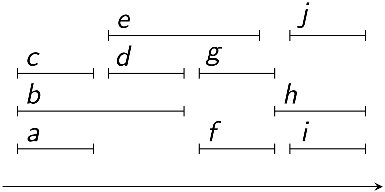


Figure : A schedule that uses 4 classrooms

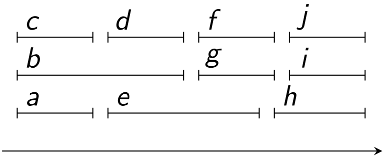


Figure : A schedule requiring 3 classrooms

Greedy Algorithm

```
Initially R is the set of all lectures
d = 0 (* number of classrooms *)
while R is not empty
    choose i ∈ R
    if i can be scheduled in some classroom k
        schedule lecture i in classroom k
    else
        allocate a new classroom d+1 and schedule lecture i in d+1
        d = d+1
```

What order should we process lectures in?

Greedy Algorithm

```
Initially R is the set of all lectures
d = 0 (* number of classrooms *)
while R is not empty
    choose i ∈ R such that start time of i is earliest
    if i can be scheduled in some classroom k
        schedule lecture i in classroom k
    else
        allocate a new classroom d+1 and schedule lecture i in d+1
        d = d+1
```

What order should we process lectures in? According to start times (breaking ties arbitrarily)

Depth of Lectures

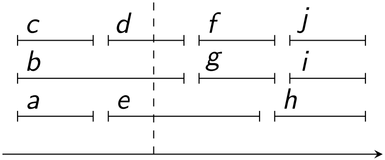
- For a set of lectures R, k are said to be in conflict if there is some time t such that there are k lectures going on at time t.

Depth of Lectures

- For a set of lectures R, k are said to be in conflict if there is some time t such that there are k lectures going on at time t.
- The depth of a set of lectures R is the maximum number of lectures in conflict at any time.

Depth of Lectures

- For a set of lectures R, k are said to be in conflict if there is some time t such that there are k lectures going on at time t.
- The depth of a set of lectures R is the maximum number of lectures in conflict at any time.



Depth and Number of Classrooms

For any set R of lectures, the number of classrooms required is at least the depth of R.

Depth and Number of Classrooms

For any set R of lectures, the number of classrooms required is at least the depth of R.

All lectures that are in conflict must be scheduled in different rooms.

Number of Classrooms used by Greedy Algorithm

Let *dep* be the depth of the set of lectures R. The number of classrooms used by the greedy algorithm is at most *dep*.

- Suppose the greedy algorithm uses more that *dep* rooms. Let j be the first lecture that is scheduled in room $dep + 1$.

Number of Classrooms used by Greedy Algorithm

Let *dep* be the depth of the set of lectures R. The number of classrooms used by the greedy algorithm is at most *dep*.

- Suppose the greedy algorithm uses more that *dep* rooms. Let j be the first lecture that is scheduled in room $dep + 1$.
- Since we process lectures according to start times, there are *dep* lectures that start (at or) before j and which are in conflict with j.

Number of Classrooms used by Greedy Algorithm

Let *dep* be the depth of the set of lectures R. The number of classrooms used by the greedy algorithm is at most *dep*.

- Suppose the greedy algorithm uses more that *dep* rooms. Let j be the first lecture that is scheduled in room $dep + 1$.
- Since we process lectures according to start times, there are *dep* lectures that start (at or) before j and which are in conflict with j.
- Thus, at the starting time for j there are at least $dep + 1$ lectures going on, which contradicts the fact that the depth is *dep*.

Correctness

The greedy algorithm does not schedule two overlapping lectures in the same room.
The greedy algorithm is correct and uses the optimal number of classrooms.

Implementation and Running Time

```
Initially R is the set of all activities
d = 0 (* number of classrooms *)
while R is not empty
    choose i ∈ R such that start time of i is earliest
    if i can be scheduled in some classroom k
        schedule lecture i in classroom k
    else
        allocate a new classroom d+1 and schedule lecture i in d+1
        d = d+1
```

Implementation and Running Time

```
Initially R is the set of all activities
d = 0 (* number of classrooms *)
while R is not empty
    choose i ∈ R such that start time of i is earliest
    if i can be scheduled in some classroom k
        schedule lecture i in classroom k
    else
        allocate a new classroom d+1 and schedule lecture i in d+1
        d = d+1
```

- Pre-sort according to start times. Picking lecture with earliest start time can be done in $O(1)$ time.

Implementation and Running Time

```
Initially R is the set of all activities
d = 0 (* number of classrooms *)
while R is not empty
    choose i ∈ R such that start time of i is earliest
    if i can be scheduled in some classroom k
        schedule lecture i in classroom k
    else
        allocate a new classroom d+1 and schedule lecture i in d+1
        d = d+1
```

- Pre-sort according to start times. Picking lecture with earliest start time can be done in $O(1)$ time.

Implementation and Running Time

```
Initially R is the set of all activities
d = 0 (* number of classrooms *)
while R is not empty
    choose i ∈ R such that start time of i is earliest
    if i can be scheduled in some classroom k
        schedule lecture i in classroom k
    else
        allocate a new classroom d+1 and schedule lecture i in d+1
        d = d+1
```

- Pre-sort according to start times. Picking lecture with earliest start time can be done in $O(1)$ time.
- Keep track of the finish time of last lecture in each room.
- Checking conflict takes $O(d)$ time.

Implementation and Running Time

```
Initially R is the set of all activities
d = 0 (* number of classrooms *)
while R is not empty
    choose i ∈ R such that start time of i is earliest
    if i can be scheduled in some classroom k
        schedule lecture i in classroom k
    else
        allocate a new classroom d+1 and schedule lecture i in d+1
        d = d+1
```

- Pre-sort according to start times. Picking lecture with earliest start time can be done in $O(1)$ time.
- Keep track of the finish time of last lecture in each room.
- Checking conflict takes $O(d)$ time.
- Total time $= O(n \log n + n\,dep)$

Recall min-priority queues

- Like stacks, queues and lists, maintains a dynamic set
- Each element has a *key*. May have other data

Operations

$\textsc{Insert}(A, x)$:	inserts element x into priority queue A
$\textsc{Minimum}(A)$:	returns element of A with smallest key
$\textsc{Extract-Min}(A)$:	removes and returns element of A with smallest key
$\textsc{Decrease-Key}(A, x, k)$:	decreases value of element x's key to k

Recall min-priority queues

- Like stacks, queues and lists, maintains a dynamic set
- Each element has a *key*. May have other data

Operations

$\textsc{Insert}(A, x)$:	inserts element x into priority queue A
$\textsc{Minimum}(A)$:	returns element of A with smallest key
$\textsc{Extract-Min}(A)$:	removes and returns element of A with smallest key
$\textsc{Decrease-Key}(A, x, k)$:	decreases value of element x's key to k

Can be implemented using heaps. All operations except MINIMUM takes $O(\log n)$ time. MINIMUM takes $O(1)$ time

Implementation and Running Time

```
Initially R is the set of all activities
d = 0 (* number of classrooms *)
while R is not empty
    choose i ∈ R such that start time of i is earliest
    if i can be scheduled in some classroom k
        schedule lecture i in classroom k
    else
        allocate a new classroom d+1 and schedule lecture i in d+1
        d = d+1
```

- Pre-sort according to start times. Picking lecture with earliest start time can be done in $O(1)$ time.
- Keep track of the finish time of last lecture in each room.
- Checking conflict takes $O(d)$ time.
- Total time $= O(n \log n + n\ dep)$

Implementation and Running Time

```
Initially R is the set of all activities
d = 0 (* number of classrooms *)
while R is not empty
    choose i ∈ R such that start time of i is earliest
    if i can be scheduled in some classroom k
        schedule lecture i in classroom k
    else
        allocate a new classroom d+1 and schedule lecture i in d+1
        d = d+1
```

- Pre-sort according to start times. Picking lecture with earliest start time can be done in $O(1)$ time.
- Keep track of the finish time of last lecture in each room.
- With priority queues, checking conflict takes $O(\log d)$ time.
- Total time $= O(n \log n + n \log dep) = O(n \log n)$

$0 - 1$ Knapsack Problem

Input: n items. Item i is worth \$ v_i and weighs w_i pounds

Output: Find a most valuable subset of items with total weight W. Have to either take an item or not take it–cannot take part of it.

Fractional Knapsack Problem

Input: n items. Item i is worth \$ v_i and weighs w_i pounds

Output: Find a most valuable subset of items with total weight W.
Can take part of an item.

Greedy algorithm and knapsack problem

Greedy algorithm and knapsack problem

Fractional knapsack has greedy solution!

Greedy algorithm and knapsack problem

Fractional knapsack has greedy solution!

- Rank items by value/weight: $\frac{v_i}{w_i}$

Greedy algorithm and knapsack problem

Fractional knapsack has greedy solution!

- Rank items by value/weight: $\frac{v_i}{w_i}$
- Take items in decreasing order of value/weight

Greedy algorithm and knapsack problem

Fractional knapsack has greedy solution!

- Rank items by value/weight: $\frac{v_i}{w_i}$
- Take items in decreasing order of value/weight
- Take all of the items with the greatest value/weight as possible, and possibly a fraction of the next item

Greedy algorithm and knapsack problem

Greedy algorithm does not work for $0 - 1$ knapsack problem!

Example:

Total capacity of knapsack, $W = 50$

i	1	2	3
v_i	60	100	120
w_i	10	20	30
$\frac{v_i}{w_i}$	6	5	4

Greedy algorithm and knapsack problem

Greedy algorithm does not work for $0-1$ knapsack problem!

Example:

Total capacity of knapsack, $W = 50$

i	1	2	3
v_i	60	100	120
w_i	10	20	30
$\frac{v_i}{w_i}$	6	5	4

Greedy solution:

- Take items 1 and 2.
- Total value $= 160$ and *weight* $= 30$.

Greedy algorithm and knapsack problem

Greedy algorithm does not work for $0-1$ knapsack problem!

Example:

Total capacity of knapsack, $W = 50$

i	1	2	3
v_i	60	100	120
w_i	10	20	30
$\frac{v_i}{w_i}$	6	5	4

Greedy solution:

- Take items 1 and 2.
- Total value $= 160$ and $weight = 30$.

Have 20 pounds of capacity left over!

Greedy algorithm and knapsack problem

Greedy algorithm does not work for $0-1$ knapsack problem!

Example:

Total capacity of knapsack, $W = 50$

i	1	2	3
v_i	60	100	120
w_i	10	20	30
$\frac{v_i}{w_i}$	6	5	4

Greedy solution:

- Take items 1 and 2.
- Total value $= 160$ and $weight = 30$.

Have 20 pounds of capacity left over!

Optimal solution:

- Take items 2 and 3
- Total value $= 220$ and $weight = 50$

Greedy Analysis: Overview

- Greedy-choice property. Argue that after each step the solution of the greedy algorithm is at least as good as the solution of any other algorithm. Example, activity selection.

Greedy Analysis: Overview

- Greedy-choice property. Argue that after each step the solution of the greedy algorithm is at least as good as the solution of any other algorithm. Example, activity selection.
- Structural property of solution. Observe some structural bound of every solution to the problem, and show that greedy algorithm achieves this bound. Example, scheduling all lectures.

This lecture was based on Sections 16.1 and 16.2 of the book
Your midterm is based on material upto now