CS 170 DISCUSSION 7

GREEDY ALGORITHMS

Raymond Chan
UC Berkeley Fall 17

APPROXIMATING SET COVER

- See board or notes/video

GREEDY ALGORITHM PROOFS

- Goal is to maximize or minimize a "cost" subject to constraints.
- Most greedy algorithms are linear or O(n log n) time.
- Approach 1:
  - Exchange Argument
- Approach 2:
  - Greedy is at least better than Optimal

EXCHANGE ARGUMENT

- Suppose you have an optimal solution S^* and greedy solution S.
- Look at the first value in which greedy solution and optimal differ. Usually talk about an ordering. Let's say greedy produced X and optimal has Y at that position of the ordering.
- Switch Y with X in optimal solution. Show that total cost is not worse (either the same or better).
- By induction, we can iterate over both solutions, exchange values until optimal becomes greedy. Since total cost has not become worse off in each step, greedy solution is optimal.
- Discussion today: Service Scheduling

GREEDY IS AT LEAST BETTER THAN OPTIMAL

- Suppose you have an optimal solution S^* and greedy solution S.
- Look at the first value in which greedy and optimal differ. Usually talk about an ordering.
- Prove that at this point, the greedy solution is at least as better as the optimal solution. And by induction, the greedy solution is better at each differing value.
- Prove optimality via contradiction. Assume greedy is not optimal. Use above point to derive a contradiction.
- Discussion today: Meeting Scheduling

APPROACH 1 VS 2

- Use exchange argument (1) instead of contradictory proofs (2) because there could be multiple optimal solutions.

- Both require induction.

- Nuanced difference. Exchange argument takes some optimal ordering and swaps elements in the ordering that violate greedy heuristic. We will reach greedy without increasing cost.

- Approach 2 compares specific differing values in greedy and optimal solutions. Choosing the greedy value at that point is at least as good as optimal.

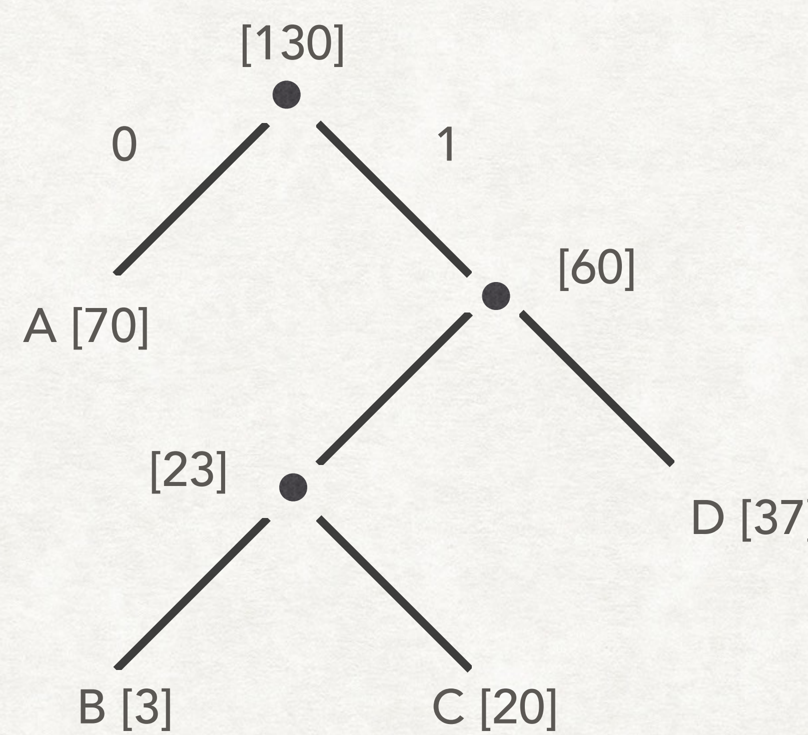
HUFFMAN ENCODING

- Motivation: Given symbols and their respective probabilities, can we encode the symbols to prevent ambiguity and minimize length of longest encoding?
- Prefix-free encoding: no codeword can be a prefix of another codeword.
- Use full binary tree. Nodes have 0 or 2 children.
- Leaves are symbols.
- Internal nodes (v) have 2 children (u, w) with probability (or frequency) equal to sum of children's probabilities. $f(v) = f(u) + f(w)$

HUFFMAN ENCODING

- The frequency of a (sub)tree at it's root is the sum of all frequencies.

Symbol	Frequency	Codeword
A	70	0
B	3	100
C	20	101
D	37	11



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

Lower the symbol, more it's frequency gets repeated.
Smallest frequency should be at bottom.

HUFFMAN ENCODING

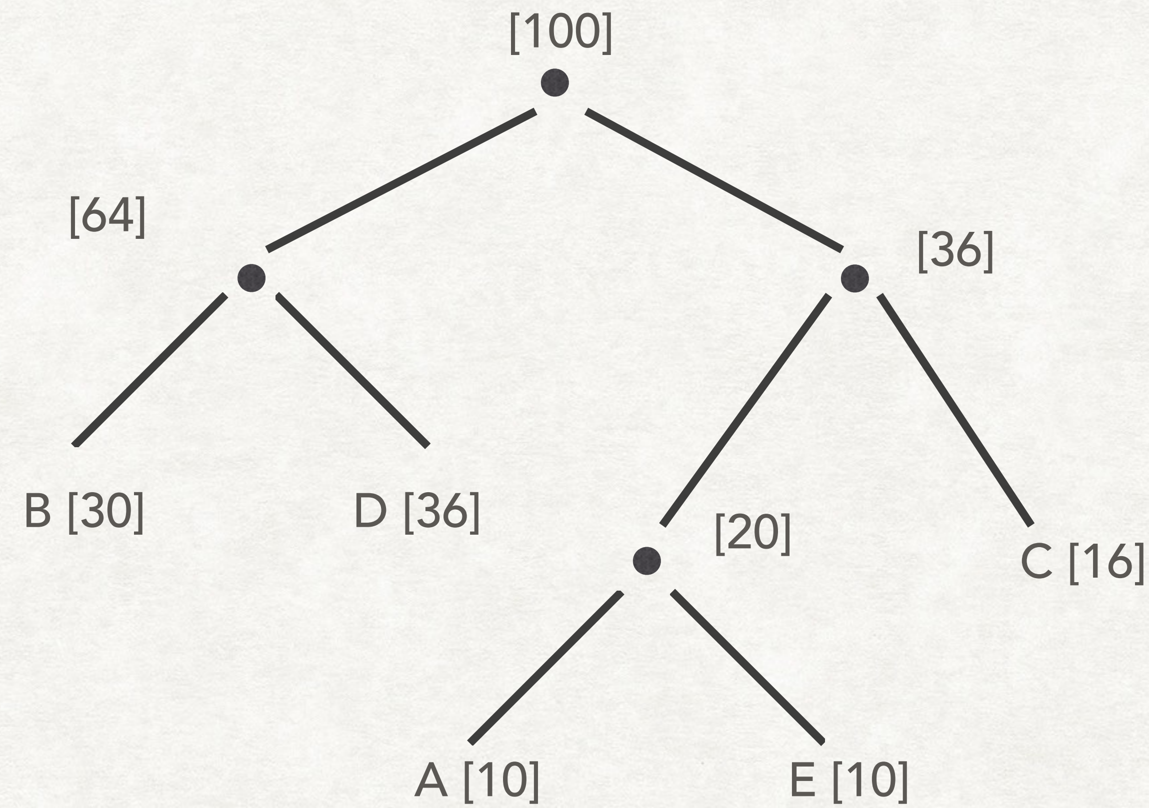
```
Huffman(f[1..n])
  Make each symbol into single tree node
  While more than one tree:
    Merge two lowest frequency trees into a new tree
```

```
Huffman(f[1..n])
  Priority Queue PQ
  for i in {1..n}: PQ.insert(i)
  for k = n + 1 to 2n - 1:
    i, j = PQ.deleteMin(), PQ.deleteMin()
    Create node k with children i, j
    f[k] = f[i] + f[j]
    PQ.insert(k)
```

HUFFMAN ENCODING

- When inserting internal node into PQ, it may be popped off immediately or later.

Symbol	Frequency
A	10/100
B	30/100
C	16/100
D	34/100
E	10/100



Internal node 20 gets popped off with C in iteration after being pushed into PQ.
Internal node 36 was until B and D gets popped off before being removed from PQ.