CS146: Data Structures and Algorithms Lecture 13

GREEDY TECHNIQUE

INSTRUCTOR: KATERINA POTIKA
CS SJSU

Optimization Problems

- Problems can have many possible solutions.
- Each solution has a **value**, and we wish to find a solution with the optimal
  - **minimum** value **(minimization problem)** or
  - **maximum** value **(maximization)**.
- We call such a solution *an* optimal solution to the problem, as opposed to *the* optimal solution
  - there may be several solutions that achieve the optimal value.

Greedy Algorithms – 2nd Technique (Ch 16)

- Activity Selection (Interval scheduling)
- Interval partitioning
- Huffman Codes
- Knapsack Problem (later)
- Graph Problems (later)

Greedy algorithms I

- Greedy algorithms make decisions based on a greedy criterion.
- Initial solution: trivial solution for small sizes.
- Top bottom: If instance is n, apply greedy criterion to reduce the instance to a smaller sized one (apply recursion).
- Bottom up: Construct the solution of an instance of size n by making local choices based on the greedy criterion..

Greedy algorithms II

- Optimal substructure:
  - if an optimal solution to the problem contains within it optimal solutions to subproblems.
- Greedy choice property:
  - we can assemble a globally optimal solution by making locally optimal (greedy) choices, i.e., when we are considering which choice to make, we make the choice that looks best in the current problem, without considering results from subproblems.
- Resursive top down approach make local choices and never changes them

Greedy algorithms III

- A *greedy algorithm* always makes the choice that looks best at the moment
  - Everyday examples??
  - A locally optimal choice will actually lead to a globally optimal solution
  - For some problems, it does work!
  - Optimal substructure
  - Greedy choice

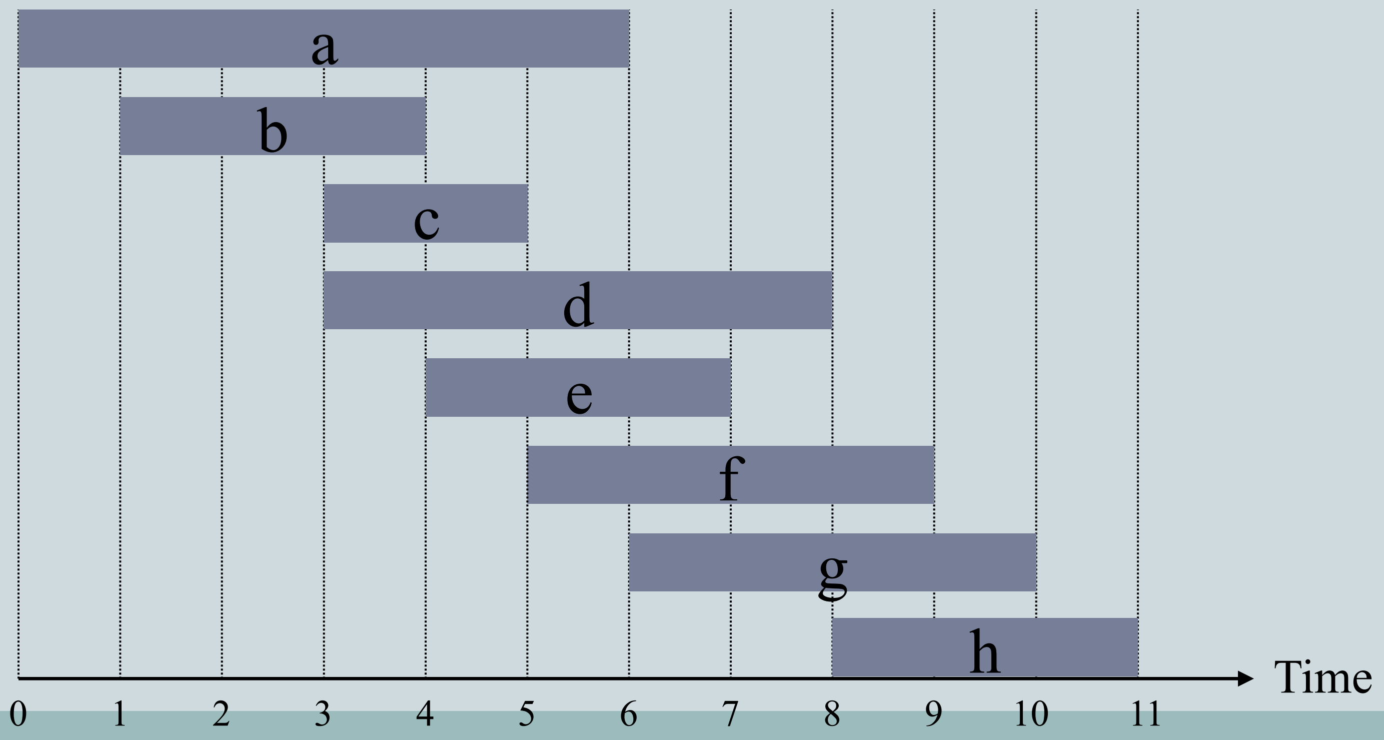
Activity-Selection (Interval Scheduling) Problem

- Problem: get your money's worth out of a carnival
  - Buy a wristband that lets you onto any ride
  - Lots of rides, each starting and ending at different times
  - Your goal: ride as many rides as possible

- Welcome to the *activity selection problem*
- *Another application?*

Interval Scheduling

- Interval scheduling.
  - Job j starts at s_j and finishes at f_j.
  - Two jobs compatible if they don't overlap.
  - Goal: find maximum subset of mutually compatible jobs.

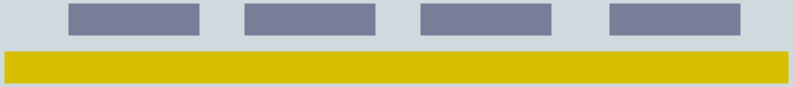


Interval Scheduling: Greedy Algorithms

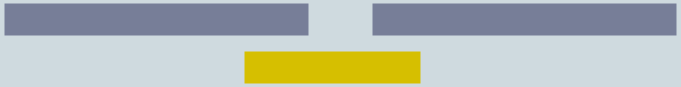
- Greedy template. Consider jobs in some order. Take each job provided it's compatible with the ones already taken.
  - [Earliest start time] Consider jobs in ascending order of start time s_j.
  - [Earliest finish time] Consider jobs in ascending order of finish time f_j.
  - [Shortest interval] Consider jobs in ascending order of interval length $f_j - s_j$.
  - [Fewest conflicts] For each job, count the number of conflicting jobs c_j. Schedule in ascending order of conflicts c_j.

Interval Scheduling: Greedy Algorithms

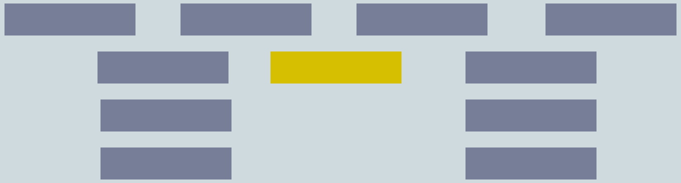
- Greedy template. Consider jobs in some order. Take each job provided it's compatible with the ones already taken.



breaks earliest start time



breaks shortest interval



breaks fewest conflicts

Interval Scheduling: Greedy Algorithm

- Greedy algorithm. Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

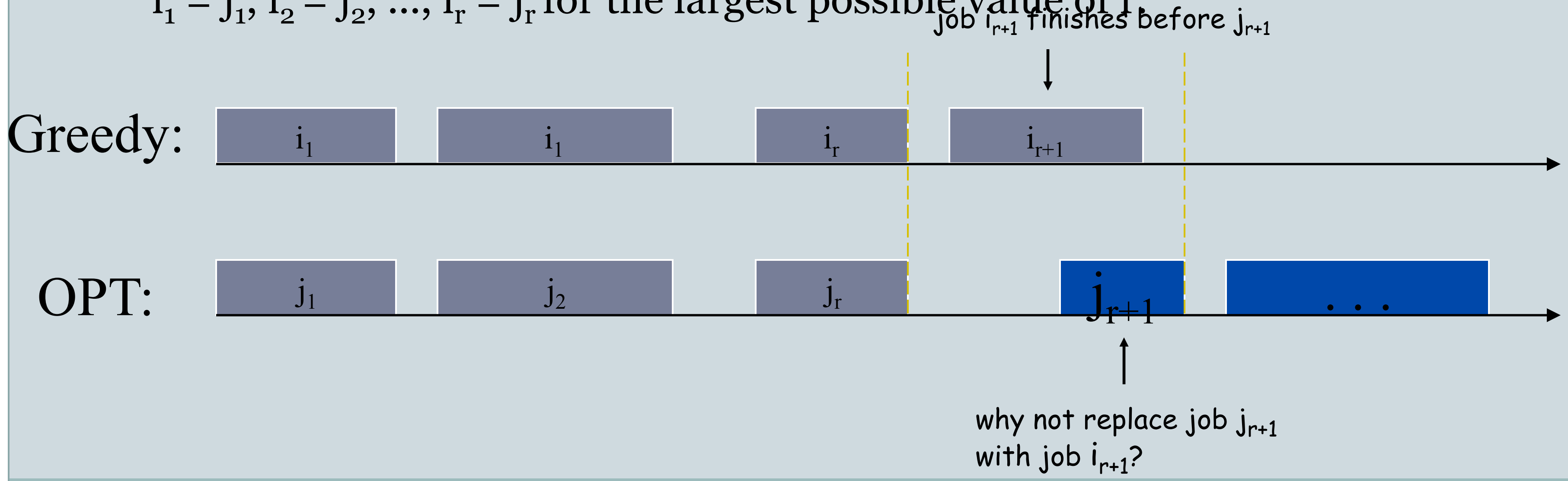
```
Sort jobs by finish times so that $f_1 \le f_2 \le ... \le f_n$.

  jobs selected
A ← ϕ
for j = 1 to n {
   if (job j compatible with A)
      A ← A ∪ {j}
}
return A
```

- Implementation. O(n log n).
  - Remember job j* that was added last to A.
  - Job j is compatible with A if $s_j \ge f_{j^*}$.

Interval Scheduling: Analysis

- Theorem. Greedy algorithm is optimal.

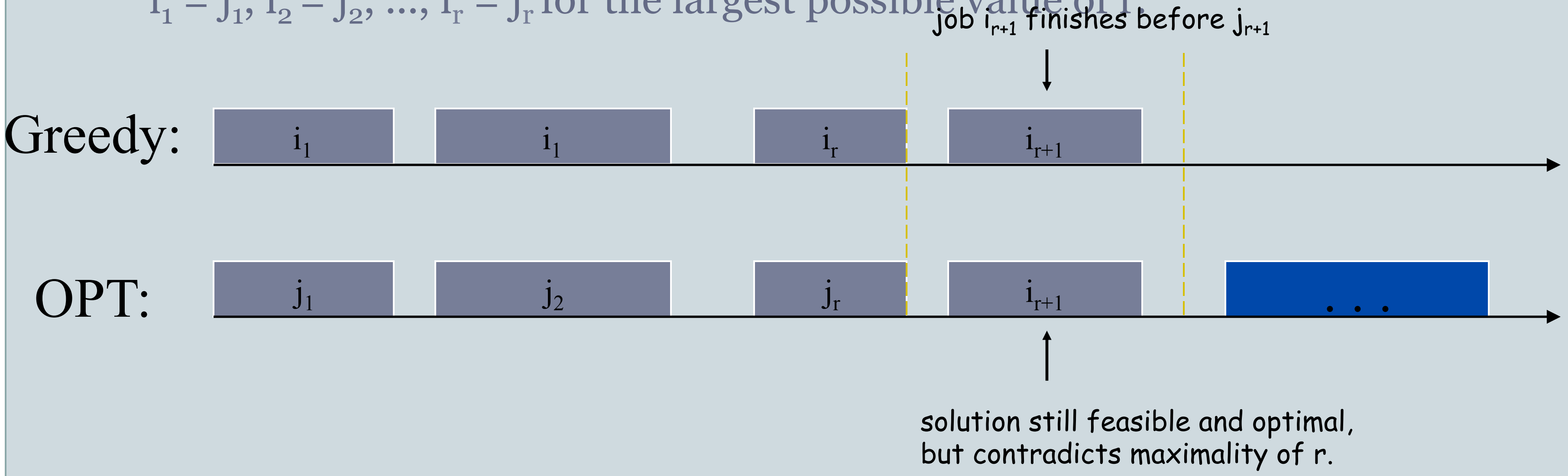
- Pf. (by contradiction)
  - Assume greedy is not optimal, and let's see what happens.
  - Let $i_1, i_2, \ldots i_k$ denote set of jobs selected by greedy.
  - Let $j_1, j_2, \ldots j_m$ denote set of jobs in the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of r.


job i_{r+1} finishes before j_{r+1}
Greedy:
i_1
i_1
i_r
i_{r+1}
OPT:
j_1
j_2
j_r
j_{r+1}
. . .
why not replace job j_{r+1} with job i_{r+1}?

Interval Scheduling: Analysis

- Theorem. Greedy algorithm is optimal.

- Pf. (by contradiction)
  - Assume greedy is not optimal, and let's see what happens.
  - Let $i_1, i_2, \ldots i_k$ denote set of jobs selected by greedy.
  - Let $j_1, j_2, \ldots j_m$ denote set of jobs in the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of r.

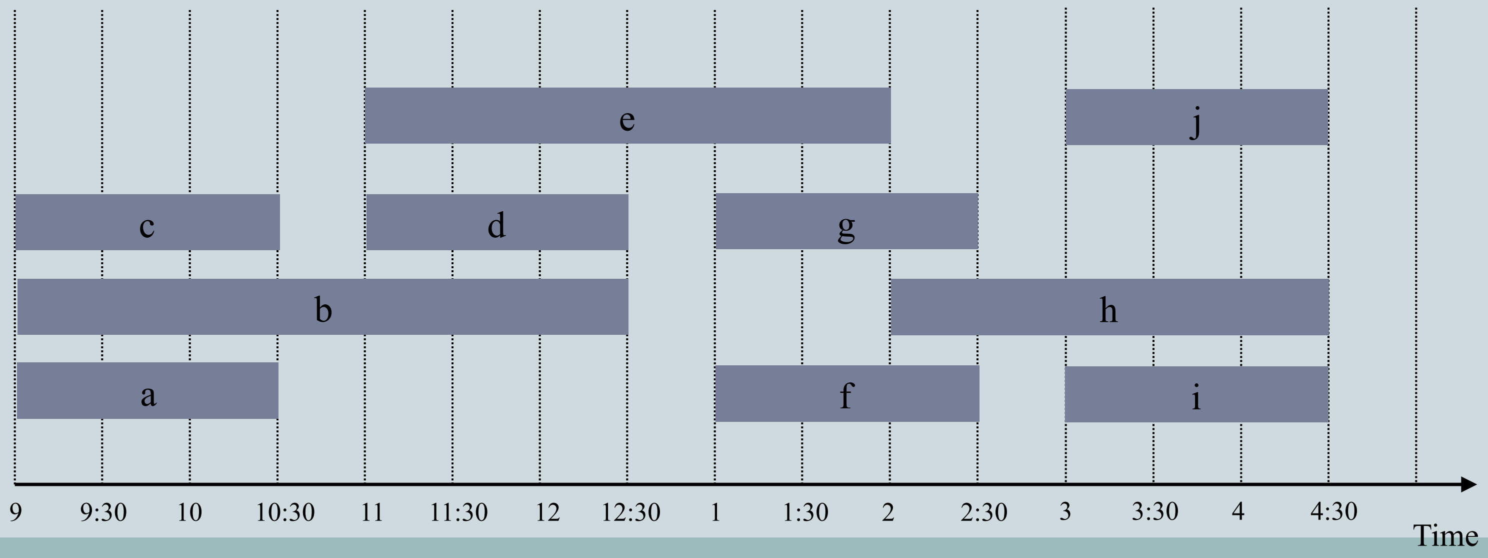

job i_{r+1} finishes before j_{r+1}
Greedy:
i_1
i_1
i_r
i_{r+1}
OPT:
j_1
j_2
j_r
i_{r+1}
. . .
solution still feasible and optimal, but contradicts maximality of r.

2. Greedy Choice Property

- Activity selection problem has the *greedy choice* property:
  - Locally optimal choice $\Rightarrow$ globally optimal solution
  - Them 17.1: if S is an activity selection problem sorted by finish time, then ∃ optimal solution $A \subseteq S$ such that {1} $\in A$
    - Sketch of proof: if ∃ optimal solution B that does not contain {1}, can always replace first activity in B with {1} (*Why?*). Same number of activities, thus optimal.

Interval Partitioning

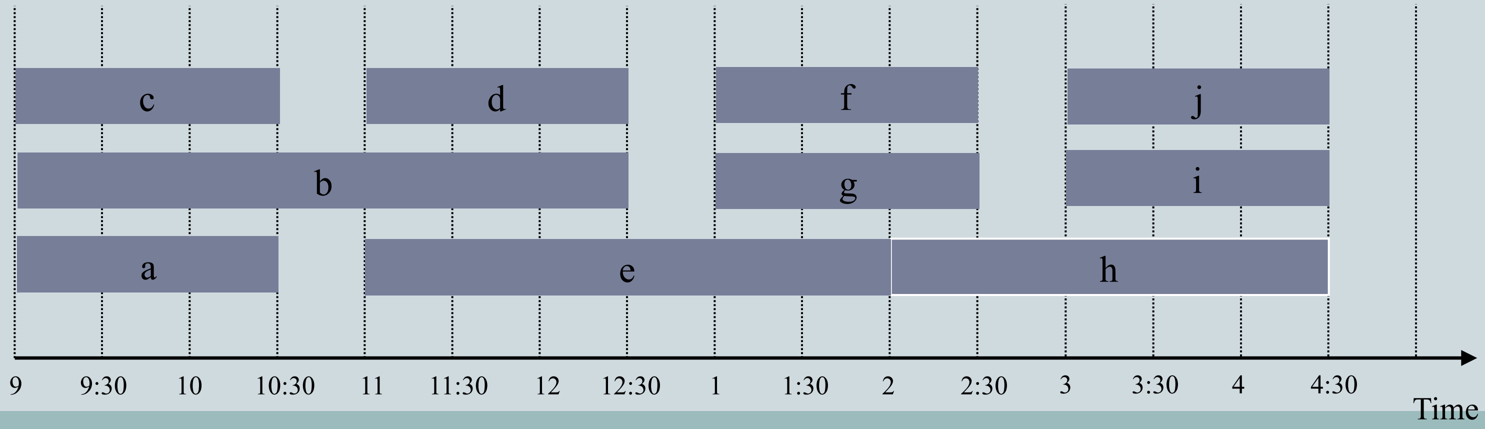
- Interval partitioning.
  - Lecture j starts at s_j and finishes at f_j.
  - Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.
- Ex: This schedule uses 4 classrooms to schedule 10 lectures.



Interval Partitioning

- Interval partitioning.
  - Lecture j starts at s_j and finishes at f_j.
  - Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

- Ex: This schedule uses only 3.



Interval Partitioning: Lower Bound on Optimal Solution

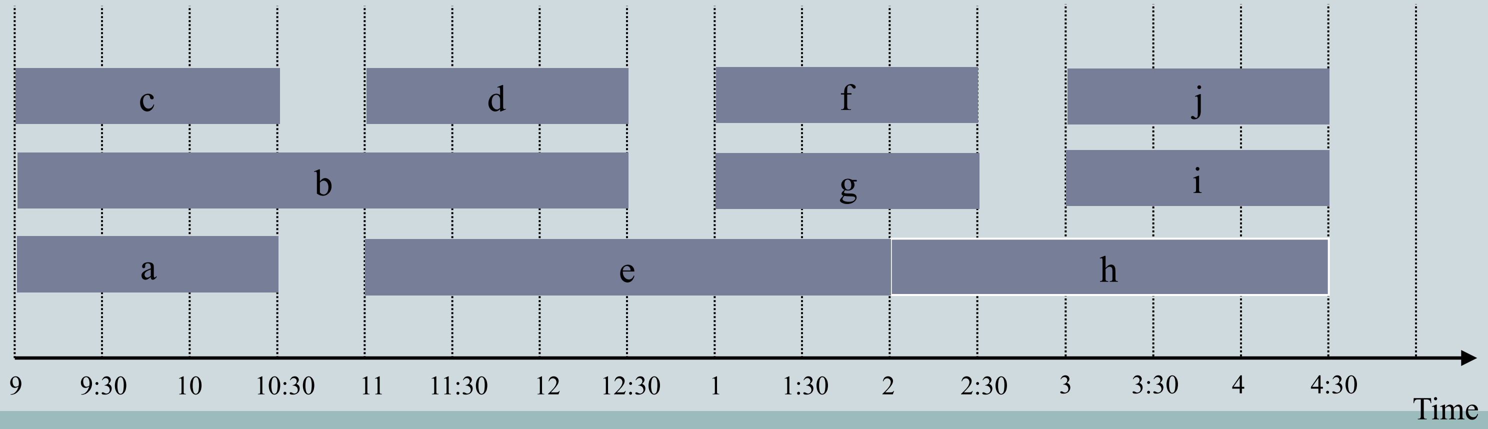
- Def. The depth of a set of open intervals is the maximum number that contain any given time.

- Key observation. Number of classrooms needed $\geq$ depth.

  ↑
  a, b, c all contain 9:30

- Ex: Depth of schedule below = 3 $\Rightarrow$ schedule below is optimal.

- Q. Does there always exist a schedule equal to depth of intervals?



Interval Partitioning: Greedy Algorithm

- Greedy algorithm. Consider lectures in increasing order of start time: assign lecture to any compatible classroom.

```
Sort intervals by starting time so that $s_1 \le s_2 \le \ldots \le s_n$.
d ← 0   ⟵ number of allocated classrooms

for j = 1 to n {
   if (lecture j is compatible with some classroom k)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d ← d + 1
}
```

- Implementation. O(n log n).
  - For each classroom k, maintain the finish time of the last job added.
  - Keep the classrooms in a priority queue.

Interval Partitioning: Greedy Analysis

- Observation. Greedy algorithm never schedules two incompatible lectures in the same classroom.

- Theorem. Greedy algorithm is optimal.
- Pf.
  - Let d = number of classrooms that the greedy algorithm allocates.
  - Classroom d is opened because we needed to schedule a job, say j, that is incompatible with all d-1 other classrooms.
  - Since we sorted by start time, all these incompatibilities are caused by lectures that start no later than s_j.
  - Thus, we have d lectures overlapping at time $s_j + \varepsilon$.
  - Key observation $\Rightarrow$ all schedules use $\geq$ d classrooms. ▪

Huffman Codes Encoding and Compression of Data

- Fax Machines
- ASCII Variations on ASCII
- min number of bits needed
- cost of savings
- patterns
- modifications

The Basic Algorithm

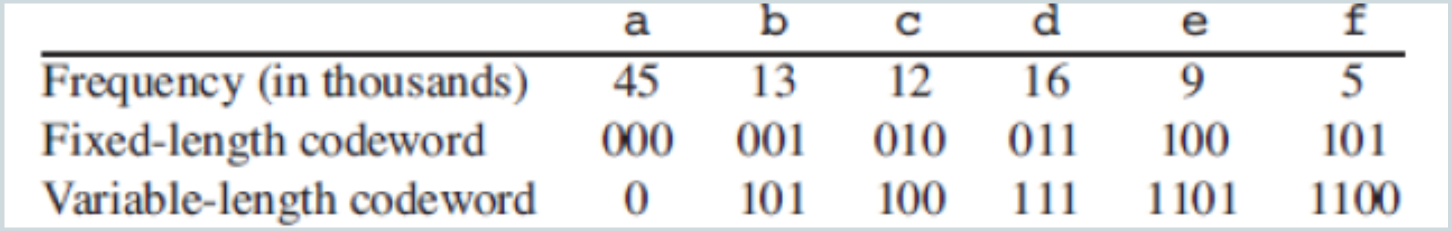
- Huffman coding is a form of statistical coding
- Not all characters occur with the same frequency!
- Yet all characters are allocated the same amount of space
  - *1 char = 1 byte, be it e or x*
- Code word lengths are no longer fixed like ASCII.
- Code word lengths vary and will be shorter for the more frequently used characters.

The Basic (Greedy) Huffman Algorithm

1. Scan text to be compressed and count occurrence of all characters.
2. Sort or prioritize characters based on the number of occurrences in text (frequencies).
3. Build Huffman code tree based on the prioritized list.
4. Perform a traversal of the tree to determine all code words.
5. Scan text again and create new file using the Huffman codes.

Building a Tree Scan the original text

- A character-coding problem.
- A data file of 100,000 characters contains only the characters a–f, with the frequencies indicated.



	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

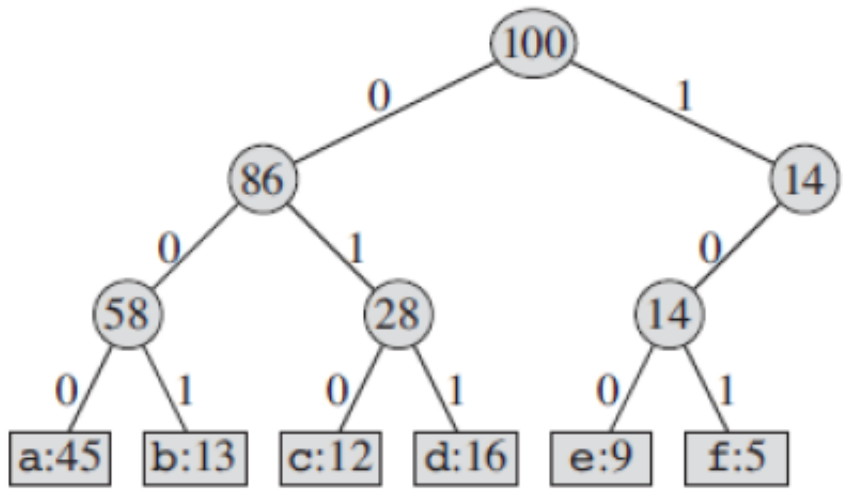
- 300,000 bits (fixed) vs. 224,000 bits (variable length)

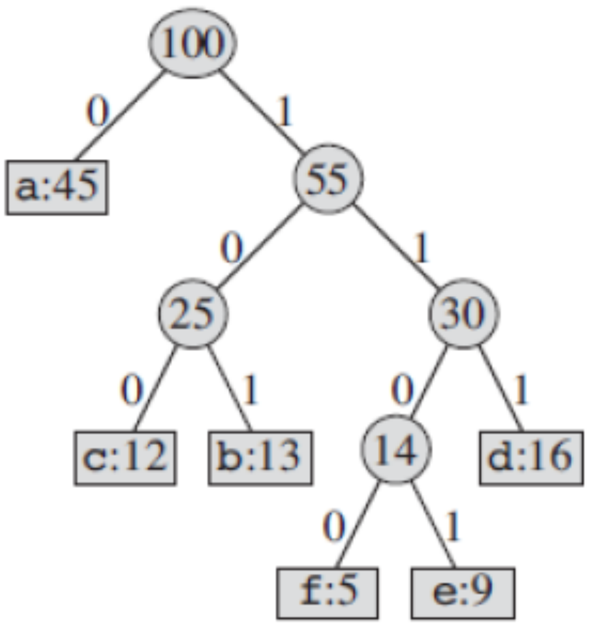
Building a Tree -Prioritize characters

- Create binary tree nodes with character and frequency of each character.

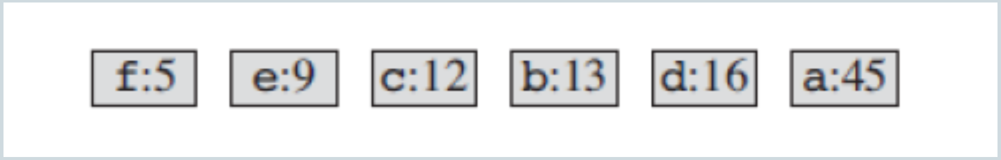
- Prefix Codes: We consider here only codes in which no codeword is also a **prefix** of some other codeword.

Tree: fixed vs optimal prefix code

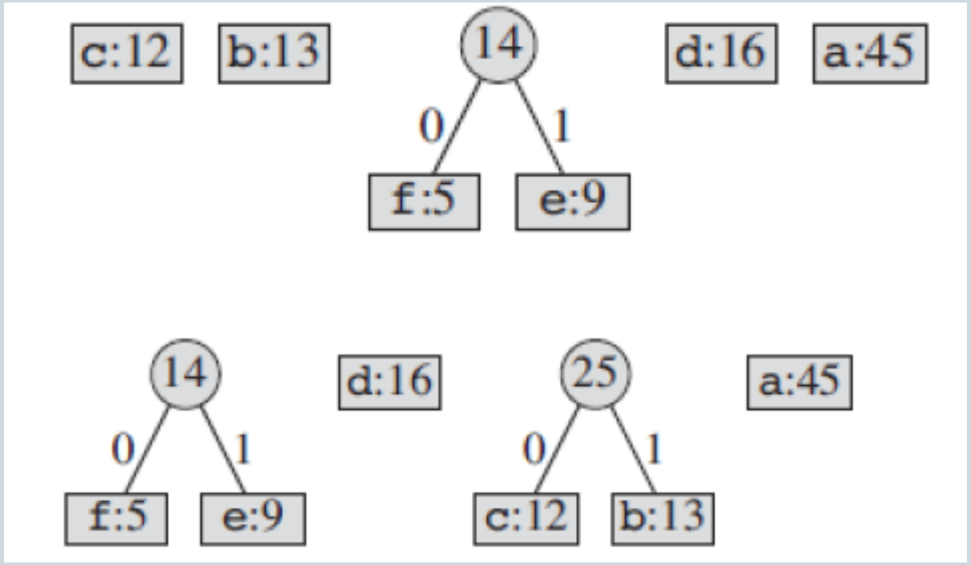




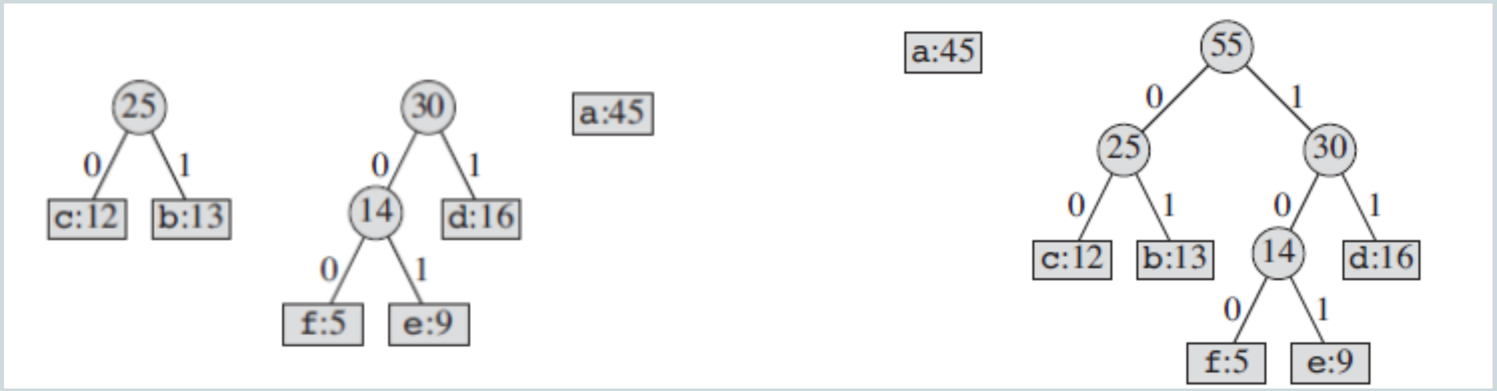
Building a Tree I

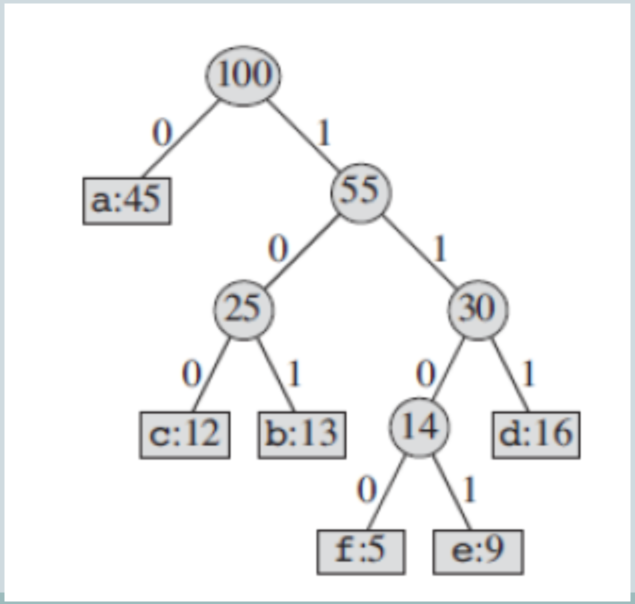


- Pair the two smallest frequencies into a tree



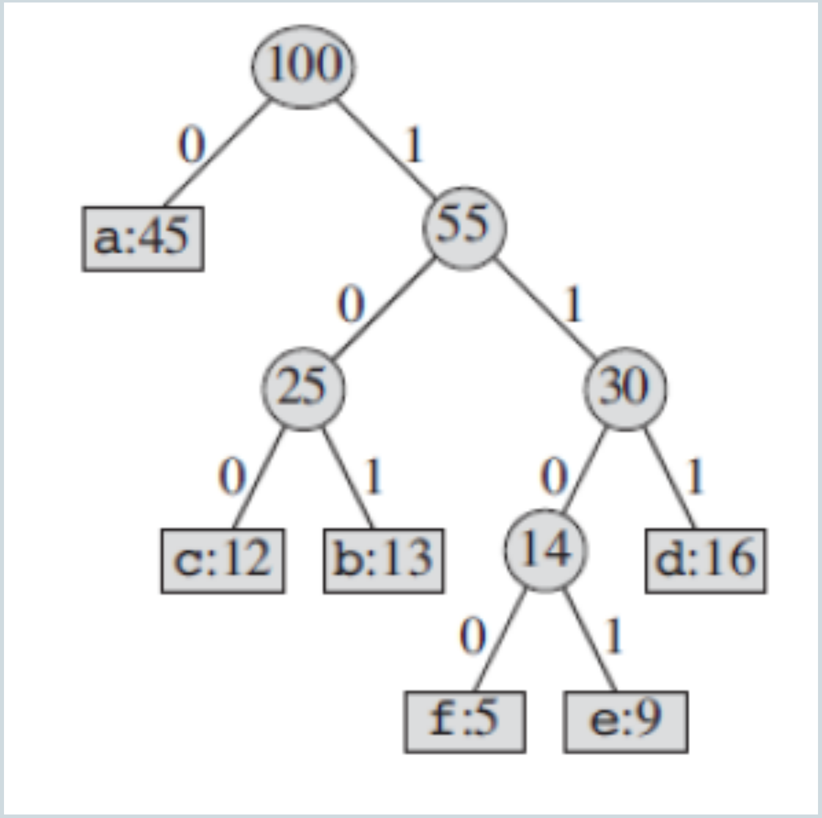
Building the tree II





Encoding the File - Traverse Tree for Codes

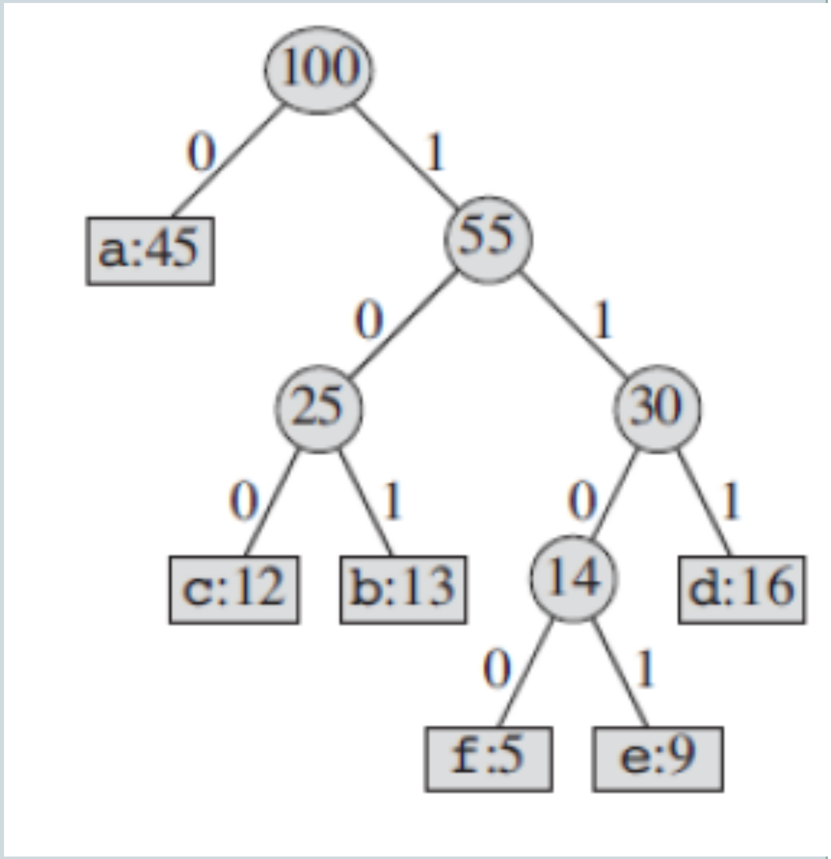
- Perform a traversal of the tree to obtain new code words
- Going left is a 0 going right is a 1
- code word is only completed when a leaf node is reached



Encoding the File - Traverse Tree for Codes

- Char Code

a	0
b	101
c	100
d	111
e	1101
f	1100



Encoding the File

- Rescan text and encode file using new code words

Decoding the File

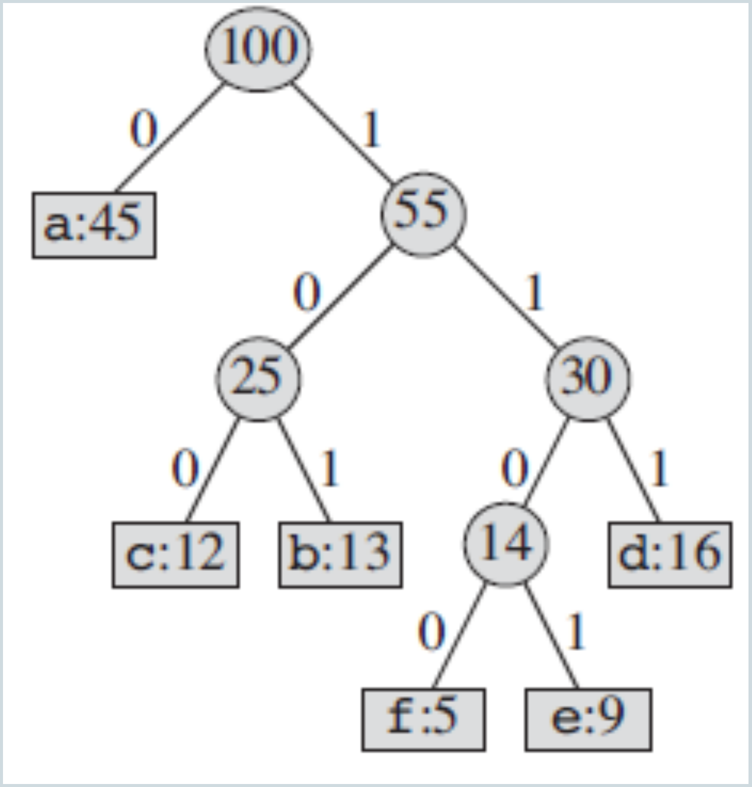
How does receiver know what the codes are?

- Tree constructed for each text file.
- Considers frequency for each file.
- Big hit on compression, especially for smaller files
- Tree predetermined
  - based on statistical analysis of text files or file types
- Data transmission is bit based versus byte based

Decoding the File

- Once receiver has tree it scans
- 0 => go left
- 1 => go right

- 001011101 is aabe



Clicker Question 13.1

- What is an optimal Huffman code for the following set of frequencies? a:6 b:9 c:4 d:2 e:1
- *run Huffman, here are the pairs*
  - a: 10, b: 000, c: 001,d: 010, e:1100
  - a: 10, b: 00, c:111,d:1101, e:1110
  - a: 10, b: 0, c:111,d: 1101, e:1100
  - a: 01, b: 1, c:011,d: 0101, e:0100

Summary

- Huffman coding is a technique used to compress files for transmission running time O(nlgn)
- Uses statistical coding
  - more frequently used symbols have shorter code words
- Works well for text and fax transmissions
- An application that uses several data structures