Introduction to Algorithms

Greedy Algorithms

My T. Thai @ UF

Overview

- A *greedy algorithm* always makes the choice that looks best at the moment
  - Make a locally optimal choice in hope of getting a globally optimal solution
  - Example: Play cards, Invest on stocks, etc.
- Do not always yield optimal solutions
- They do for some problems with optimal substructure (like Dynamic Programming)
- Easier to code than DP

My T. Thai
mythai@cise.ufl.edu

An Activity-Selection Problem

- Input: Set S of n activities, $a_1, a_2, \ldots, a_n$.
  - Activity a_i starts at time s_i and finishes at time f_i
  - Two activities are compatible, if their intervals don't overlap.
- Output: A subset of maximum number of mutually compatible activities.

My T. Thai
mythai@cise.ufl.edu

An Activity-Selection Problem

- Assume activities are sorted by finishing times.

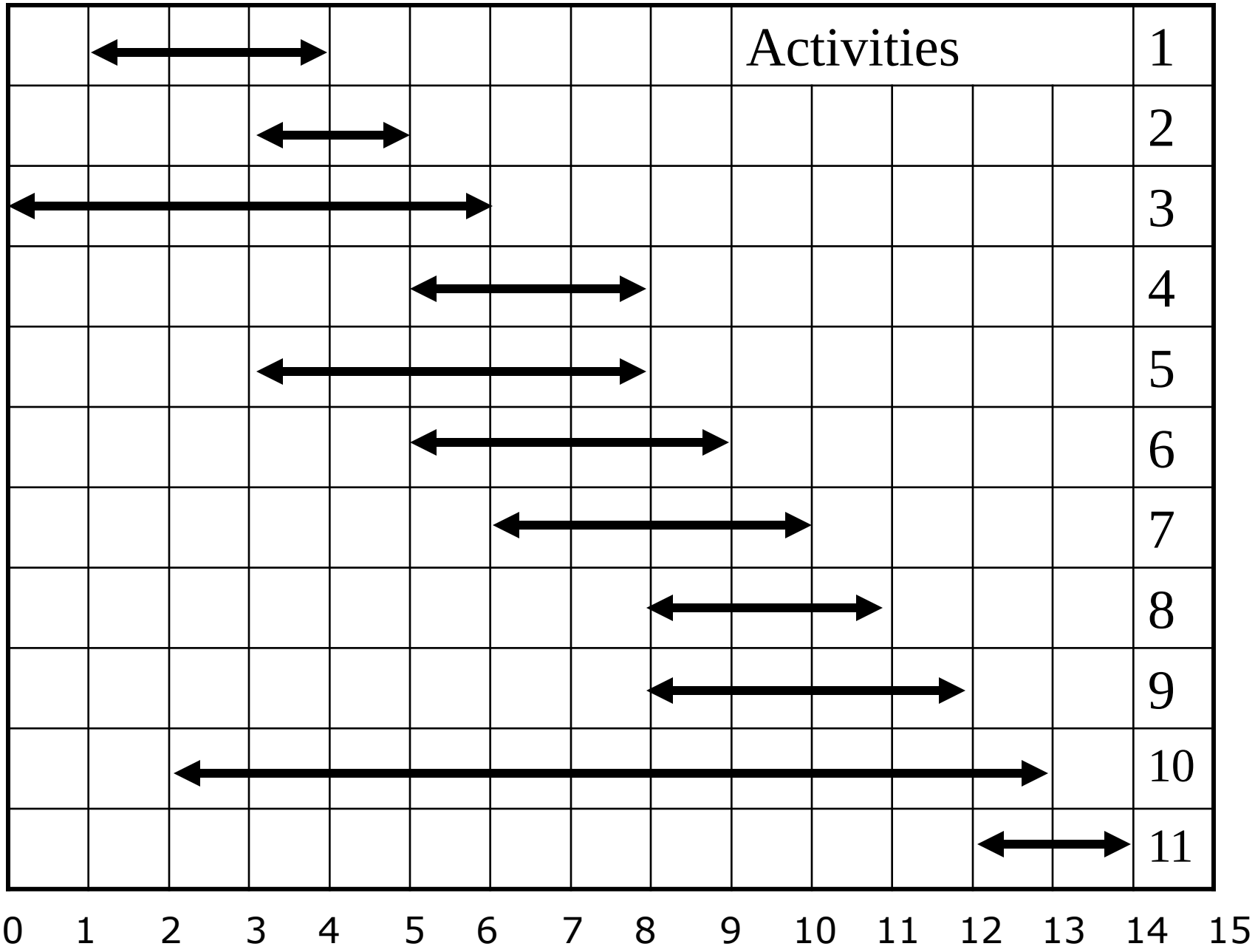
$$f_1 \leq f_2 \leq \ldots \leq f_n.$$

- Example:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

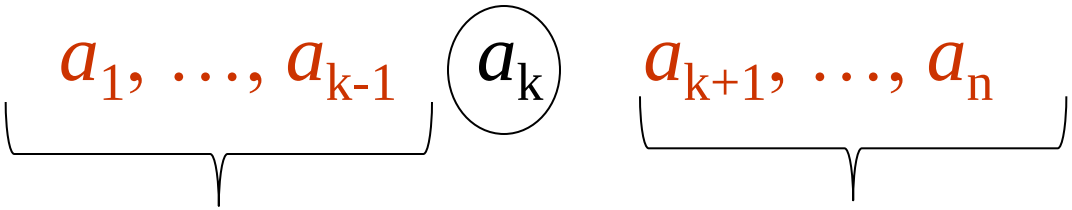
- Possible sets of mutually compatible activies:
  - $\{a_3, a_9, a_{11}\}$
  - $\{a_1, a_4, a_8, a_{11}\}$ } Largest subsets
  - $\{a_2, a_4, a_9, a_{11}\}$ } Largest subsets

My T. Thai
mythai@cise.ufl.edu


Activities
1
2
3
4
5
6
7
8
9
10
11
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Optimal Substructure

- Suppose an optimal solution includes activity a_k. Two subproblems:



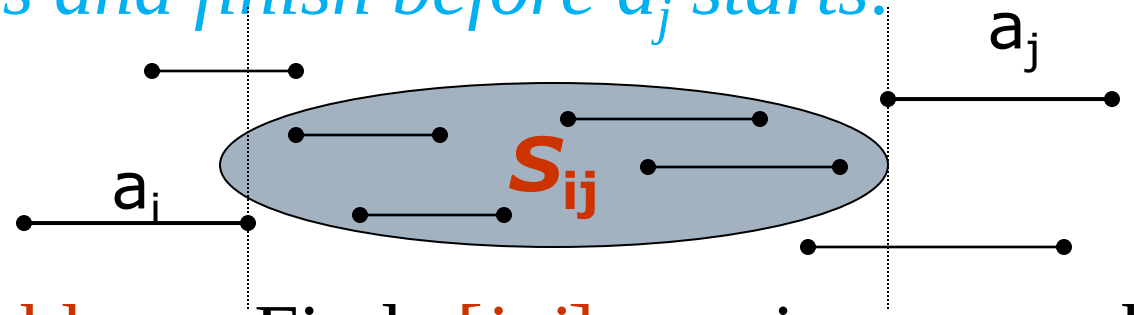
1. Select compatible activities that finish before a_k starts

2. Select compatible activities that finish after a_k finishes

  - The solutions to the two subproblems must be optimal (prove using cut-and-paste argument)

Recursive Solution

- S_{ij} : Subset of activities that start *after* a_i *finishes and finish before* a_j *starts*.



- Subproblems: Find $c[i, j]$, maximum number of mutually compatible activities from S_{ij}.
- Recurrence:

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset \end{cases}$$

My T. Thai
mythai@cise.ufl.edu

Activity-selection: Dynamic Programming

- We can use Dynamic Programming
  - Memoize OR
  - Bottom-up filling the table entries

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset \end{cases}$$

- But, simpler approach exists

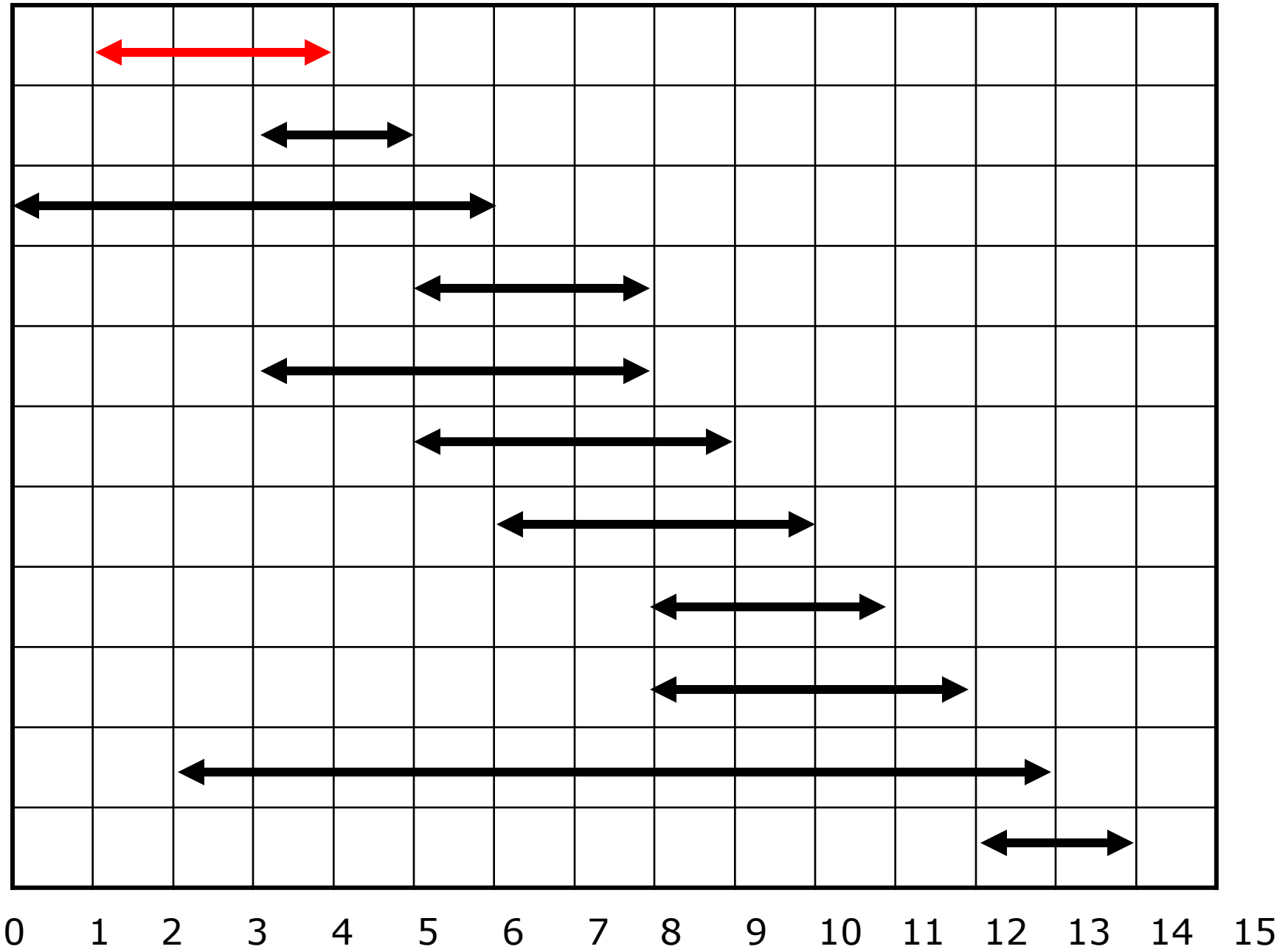
My T. Thai
mythai@cise.ufl.edu

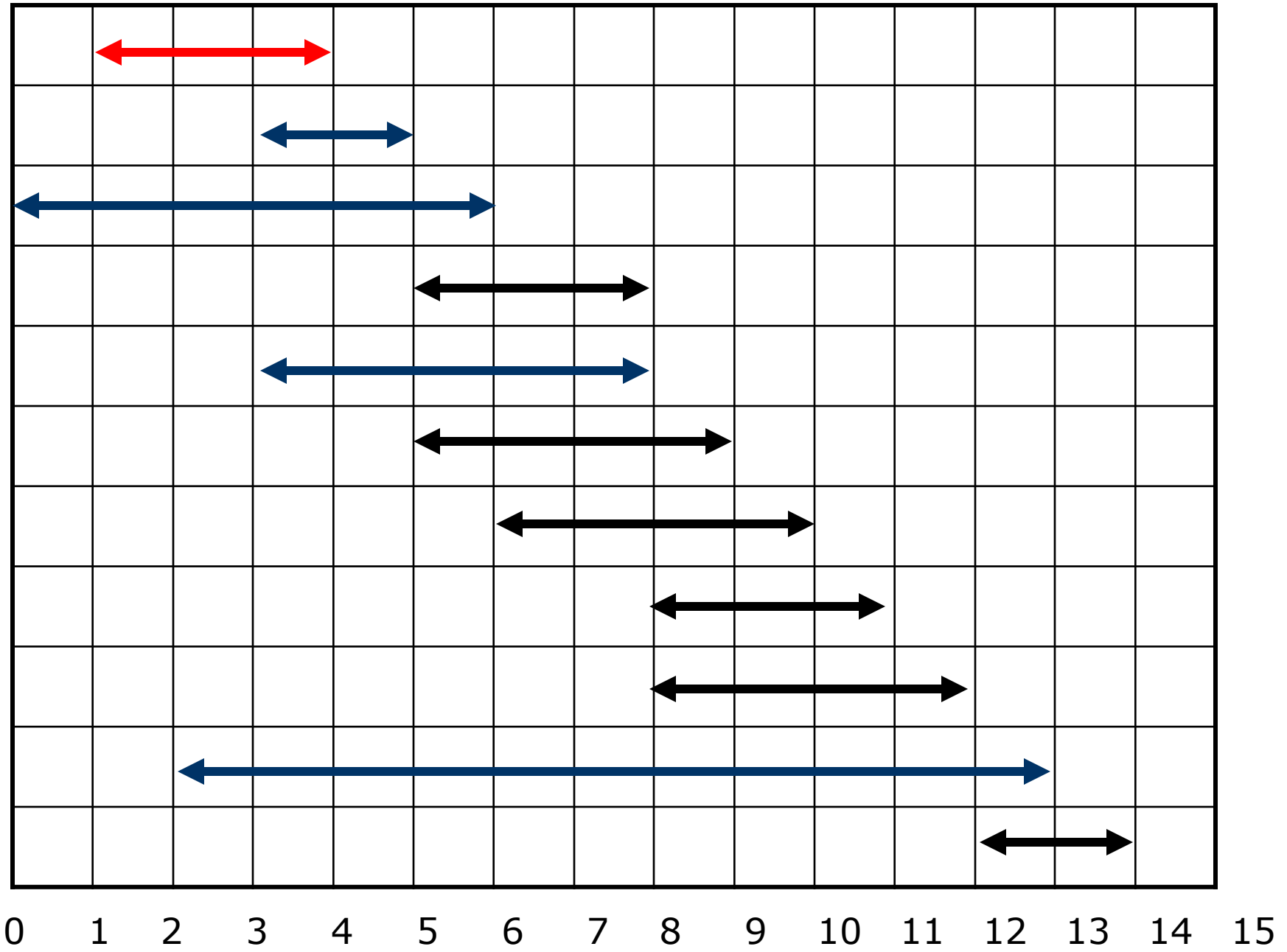
Early Finish Greedy

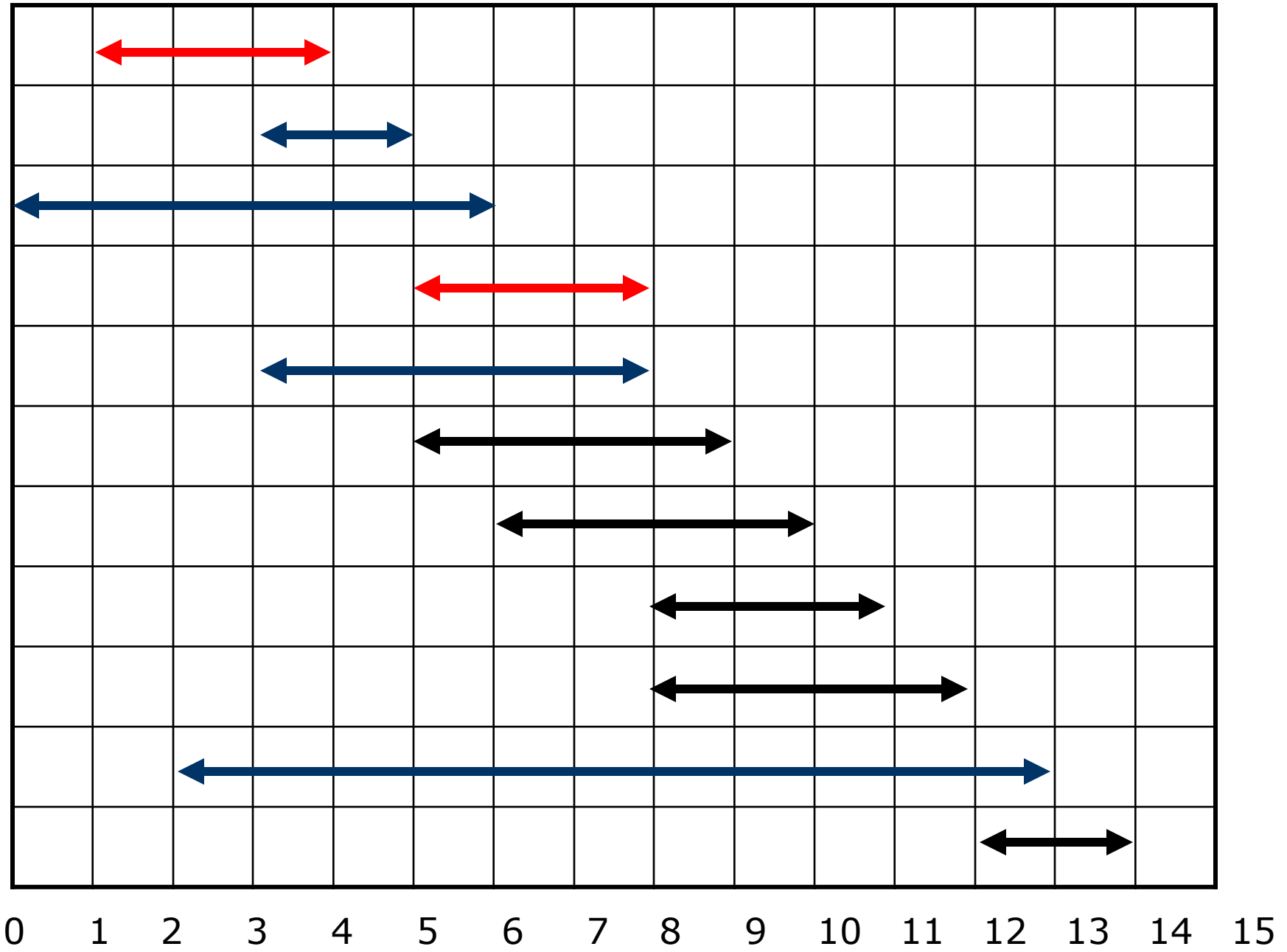
1. **while** (∃ activities)
2. Select the activity with the earliest finish
3. Remove the activities that are not compatible
4. **end while**

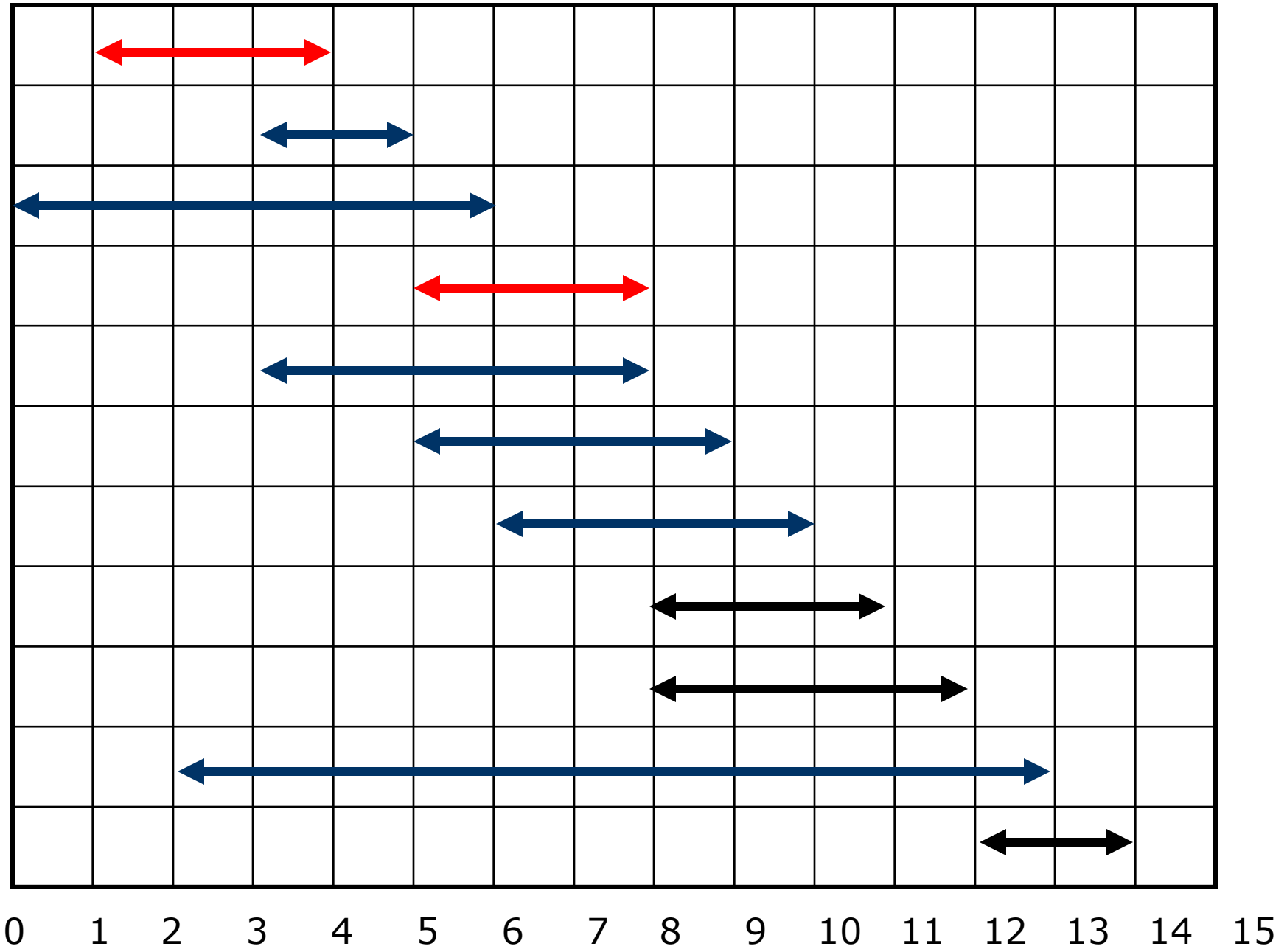
- Greedy in the sense that:
  - It leaves as much opportunity as possible for the remaining activities to be scheduled.
  - Maximizes the amount of unscheduled time remaining

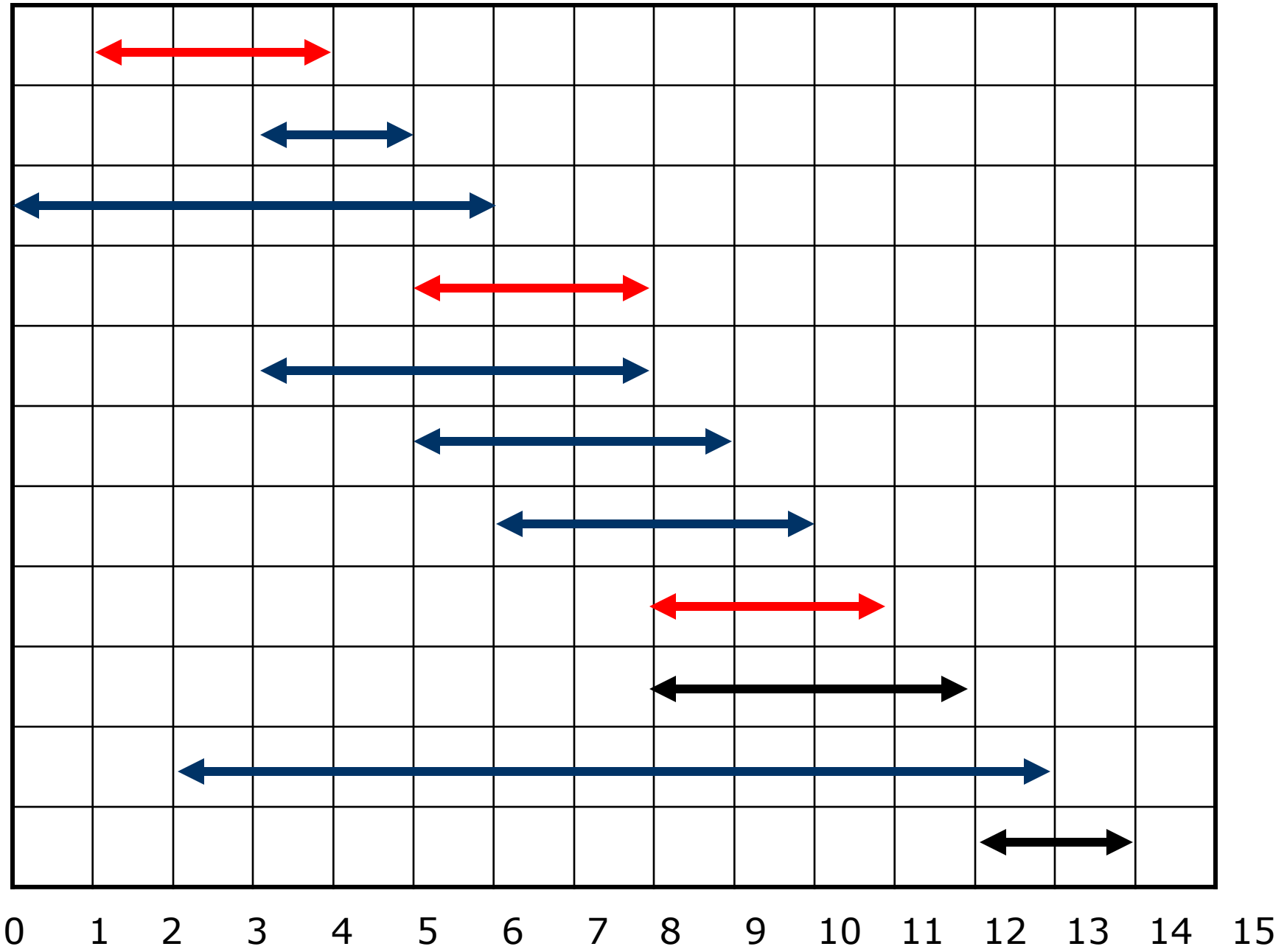
My T. Thai
mythai@cise.ufl.edu


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Greedy Choice Property

- Locally optimal choice $\Rightarrow$ globally optimal solution
  - Them 16.1: if S is an non-empty activity-selection subproblem, then $\exists$ optimal solution $A \subseteq S$ such that $a_{s1} \in A$, where a_{s1} is the earliest finish activity in A
  - Sketch of proof: if $\exists$ optimal solution B that does not contain a_{s1}, can always replace the first activity in B with a_{s1}. Same number of activities, thus optimal.

My T. Thai
mythai@cise.ufl.edu

Recursive Algorithm

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
1  m = k + 1
2  while m ≤ n and s[m] < f[k]        // find the first activity in S_k to finish
3      m = m + 1
4  if m ≤ n
5      return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6  else return ∅
```

- Initial call: RECURSIVE-ACTIVITY-SELECTOR$(s, f, 0, n)$.
- Complexity: $\Theta(n)$
- Straightforward to convert to an iterative one

My T. Thai
mythai@cise.ufl.edu

Iterative Algorithm

```
GREEDY-ACTIVITY-SELECTOR(s, f)
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5      if s[m] ≥ f[k]
6          A = A ∪ {a_m}
7          k = m
8  return A
```

- Initial call: GREEDY-ACTIVITY-SELECTOR(s, f)
- Complexity: $\Theta(n)$

My T. Thai
mythai@cise.ufl.edu

Elements of the greedy strategy

1. Determine the optimal substructure
2. Develop the recursive solution
3. Prove that if we make the greedy choice, only one subproblem remains
4. Prove that it is safe to make the greedy choice
5. Develop a recursive algorithm that implements the greedy strategy
6. Convert the recursive algorithm to an iterative one.

My T. Thai
mythai@cise.ufl.edu

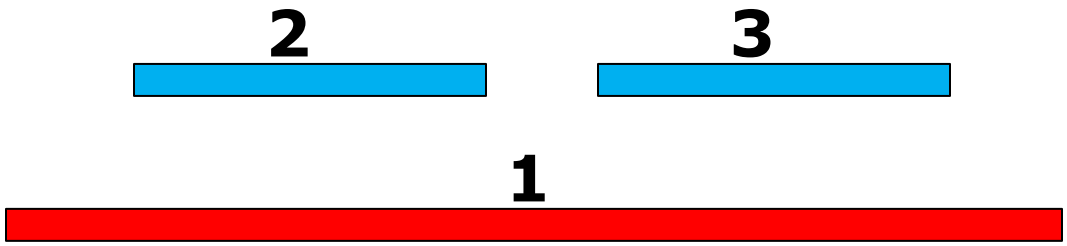
Not All Greedy are Equal

- Earliest finish time: Select the activity with the earliest finish time ← Optimal
- Earliest start time: Select activity with the earliest start time
- Shortest duration: Select activity with the shortest duration $d_i = f_i - s_i$
- Fewest conflicts: Select activity that conflicts with the least number of other activities first
- Last start time: Select activity with the last start time

My T. Thai
mythai@cise.ufl.edu

Not All Greedy are Equal

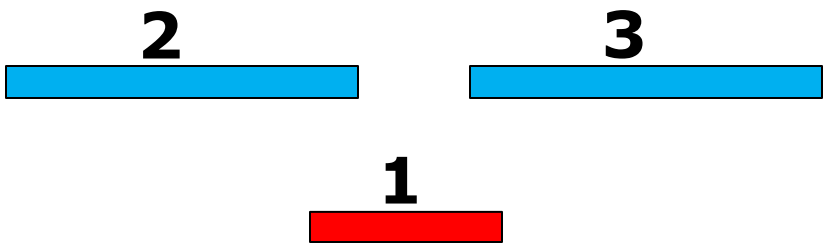
- Earliest start time: Select activity with the earliest start time



My T. Thai
mythai@cise.ufl.edu

Not All Greedy are Equal

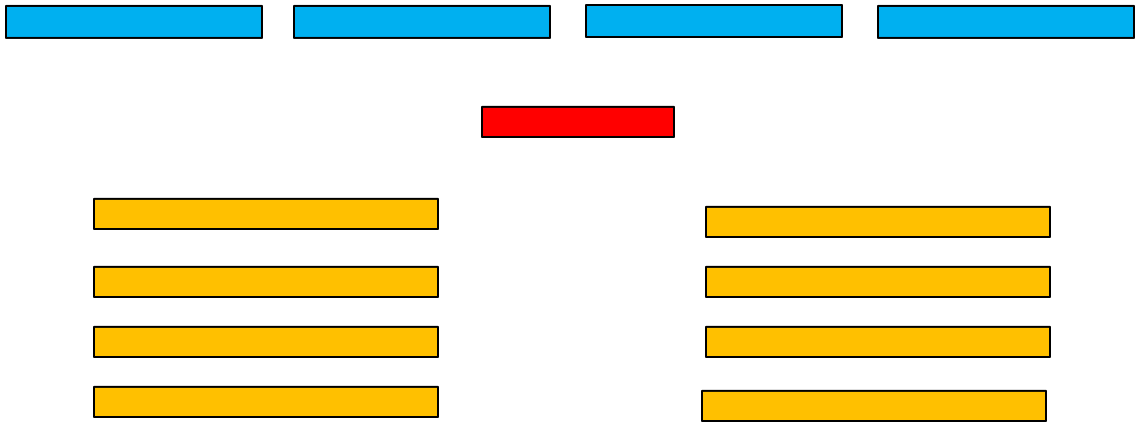
- Shortest duration: Select activity with the shortest duration $d_i = f_i - s_i$



My T. Thai
mythai@cise.ufl.edu

Not All Greedy are Equal

- Fewest conflicts: Select activity that conflicts with the least number of other activities first



My T. Thai
mythai@cise.ufl.edu

Not All Greedy are Equal

- Last start time: Select activity with the last start time

Greedy vs. Dynamic Programming

- Knapsack Problem: A thief robbing a store and find *n* items:
  - The *i*th item is worth v_i dollars and weighs w_i pounds
  - The thief can carry at most *W* pounds
  - v_i , w_i, *W* are integers
  - Which items should he take to obtain the maximum amount of money?????
- 0-1 knapsack → each item is taken or not taken
- Fractional knapsack → fractions of items can be taken

My T. Thai
mythai@cise.ufl.edu

Knapsack Problem

- Both exhibit the optimal-substructure property
  - 0-1: If item *j* is removed from an optimal packing, the remaining packing is an optimal packing with weight at most W-w_j
  - Fractional: If *w pounds* of item *j* is removed from an optimal packing, the remaining packing is an optimal packing with weight at most W-w that can be taken from other n-1 items plus $w_j - w$ of item *j*

My T. Thai
mythai@cise.ufl.edu

Fractional Knapsack Problem

- Can be solvable by the greedy strategy
  - Compute the value per pound v_i/w_i for each item
  - Obeying a greedy strategy, take as much as possible of the item with the greatest value per pound.
  - If the supply of that item is exhausted and there is still more room, take as much as possible of the item with the next value per pound, and so forth until there is no more room
  - O(n lg n) (we need to sort the items by value per pound)
  - Greedy Algorithm?
  - Correctness?

My T. Thai
mythai@cise.ufl.edu

O-1 knapsack

- Much harder!
- Cannot be solved by the greedy strategy. Counter example?
- We must compare the solution to the sub-problem in which the item is included with the solution to the sub-problem in which the item is excluded before we can make the choice
  - Dynamic Programming

My T. Thai
mythai@cise.ufl.edu

Counter Example

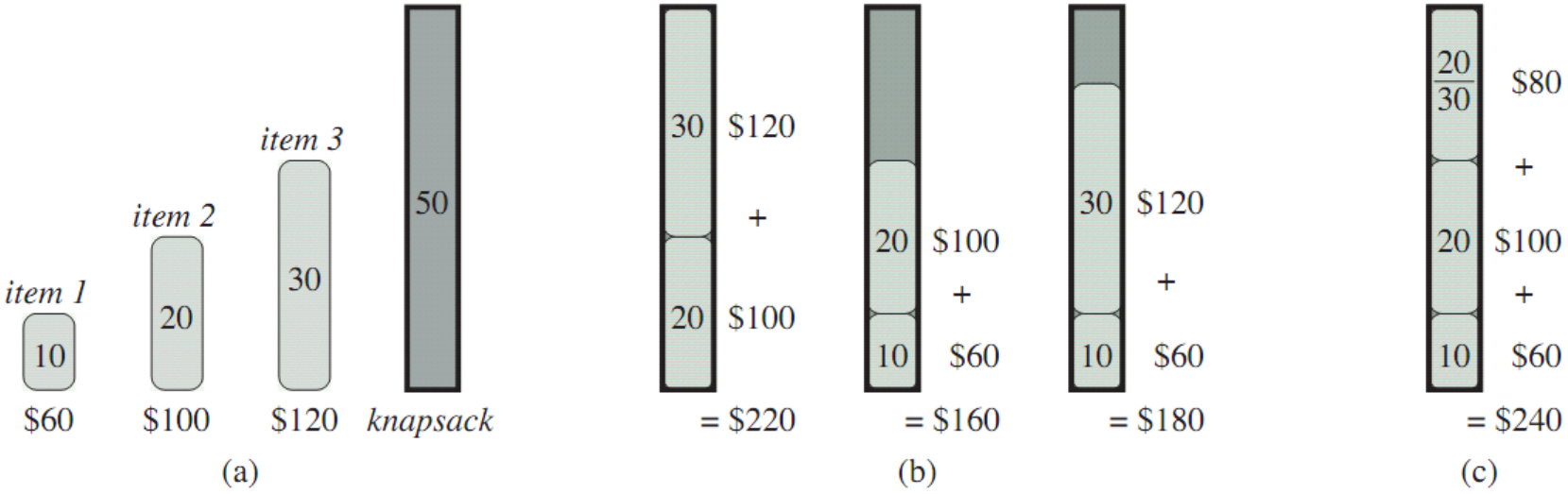


Figure 16.2 An example showing that the greedy strategy does not work for the 0-1 knapsack problem. **(a)** The thief must select a subset of the three items shown whose weight must not exceed 50 pounds. **(b)** The optimal subset includes items 2 and 3. Any solution with item 1 is suboptimal, even though item 1 has the greatest value per pound. **(c)** For the fractional knapsack problem, taking the items in order of greatest value per pound yields an optimal solution.

Matroid

- Hassler Whitney coined the term "**matroid**" while studying rows of a given matrix and sets of linearly **independent** row vectors.

A matroid $M = (S, I)$ satisfies the following conditions

1. A finite ground set S.
2. A nonempty family I of subset of S, called the independent subsets of S such that
   $B \in I, A \subseteq B \Rightarrow A \in I$ (hereditary property.)
3. $A \in I$, $B \in I$, and $|A| < |B| \Rightarrow \exists x \in B - A$ such that $A \cup \{x\} \in I$ (exchange property).

My T. Thai
mythai@cise.ufl.edu

Examples – Matrix Matroid

- Ground set *S*: The set of rows (or columns) of a matrix
- Independent sets *I*: The sets of *linearly independent* rows (columns)

$$A = \begin{vmatrix} 3 & 1 & 3 & 0 & 2 & 1 \\ 0 & 2 & 1 & 1 & 0 & 0 \\ 1 & 1 & 2 & 0 & 1 & 0 \\ 2 & 0 & 0 & -1 & 0 & 2 \end{vmatrix}$$

$$e_1 \quad e_2 \quad e_3 \quad e_4 \quad e_5 \quad e_6$$

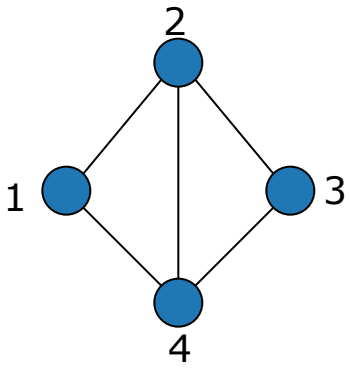
$I_1 = \{e_1, e_2, e_3, e_4\}$ are linearly independent $\Rightarrow I_1$ is a independent set.

$I_2 = \{e_1, e_4, e_5, e_6\}$ are not linearly independent $\Rightarrow I_2$ is a dependent set

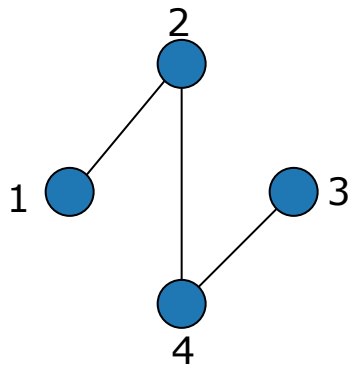
UNIVERSITY OF FLORIDA

Examples- Graphic Matroid

- Ground set *S*: The set of edges of graph
- Independent sets *I*: The sets of edges that **do not form a cycle**.

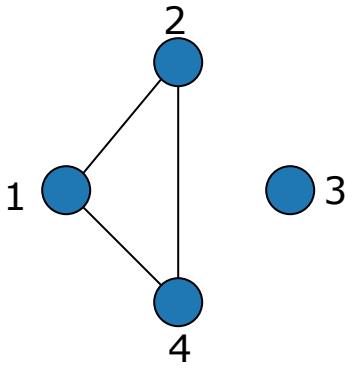


$G = (V, E)$



$E_1 = \{(1;2), (2;4), (3;4)\}$

Independent set



$E_2 = \{(1;2), (2;4), (3;4)\}$

Dependent set

Partition matroid

- Ground set is the union of m finite disjoint sets E_i, for $1 \leq i \leq r$
- Independent sets are sets formed by taking at most one element from each set E_i

Matroid Properties

- **Extension**: For $A \in I$, If $x \notin A$ and $A \cup \{x\} \in I$, then x is an **extension** of A.
- **Maximal**: $A \in I$ and A has no extension, then A is maximal.

All maximal independent subsets in a matroid have the same size.

- Proof. Suppose A, B are maximal independent sets and $|A|<|B|$. **Exchange property** implies $\exists x \in B - A$ such that $A \cup \{x\} \in I$, contradicting the assumption that A is maximal.

My T. Thai
mythai@cise.ufl.edu

Weighted Matroid

- Assign weight $w(x) > 0$ for all $x \in S$.
- Weight of an independent set $A \in I$:

  $w(A) = \sum_{x \in A} w(x)$,
- **Problem**: Find an independent set $A \in I$ that has maximum possible weight.
  - The optimal independent set is always **maximal**. Why?

My T. Thai
mythai@cise.ufl.edu

Minimum Spanning Tree

- Connected undirected graph $G=(V, E)$, each edge $e \in E$ has length $w(e)$.
- **Minimum-spanning-tree** problem: Find a subsets of edges that connects all of the vertices and has minimum total length.
- How to relate to weighted matroid?

Minimize length

vs.

Maximize weight independent set

My T. Thai
mythai@cise.ufl.edu

Minimum Spanning Tree

- Recall $M_G = (S_G, I_G)$
  - S_G : Set of edges in G (E)
  - I_G : Set of acyclic subsets of edges.
- Weight function: $w'(e) = w_0 - w(e)$, where w_0 is larger than the maximum length of any edge.
- A maximal independent set $\leftrightarrow$ A spanning tree of $|V| - 1$ edges.

My T. Thai
mythai@cise.ufl.edu

Minimum Spanning Tree

- For a maximal independent set A of $|V|$ - *1* elements:

$$
\begin{aligned}
w'(A) &= \sum_{e \in A} w'(e) \\
&= \sum_{e \in A} (w_0 - w(e)) \\
&= (|V| - 1) w_0 - \sum_{e \in A} w(e) \\
&= (|V| - 1) w_0 - w(A)
\end{aligned}
$$

A constant

- Maximize $w'(A)$ $\leftrightarrow$ Minimize $w(A)$

My T. Thai
mythai@cise.ufl.edu

Weighted Matroids: Greedy Algorithm

- Matroid $M = (S, I)$: Select $x \in S$, in order of monotonically decreasing weight, and adds it to the set A if $A \cup \{x\}$ is still an independent set.

```
GREEDY(M, w)
1  A = ∅
2  sort M.S into monotonically decreasing order by weight w
3  for each x ∈ M.S, taken in monotonically decreasing order by weight w(x)
4       if A ∪ {x} ∈ M.I
5            A = A ∪ {x}
6  return A
```

To check if $A \cup \{x\} \in I$

- Time complexity: $O(n \log n + n f(n))$

Sorting time

My T. Thai
mythai@cise.ufl.edu

Matroid Greedy-choice property

- Lemma 16.7: Sort S into monotonically decreasing order by w. Let x be the first element of S such that $\{x\}$ is independent. If x exists, then there exists an optimal subset A of S that contains x.
- Proof.
  - If no such x exists, $\varnothing$ is the optimal solution.
  - Assume B is the optimal set and $x \notin B$
  - Construct $A = \{x\}$, repeatedly find a new element of B/A that can be added to A (**exchange property**).

My T. Thai
mythai@cise.ufl.edu

Greedy-choice property

- Lemma 16.7: Sort S into monotonically decreasing order by w. Let x be the first element of S such that $\{x\}$ is independent. If x exists, then there exists an optimal subset A of S that contains x.
- Proof (cont.)
  - Finally, $A = B - \{y\} \cup \{x\}$ for some $y \in B$.
  - We have: $$\begin{aligned} w(A) &= w(B) - w(y) + w(x) \\ &\geq w(B) . \end{aligned}$$
  - B is optimal $\Rightarrow$ A is optimal. □

Greedy-choice property

If element x cannot be used immediately by GREEDY, it can never be used later.

- **Proof**. Assume x is an *extension* of an independent set $A \in I$ (x can be used later) i.e. $A \cup \{x\} \in I$.
- $\{x\} \subseteq A \cup \{x\} \Rightarrow \{x\} \in I$ (hereditary property) *i.e.* x is an extension of $\varnothing$ (can be used immediately)
- Hence, if $\{x\} \notin I$, x cannot be an extension of A □

My T. Thai
mythai@cise.ufl.edu

Optimal-substructure property

- Let x be the first element chosen by GREEDY
- Subproblem: Find *optimal* (maximum-weight) independent subset of a new weighted matroid $M' = (S', I')$
  - $S' = \{y \in S : \{x, y\} \in \mathcal{I}\}$,
  - $\mathcal{I}' = \{B \subseteq S - \{x\} : B \cup \{x\} \in \mathcal{I}\}$

A is an *optimal* independent set of M, iff $A' = A - \{x\}$ is an *optimal* independent set of M'.

My T. Thai
mythai@cise.ufl.edu

Correctness of GREEDY

- By Greedy-choice property:
  - Any elements that GREEDY passes over initially will never be useful.
  - If GREEDY select *x*, then there exists an optimal solution containing *x*.
- By Optimal-substructure property
  - The remaining problem after selecting x can be interpreted as finding optimal solution on the new matroid $M'=(S', I')$.
  - Subsequence operation GREEDY will also make correct (optimal) choices.

Minimum-spanning tree (again)

- Three greedy algorithms:
  - Kruskal's 1956 (Boruvka 1926)
  - Prim's 1957 (Jarnik 1930)
  - Sollin's (Baruvka 1926)
    (Parallel prim's algorithm)
- Kruskal's algorithm = GREEDY algorithm
  - Select edges in non-decreasing order of weights.
  - Add edges if they do not create cycles.
- GREEDY's correctness →Kruskal's correctness

My T. Thai
mythai@cise.ufl.edu

A task-scheduling problem

- Scheduling unit-time tasks with deadlines and penalties for a single processor :
- Given :
  - n unit-time tasks : $S = \{a_1, a_2, \ldots, a_n\}$
  - Deadlines $d_1, \ldots, d_n$ (positive integers)
  - Penalties $w_1, \ldots, w_n \geq 0$: we incur a penalty w_i if task a i is not finished by time d_i. No penalty, otherwise.
- Find a schedule that minimizes the total penalty

My T. Thai
mythai@cise.ufl.edu

A task-scheduling problem

- Example:

	Task						
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

- Set *A* of tasks is independent, if we can schedule *A* so that no task is late.
- For $t = 1, \ldots, n$

  $N_t(A)$ = number of tasks in *A* with deadline *t* or earlier

My T. Thai
mythai@cise.ufl.edu

A task-scheduling problem

- The following statements are equivalent:
  - A is independent.
  - $N_t(A) \leq t$, for all $t = 1, \ldots, n$
  - If the tasks in A are scheduled in order of non-decrease deadlines, then no task is late.
- Define $M = (S, I)$
  - S: Set of tasks.
  - I : Family of independent sets of tasks.

My T. Thai
mythai@cise.ufl.edu

A task-scheduling problem

- Theorem: $M=(S, I)$ is a matroid.
- Proof.
  - I is non-empty.
  - If A is an independent set, then clearly all subsets of A can be scheduled with no late tasks (**heredity**)
  - Assume $|A| < |B|$, and A, B are independent sets.
    - Let k be the largest t, $N_t(B) \leq N_t(A)$.
    - $N_j(B) > N_j(A)$ for all $j > k$
    - Let x be a task in B/A with deadline $k + 1$, $A' = A \cup \{x\}$
    - $N_t(A) \leq N_t(A') \leq N_t(B) \Rightarrow A' \in I$ (**exchange**)

My T. Thai
mythai@cise.ufl.edu

A task-scheduling problem

- Minimizing total penalty = maximizing weight of an independent subset
- GREEDY algorithm to find maximum weight independent set
- Complexity: $O(n^2)$ (each of $O(n)$ independence checks takes time $O(n)$)

My T. Thai
mythai@cise.ufl.edu

A task-scheduling problem

- Example

	Task						
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

- Select a_1 , a_2 , a_3 , and a_4
- Reject a_5 , a_6
- Select a_7
- Final optimal schedule: $<a_2,a_4,a_1,a_3,a_7,a_5 ,a_6>$
- Penalty incurred: $w_5 + w_6 = 50$.

My T. Thai
mythai@cise.ufl.edu