- Huffman tree
  - Prefix encoding vs. binary tree

Claim: if all characters are leaf nodes in a tree, then the encoding is prefix free (and vice versa)

Claim: all prefix-free encodings allow unique decoding.

Decoding: (input: binary string of length)
- Start from root of the tree
- For i = 1 to l
  - Follow the edge labeled by a[i]
  - If we have reached a leaf
    - Output the corresponding character
    - Return to root

- Construct a tree by making a sequence of decisions
- Initial: n nodes that are leaves
- Every step: Pick 2 nodes and "merge" them

- Cost of merging: total frequency of the two merged nodes

- Running time
  - Naive implementation
naive implementation

\( n - 1 \) iteration (every iteration reduces \#char. by 1)

\( O(n) \) for each iteration

\( O(n^2) \)

use priority queue / heap

- support: finding min element, add, delete \( O(\log n) \)

\( O(n \log n) \)

- Proof of correctness:

we use induction.

Induction Hypothesis: Huffman Tree algorithm finds an optimal encoding for all alphabets of size at most \( N \).

Base Case: when \( N = 1 \), there is only one solution with cost 0.

Induction Step: Assume IH is true for \( N \), consider an alphabet of size \( N + 1 \).

assume towards contradiction that Huffman Tree algorithm does not find the optimal solution. Let \( T_{ALG} \) be the tree found by algorithm

\( T_{OPT} \) be the tree found by OPT, and \( i, j \) be the first two characters that the algorithm merged.

if \( i, j \) are not children of the same node in \( T_{OPT} \)

let \( i', j' \) be two nodes at the highest depth in \( T_{OPT} \) that share the same parent (note: one of \( i', j' \) may overlap with one of \( i, j \)).

let \( T'_{OPT} \) be a solution where \( i, j \) are swapped with \( i', j' \) in \( T_{OPT} \).

let \( d_i \) be depth of \( i \) in \( T_{OPT} \) (similarly for \( d_j, d_i', d_j' \)); we have

\[
\text{Cost}(T'_{OPT}) = \text{Cost}(T_{OPT}) - (W_i \cdot d_i + W_j \cdot d_j + W_i \cdot d_i' + W_j \cdot d_j' + W_i \cdot d_i + W_j \cdot d_j)
\]

\[
= \text{Cost}(T_{OPT}) - (W_i - W_i)(d_i - d_i') - (W_j - W_j)(d_j - d_j')
\]

\[
\leq \text{Cost}(T_{OPT})
\]

here the last inequality is because \( W_i < W_i', W_j < W_j' \) (ALG has chosen two characters with lowest freq.)

\( d_i < d_i', d_j < d_j' \) (both \( i \) and \( j \) have highest depth)

therefore, \( T'_{OPT} \) is also an optimal solution.

now we know there is always an optimal solution that merges \( i \) and \( j \).

the problem reduces to an alphabet of size \( n \).
Now we know there is always an optimal solution. The problem reduces to an alphabet of size \( n \).

By induction hypothesis, Huffman tree algorithm is optimal for this instance. Therefore, \( \text{cost}(T_{\text{acc}}) \leq \text{cost}(T_{\text{opt}}) \leq \text{cost}(T_{\text{opt}}) \), this contradicts with the assumption that \( T_{\text{acc}} \) is not optimal.

Now we know \( T_{\text{acc}} \) is always optimal even for alphabet of size \( n+1 \), this finishes the induction. \( \square \)