🪀

Huffman codes

📘 Huffman Coding — Exam Notes

1. What is Huffman Coding?

- A **compression algorithm** that assigns **variable-length, prefix-free codes** to symbols based on their frequencies.
- Frequently used in real-world compression (e.g., `zip`, `gzip`, `PNG`).lecture12_huffman
- Idea: more frequent symbols → shorter codes; rare symbols → longer codes.
- Reduces total bits needed compared to fixed-length encoding.

2. What is it used for?

- **Data compression**: reduces file size without losing information.
- Core in many **lossless compression formats** (text, images, executables).
- Ensures efficient storage and transmission of data.

3. How does it work?

1. Count frequency of each symbol in the data.
2. Build a forest of nodes (one node per symbol, weighted by frequency).
3. Repeat until one tree remains:
    - Pick the **two lowest-frequency nodes**.
    - Merge them into a new parent node with combined weight.
    - Insert back into the forest.

4. Final tree = Huffman tree.
    - Left edge = bit `0`, right edge = bit `1`.
    - Codeword for a symbol = path from root to its leaf.

Key property: The resulting codes are **prefix-free** (no codeword is a prefix of another), so concatenated codes can be uniquely decoded.

4. Why is it correct?

- **Optimal substructure:**
    - If `x` and `y` are the least frequent symbols, in *some* optimal code they must be siblings at maximum depth.
- **Greedy choice property:**
    - Always merging the two least frequent symbols is safe, because if they weren't deepest siblings already, swapping them would only improve efficiency.
- **Proof sketch (exchange argument):**
    - Replace `x` and `y` by new symbol `z = f(x) + f(y)`.
    - Solve smaller problem optimally.
    - Expand back into `x` and `y`.
    - If there were a better solution, it would contradict the subproblem's optimality.
- Therefore, Huffman's greedy construction yields an **optimal prefix-free code**lecture12_huffman.

5. Is it a greedy algorithm?

✅ Yes — Huffman coding is one of the most famous **greedy algorithms**.

- **Greedy step:** Always merge the two least frequent symbols.
- **Why greedy works here:**

    - Locally optimal decision (merging the least frequent pair) leads to globally optimal result.
    - Exchange argument + optimal substructure guarantee correctness.

6. Complexity of Huffman Coding

- Build priority queue (min-heap) of frequencies: O(N)O(N)O(N).
- Repeatedly extract 2 smallest + reinsert merged node, N−1N-1N−1 times.
- Total: O(NlogN)O(N \log N)O(NlogN).