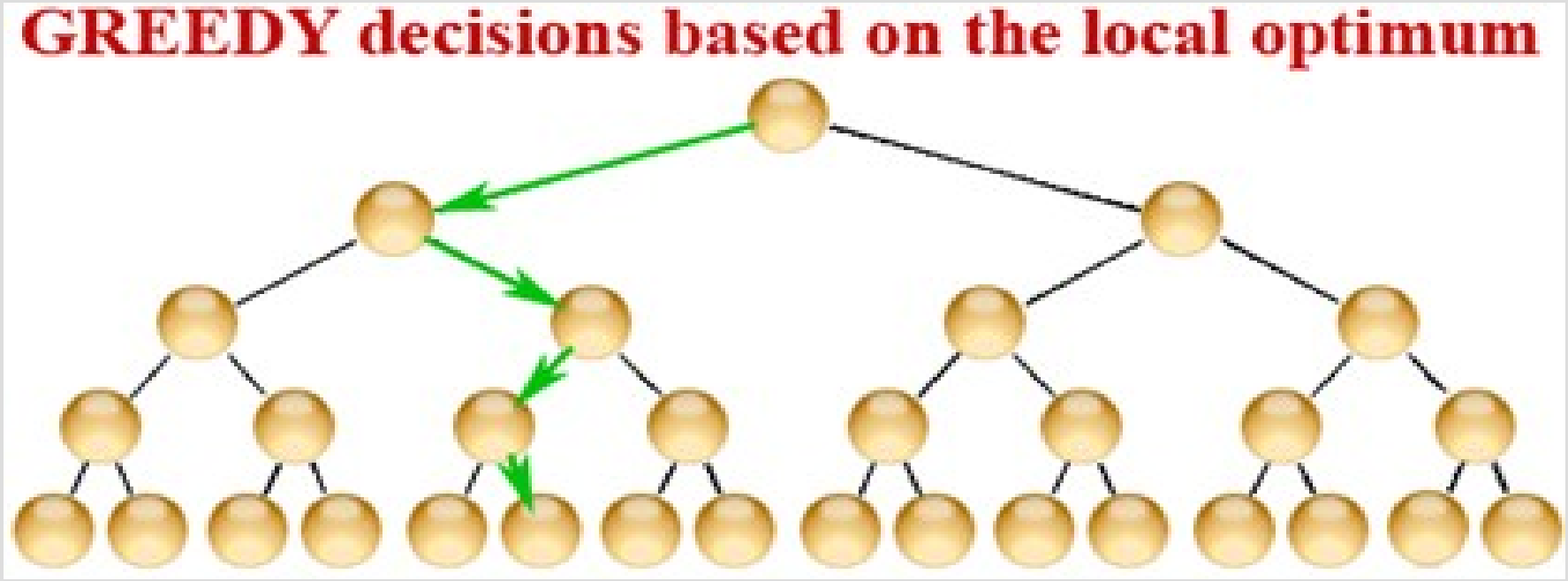
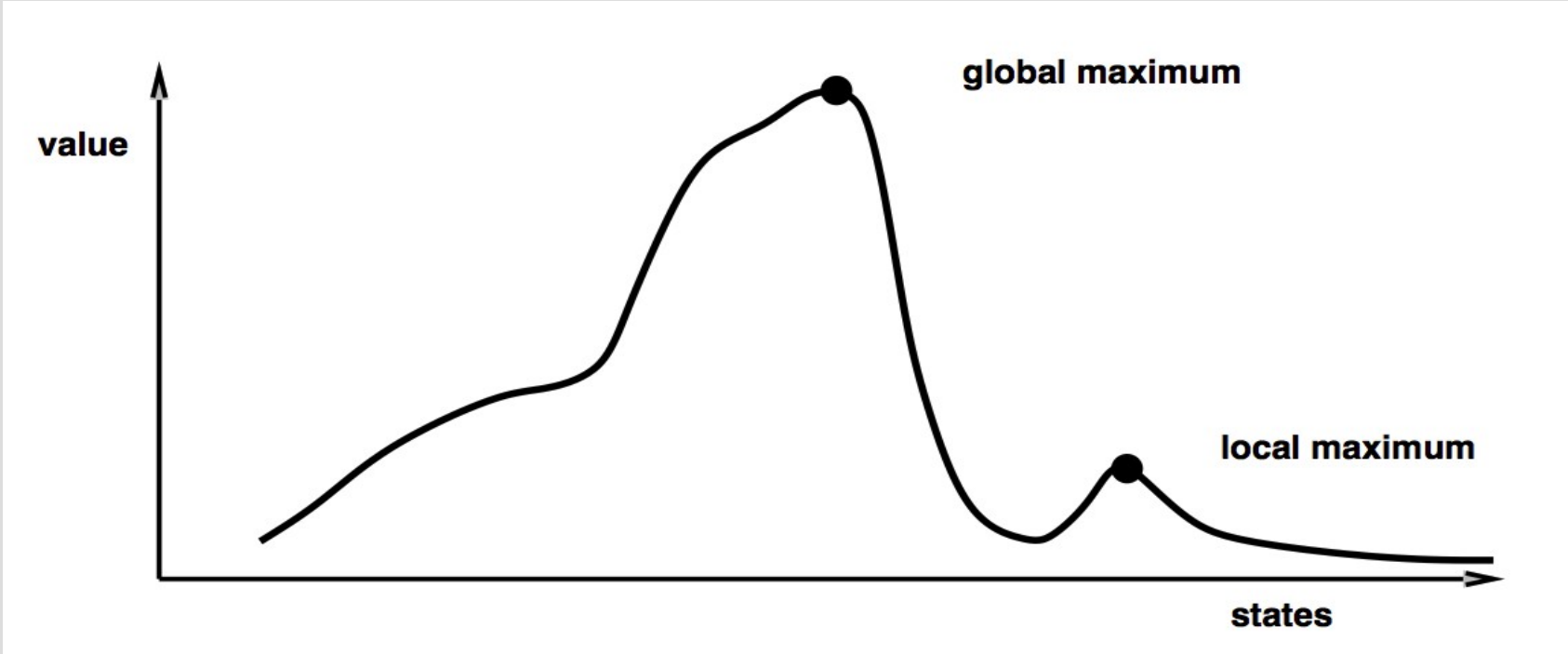
Greedy algorithms



Greedy algorithms

Find the best solution to a local problem and (hope) it solves the global problem



Greedy algorithm

Greedy algorithms find the global maximum when:

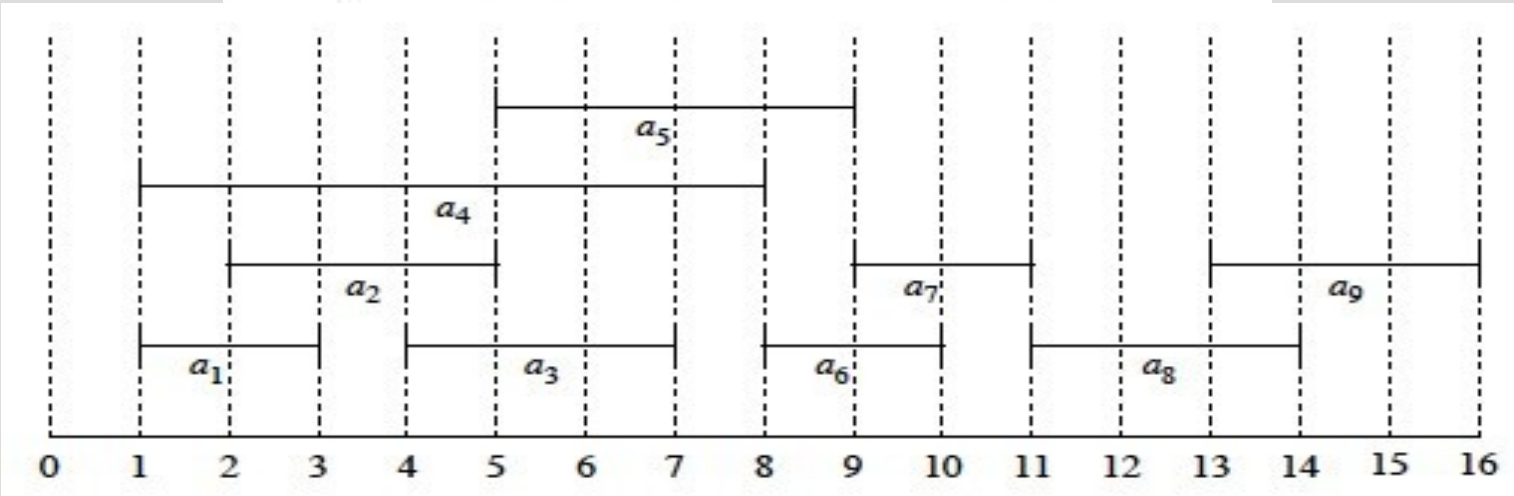
1. optimal substructure – optimal solution to a subproblem is a optimal solution to global problem
2. greedy choices are optimal solutions to subproblems

Activity selection

A list of tasks with start/finish times

Want to finish most number of tasks

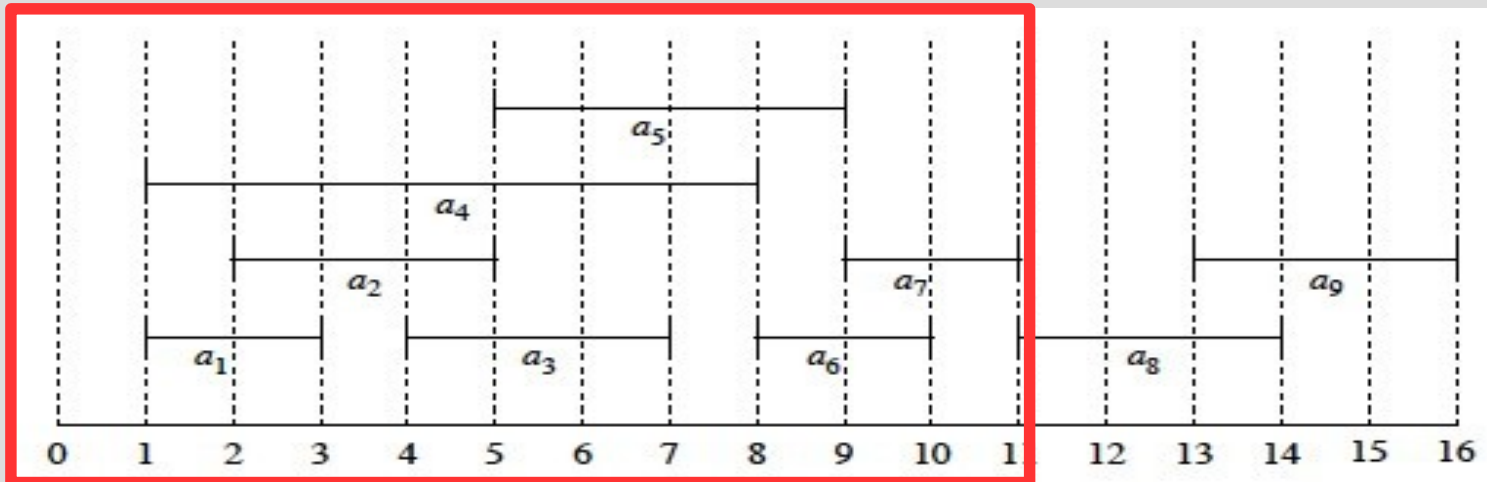
i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16



Activity selection

Optimal substructure:

Finding the largest number of tasks that finish before time t can be combined with the largest number of tasks that start after time t



Activity selection

Greedy choice:

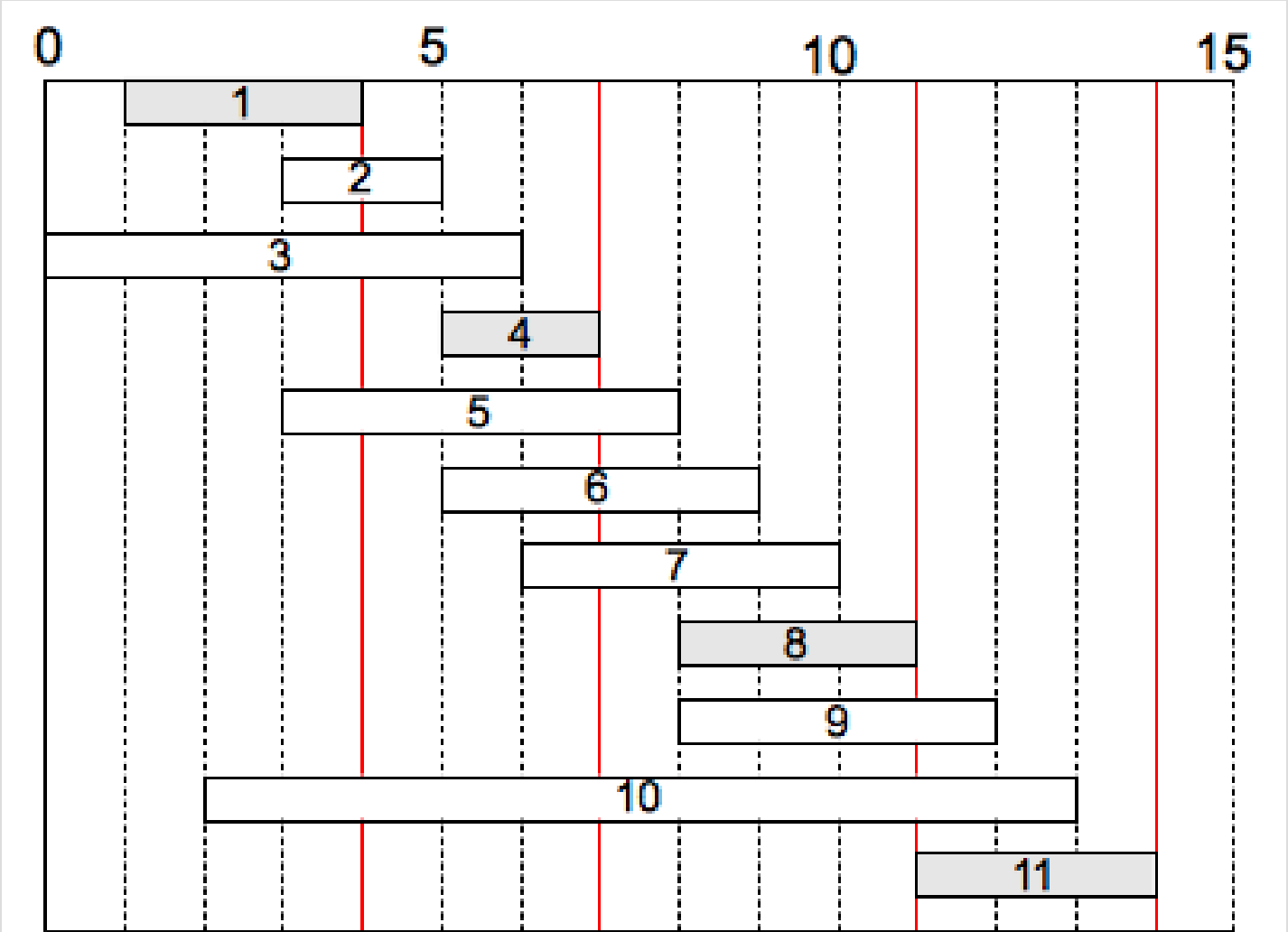
The task that finishes first is in a optimal solution

Proof:

Suppose we have optimal solution A. If quickest finishing task in A, done. Otherwise we can swap it in.

Activity selection

Greedy: select earliest finish time



Knapsack problem

A list of items with their values, but your knapsack has a weight limit

Goal: put as much value as you can in your knapsack

item	weight	value
1	3	$25
2	2	$20
3	1	$15
4	4	$40
5	5	$50

, capacity $W = 6$.

Knapsack problem

What is greedy choice?

item	weight	value
1	3	\$25
2	2	\$20
3	1	\$15
4	4	\$40
5	5	\$50

, capacity $W = 6$.

Knapsack problem

What is greedy choice?

A: pick the item with highest value to weight ratio (value/weight)
(only optimal if fractions allowed)

item	weight	value
1	3	\$25
2	2	\$20
3	1	\$15
4	4	\$40
5	5	\$50

, capacity $W = 6$.

Huffman code

Who has used a zip/7z/rar/tar.gz?

Compression looks at the specific files you want to compress and comes up with a more efficient binary representation

Huffman code

How many letters in alphabet?
How many binary digits do we need?

If we are given a specific set of letters, we can have variable length representations and save space:
aaabaaabaa : a=0,b=1->0001000100
or :aaab=1,a=0 -> 1100

Huffman code

Huffman code uses variable size letter representation compress binary representation on a specific file

letter:	a	b	c	d	e
count:	15	7	6	6	5

What is greedy choice?

Huffman code

We want longer representations for less frequently used letters

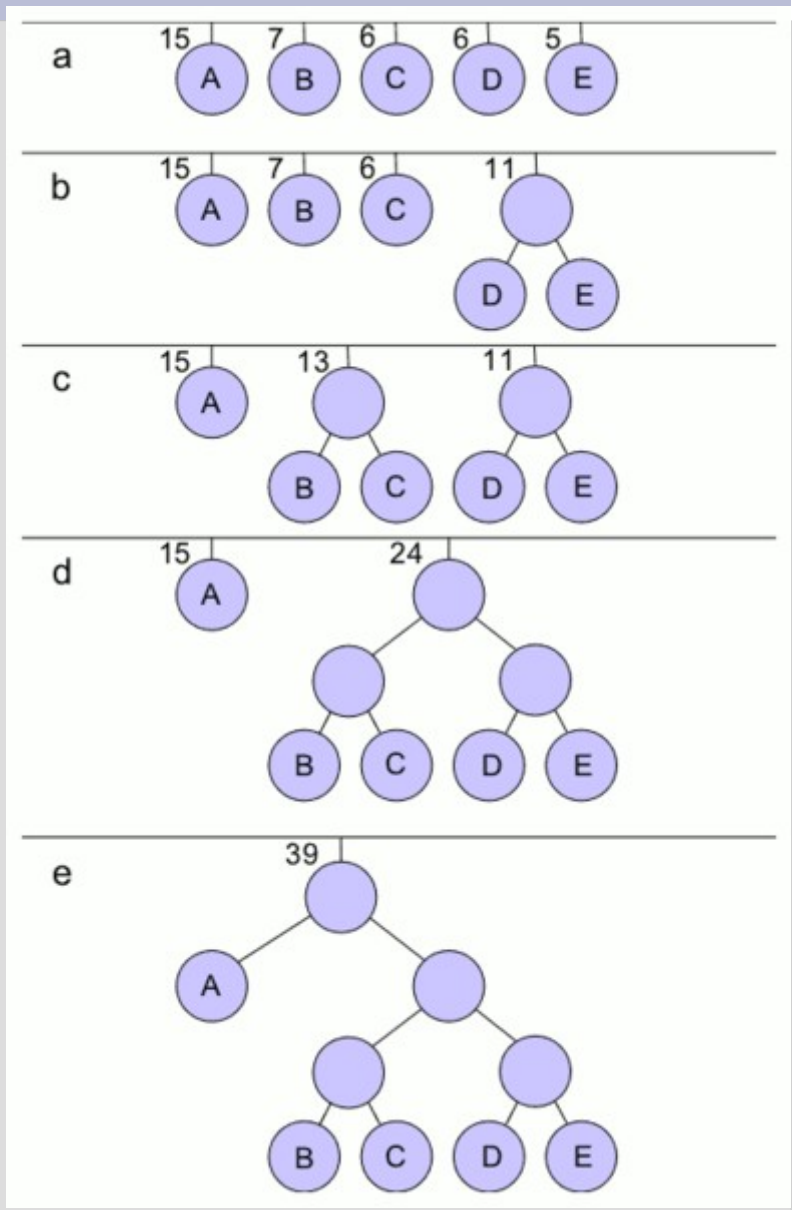
Greedy choice: Find least frequently used letters (or group of letters) and assign them an extra 1/0

Repeat until all letters unique encode

Huffman code

1. Merge least frequently used nodes into a single node (usage is sum)

2. Repeat until all nodes on a tree



Huffman code

Huffman coding length =
15 * 1 + 3 * 24 = 87

Original coding length =
15 * 3 + 3 * 24 = 117

25 percent compression

Dynamic programming

Greedy algorithms are closely related to dynamic programming

(You will learn this in CSci 5421)

Idea: “forward” solution hard, so start from end (subproblem) and recombine to get start

Dynamic programming

Shortest path from A to D?
(Can start/end on x or y)