Announcements

Announcements

- HW4 is out today (due April 1)
  - Tuesday - no class

Greedy Algorithms

Introduction to Greedy Algorithms

- For some algorithms, you need to look ahead or revise past decisions to get the best solution
- Greedy algorithms build a solution piece-by-piece, without revising past decisions
  - What greedy algorithms have we seen?

Huffman Encoding

- We want to encode a string of characters using binary
- Example: Alphabet has four characters A – D
- One way:
  - A = 00
  - B = 01
  - C = 10
  - D = 11
- AABDC = 0000011110
- How many digits are in our string?

A , B
0 1

$\log_2 4 = 2$ digits

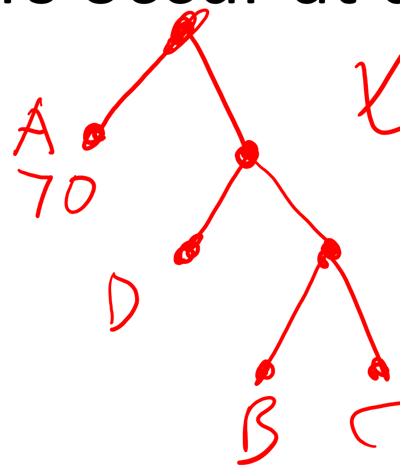
$\log_2 N$ digits per char.

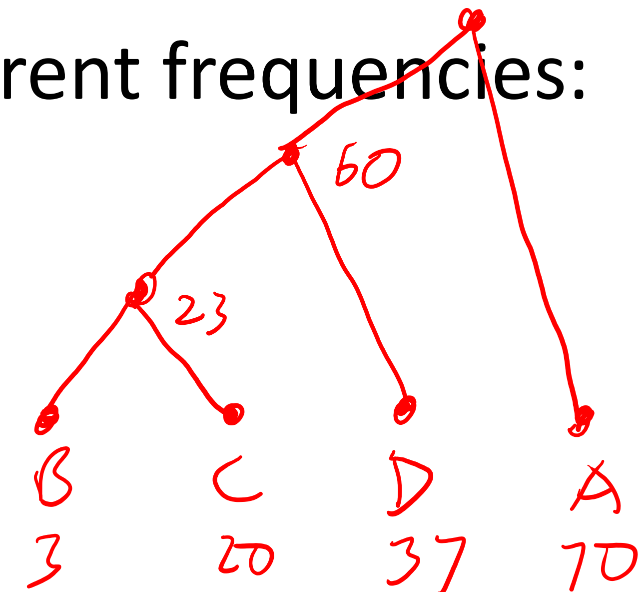
N chars in lang.
string has length k

$(k \cdot \log_2 N)$

Huffman Encoding

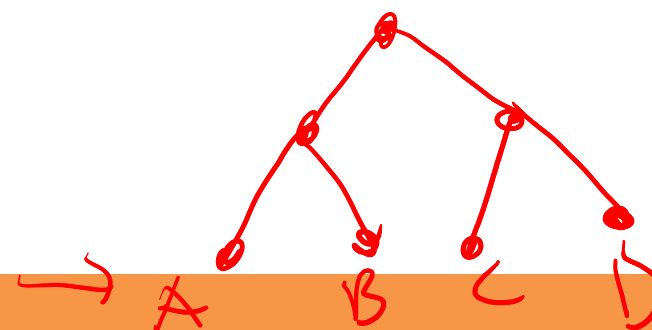
- Suppose characters occur at different frequencies:
  - A: 70 /130
  - B: 3 /130
  - C: 20 /130
  - D: 37 /130





- Use fewer digits for more frequent characters!

reduce space required to store encoding



Huffman Encoding

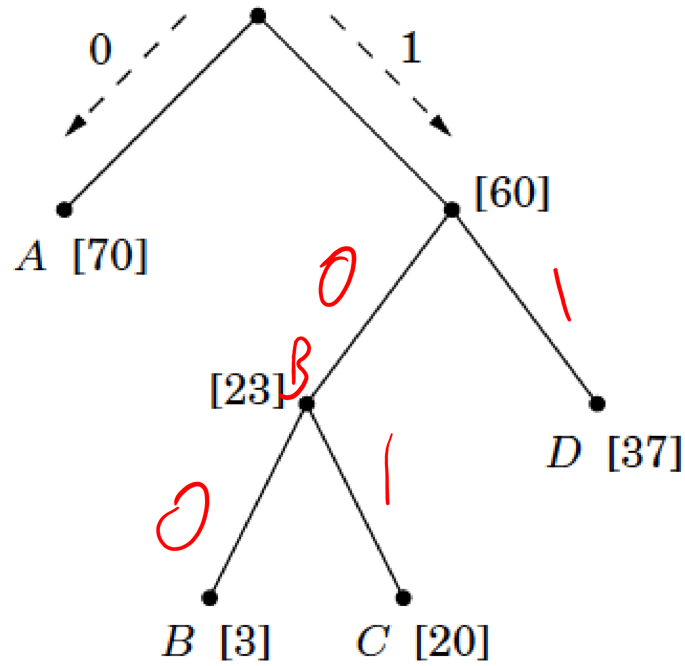
- Assign the following strings:
  - A: 0
  - B: 00
  - C: 01
  - D: 1
- Then AABDC = 0***0***00***1***01 7 digits
- Anyone see a problem with this? ambiguous

Huffman Encoding

- Solution: Make sure that no character encoding is a prefix of any other character's encoding

char

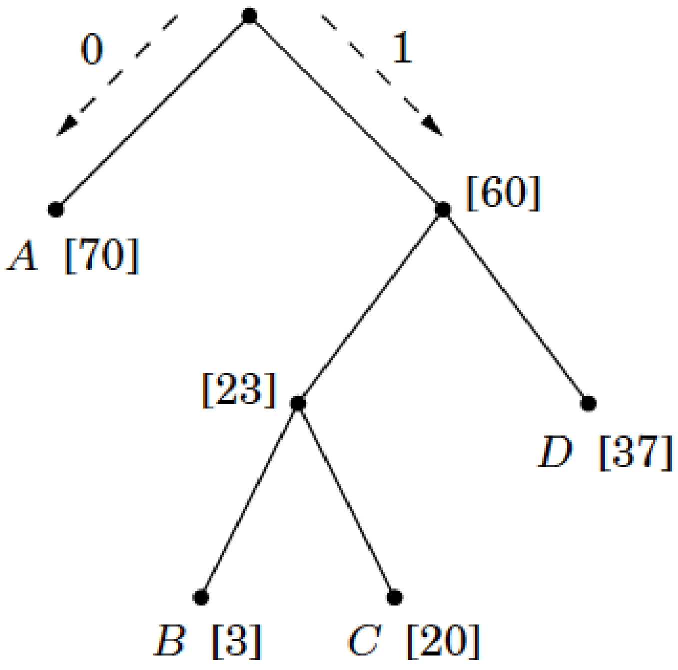
Symbol	Codeword
A	0
B	100
C	101
D	11



Huffman Encoding

- What is the length of the encoding of a character, in terms of the encoding tree? depth of char. in tree

Symbol	Codeword
A	0
B	100
C	101
D	11



Huffman Encoding

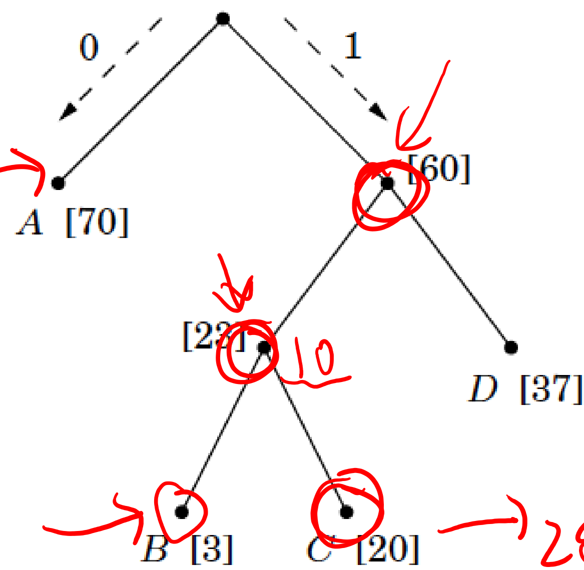
language # char's

$$\text{cost of tree} = \sum_{i=1}^{n} f_i \cdot (\text{depth of } i\text{th symbol in tree})$$

length of encoding

- The total cost of a tree is the sum of the frequencies for all leaves and internal ~~roots~~ nodes, except the root

long document how many times does a char appear



20 · 3

ancestors (excluding roots) is = depth

Huffman Encoding

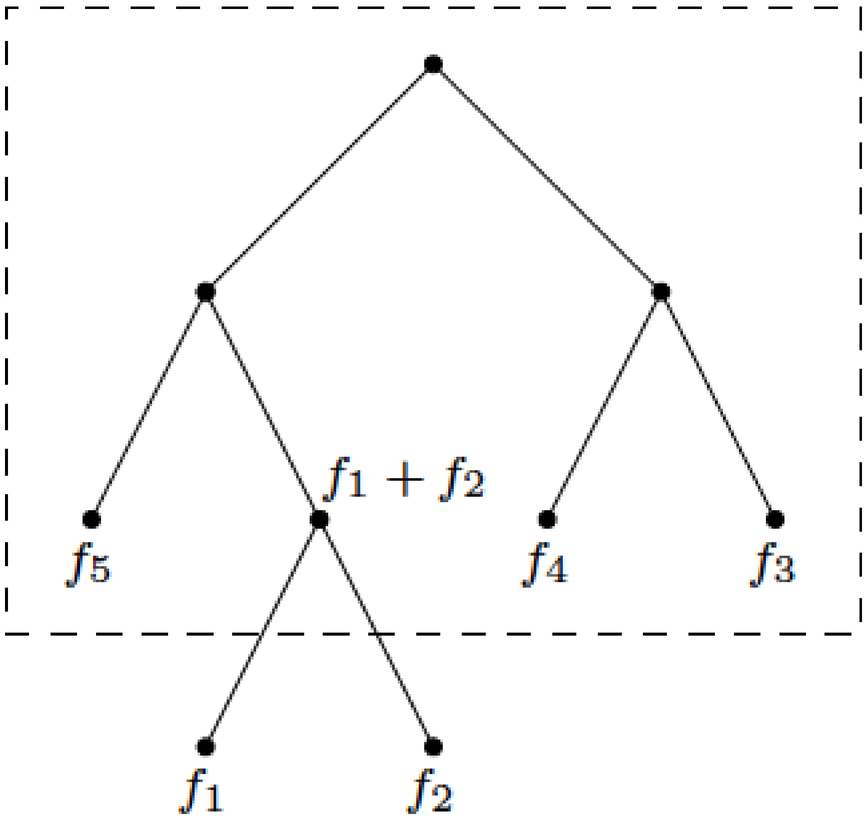
$$\text{cost of tree} = \sum_{i=1}^{n} f_i \cdot (\text{depth of } i\text{th symbol in tree})$$

- Where do we want the infrequent characters to be in the tree? greater depth
- Where do we want the frequent characters to be in the tree? closer to root

Huffman Encoding

- Idea: find the two characters that are least frequent, make them children of a new internal node
  - What is the frequency of the internal node? sum of the two char frequencies
- At each step, find two least frequent characters/nodes (could be an internal node!), and make them children of a new internal node

Huffman Encoding



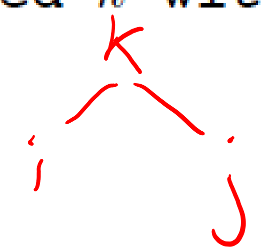
Huffman Encoding

$O(n \log n)$

<u>procedure Huffman</u>(f)
Input: An array $f[1 \cdots n]$ of frequencies
Output: An encoding tree with n leaves

Dijkstra's

let H be a priority queue of integers, ordered by f
for $i = 1$ to n: insert(H, i) ← $n \log n$
for $k = n + 1$ to $2n - 1$:
  $i =$ deletemin(H), $j =$ deletemin(H) $2n \log n$
  create a node numbered k with children i, j
  $f[k] = f[i] + f[j]$
  insert(H, k)



$n \log n$

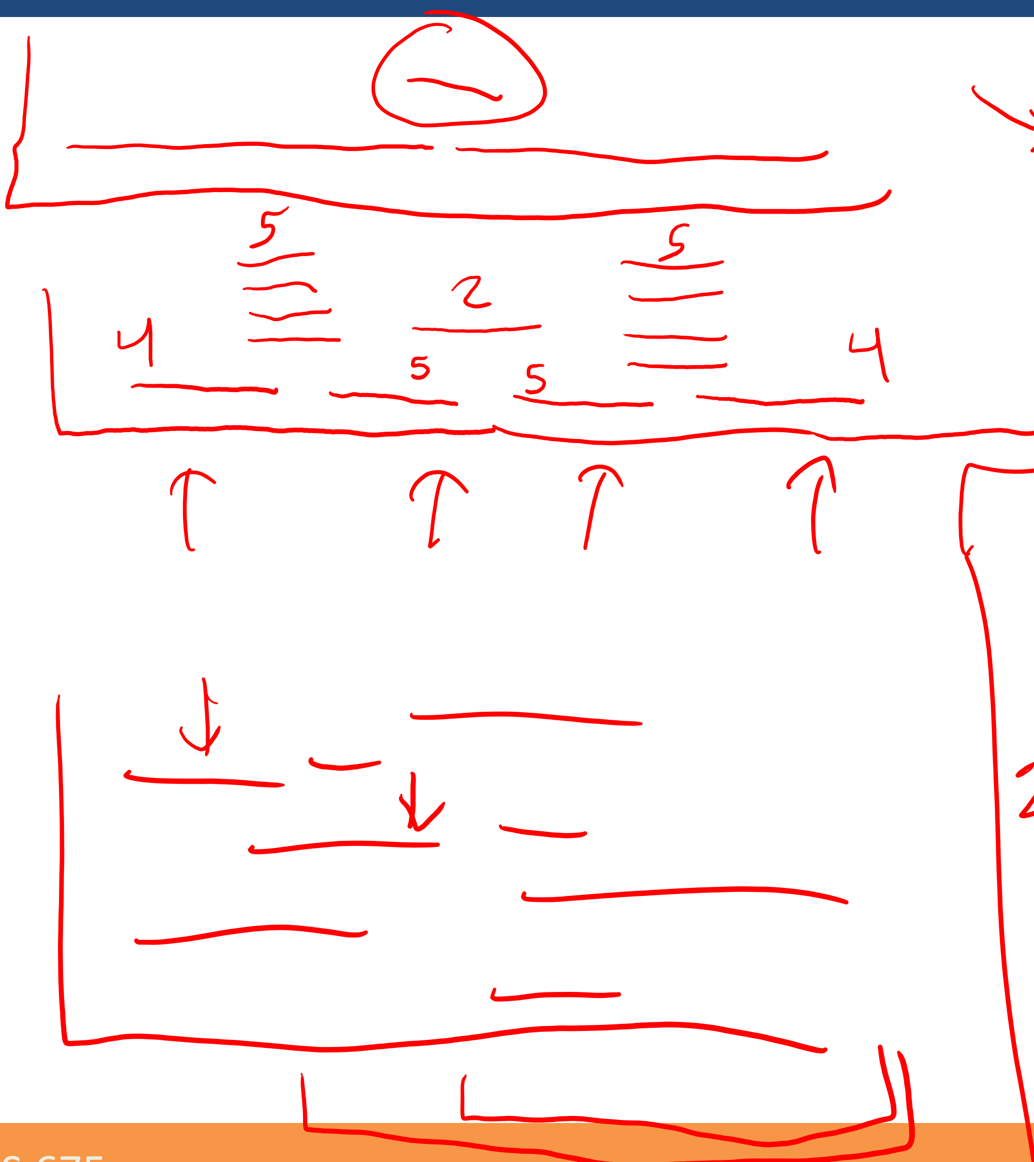
Interval Scheduling

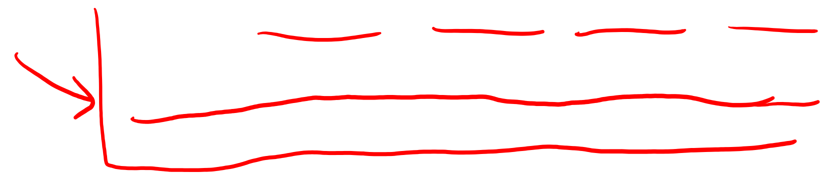
- You are given a series of lectures to attend. Lecture *i* runs from time s_i to t_i. Design an algorithm to determine which lectures to attend so you attend as many as possible, without conflicts.
- Options:
  - Sort by starting time?
  - Sort by ending time?
  - Sort by # of conflicts?
  - Sort by total length?

no overlap

1. Sol. comps: lectures
2. Sort according to ?
3. Add comps in order as long as a constraint is not violated

Solving Interval Scheduling





Greedy IS

1. Sort in increasing order of end time
2. Add them in order as long as it doesn't overlap anything added

Proof of Correctness for Int. Scheduling

Claim: GreedyIS is optimal

Proof: Let $A = \{a_1, \dots, a_k\}$ be the set of lectures returned by GreedyIS. Let $OPT = \{b_1, \dots, b_m\}$ be the optimal solution. We want to show $k = m$. Assume both A and OPT are sorted by ~~start~~ end time (increasing order). Assume all start times distinct.

Proof of Correctness for Int. Scheduling

We will prove by induction that for all r, $t_{A_r} \le t_{OPT_r}$.

Base case: $r=1$. We want that $t_{A_1} \le t_{OPT_1}$. Because of how GreedyIS works, the first element it inspects is the one with earliest end time. This element is definitely added. So t_{A_1} is the element with earliest end time, so $t_{A_1} \le t_{OPT_1}$.

Proof of Correctness for Int. Scheduling

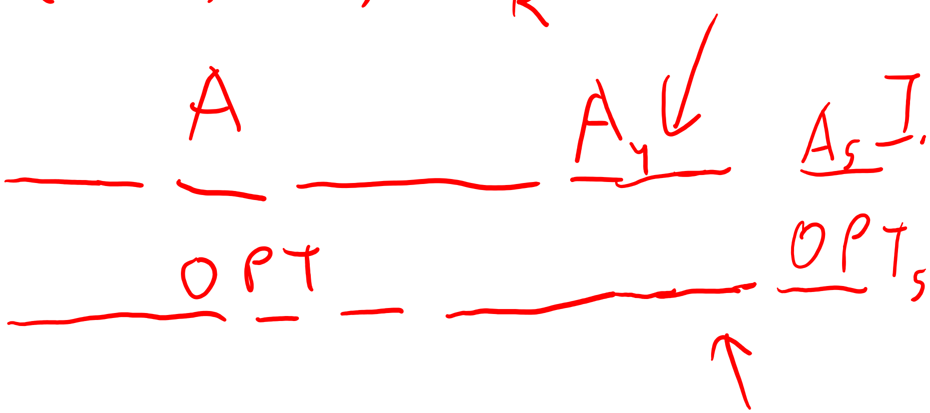
I.H.: Assume that $t_{A_r} \leq t_{OPT_r}$ for some r.

I.S.: We need to show that $t_{A_{r+1}} \leq t_{OPT_{r+1}}$.

Assume for a contradiction that $t_{A_{r+1}} > t_{OPT_{r+1}}$. Because OPT_{r+1} is the next element after OPT_r in the sorted order, it doesn't conflict with anything earlier. ~~Beca~~ By I.H., $t_{A_r} \leq t_{OPT_r}$, so then OPT_{r+1} also doesn't conflict with $\{A_1 \cdots A_r\}$.

Proof of Correctness for Int. Scheduling

So if $t_{A_{r+1}} > t_{OPT_{r+1}}$, this means that GreedyIS didn't pick the earliest end time that had no conflicts with $\{A_1, \dots, A_R\}$. Contradiction.



I.H.: $t_{A_R} \le t_{OPT_r}$

Proof of Correctness for Int. Scheduling

Now, we need to show that $k = m$. Certainly, $k \leq m$, because OPT is optimal, so we need to show that $k \nless m$. Assume for a contradiction that $k < m$. Let's consider $r = k$. We know that $t_{A_k} \leq t_{OPT_k}$. So then look at interval OPT_{k+1}. This doesn't interfere with OPT_k, so it also doesn't interfere with A_k. So when Greedy IS sees it, it would add it! Contradiction. $\square$

Proving Correctness of Greedy Algorithms

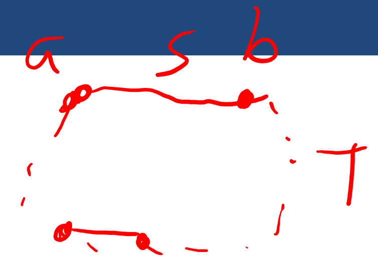
$a_1 \cdots a_k$

$o_1 \cdots o_m$

- "Greedy stays ahead"
  1. Label your solution A = {a_1, ..., a_k}, label the optimal solution O = {o_1, ..., o_m}. (Note, the ordering of these elements depends on your particular problem.)
  2. Define a quality measure of how good a solution (or partial solution) is

     end time of last interval
  3. Prove that the greedy method is always at least as good as the optimal solution for all indices *r*

Proving Correctness of Greedy Algorithms



- "Greedy exchange"
  1. Label your solution A = {a_1, ..., a_k}, label the optimal solution O = {o_1, ..., o_m}. (Note, the ordering of these elements depends on your particular problem.)
  2. Find a way in which your solution A differs from the optimal solution
  3. Show that if you swap the different element from A into O, the solution does not decrease in quality
  4. Repeat until O is converted into A; thus showing that A is at least as good as O