CS 170 Section 5
Greedy Algorithms II

Owen Jow | owenjow@berkeley.edu

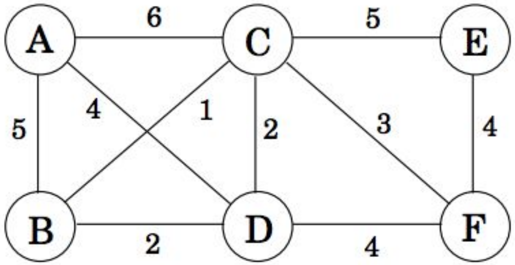
Agenda

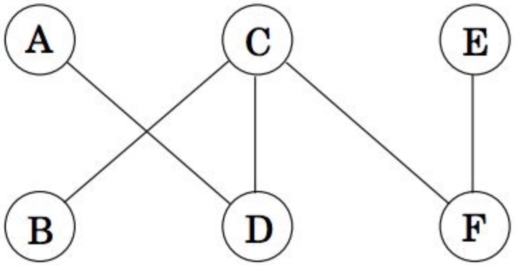
- Greedy algorithms
  - MST construction
  - Huffman encoding
  - Horn formulas
- Side note: additional topics in scope for MT2 are greedy algorithms, MSTs, DP, LP, and max flow

Greedy Algorithms

Minimum Spanning Trees

- An MST represents the “cheapest” set of edges that connects all the nodes
- It is a connected, acyclic graph with the minimum total weight





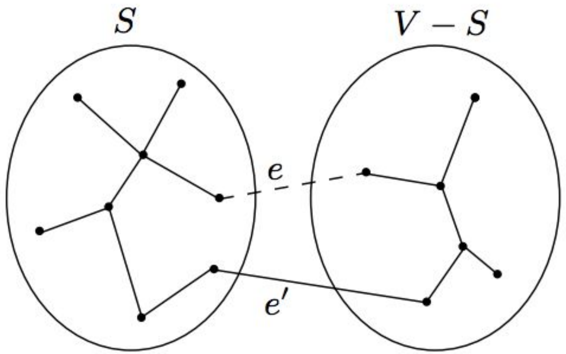
Kruskal's Algorithm

- A greedy approach to constructing an MST
- Repeatedly adds the next lightest edge that doesn't produce a cycle
- Correctness guaranteed by the **cut property**

- Implemented using disjoint sets (use union-find to check for cycles)
- Runtime: O(|E|log*|V|)

The Cut Property

- Let X be an MST in progress.
- Pick any subset of nodes S for which the edges of X do not cross between S and V - S.
- The lightest edge across this partition, joined with X, will be part of some MST.



so it is always safe to add the lightest edge across any such cut!

Prim's Algorithm

- Greedily adds the cheapest edge across the cut
  - cut: set of nodes inside the MST vs. set of nodes outside the MST
- Pseudocode:
  - $S, X = \{s\}, \varnothing$
  - repeat $|V| - 1$ times:
    - pick cheapest edge $e = (u, v)$ across cut $S, V - S$
    - $S, X = S \cup \{v\}, X \cup \{e\}$
- Uses the cut property $|V| - 1$ times, alters cut so the same edge is never picked twice
- Can do this quickly by modifying Dijkstra's code
- Runtime: $O((|V| + |E|)\log|V|)$

slide credit: Nick Titterton

Huffman Encoding

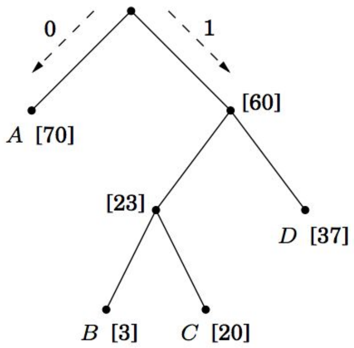
- We have a **string**, an **alphabet**, and a **frequency of occurrence for each symbol**
- We want to encode the string (losslessly) into a minimum-size bit representation
- Intuition: more frequently-occurring symbols should use less bits
  - (therefore, each symbol will use a different number of bits)
- For decoding to be possible, we'll also need our codewords to be **prefix-free**
  - no codeword should be a prefix of another codeword

- To encode: simply go symbol by symbol in the string and substitute with respective codewords
- To decode: scan the bitstring, and every time we hit a full codeword replace it with the symbol

Huffman Encoding, cont.

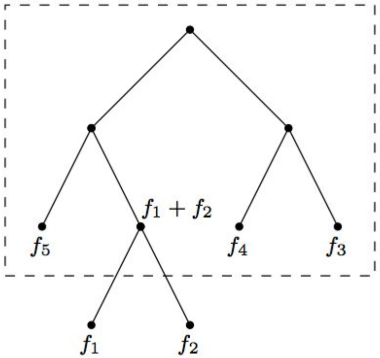
- A prefix-free encoding can be represented as a full binary tree
  - “full”: each node must have either zero or two children
- Our goal in Huffman encoding will be to identify the optimal coding tree

Symbol	Codeword
A	0
B	100
C	101
D	11



in a coding tree, symbols are at the leaves, and each codeword is generated by a path from root to leaf (interpreting left as 0 and right as 1)

Huffman Encoding, cont.



- The optimal coding tree minimizes

$$\sum_{i \in \text{symbols}} (\text{frequency of } i) \cdot (\text{depth of } i \text{ in the tree})$$

- Note: the frequency of an internal node is the sum of the frequencies of its descendant leaves
- In Huffman, we continually **merge the two nodes with the smallest frequencies**
  - this will ensure that these nodes end up with the greatest depth, and hence the longest codewords
- Runtime: O(nlogn)
  - implementation uses a priority queue

Horn Formulas

- A Horn formula is a logical expression composed of implications and negative clauses.
  - **Implications**
    - (AND of positive literals) ⇒ positive literal
    - “if the conditions on the left hold, then the RHS must be true”
  - **Negative clauses**
    - OR of negative literals
    - “they can’t all be true”

$$(w \wedge y \wedge z) \Rightarrow x, \;\; (x \wedge z) \Rightarrow w, \;\; x \Rightarrow y, \;\; \Rightarrow x, \;\; (x \wedge y) \Rightarrow w, \;\; (\overline{w} \vee \overline{x} \vee \overline{y}), \;\; (\overline{z})$$

Horn Formulas, cont.

- Goal: find a satisfying assignment (an assignment of true/false to the boolean variables that satisfies all the clauses)
- A greedy algorithm:
  - Start with everything false
  - For every unsatisfied implication, set the RHS literal to true
  - Check if all the pure negative clauses are satisfied (if so, we've found a satisfying assignment)

$(w \wedge y \wedge z) \Rightarrow x,\ \ (x \wedge z) \Rightarrow w,\ \ x \Rightarrow y,\ \ \Rightarrow x,\ \ (x \wedge y) \Rightarrow w,\ \ (\neg w \vee \neg x \vee \neg y),\ \ (\neg z)$

$(w \wedge y \wedge z) \Rightarrow x,\ \ (x \wedge z) \Rightarrow w,\ \ x \Rightarrow y,\ \ \Rightarrow x,\ \ (x \wedge y) \Rightarrow w,\ \ (\neg w \vee \neg x \vee \neg y),\ \ (\neg z)$

$(w \wedge y \wedge z) \Rightarrow x,\ \ (x \wedge z) \Rightarrow w,\ \ x \Rightarrow y,\ \ \Rightarrow x,\ \ (x \wedge y) \Rightarrow w,\ \ (\neg w \vee \neg x \vee \neg y),\ \ (\neg z)$

$(w \wedge y \wedge z) \Rightarrow x,\ \ (x \wedge z) \Rightarrow w,\ \ x \Rightarrow y,\ \ \Rightarrow x,\ \ (x \wedge y) \Rightarrow w,\ \ (\neg w \vee \neg x \vee \neg y),\ \ (\neg z)$

no satisfying assignment!