Design and Analysis of Algorithms
Greedy Technique

Optimal Encoding with Huffman's Algorithm

Dr. Soharab Hossain Shaikh
BML Munjal University

1

Greedy Technique

Computer scientists consider ***greedy*** approach a general design technique despite the fact that it is applicable to optimization problems only. The greedy approach suggests constructing a solution through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached. On each step—and this is the central point of this technique—the choice made must be:

- *feasible*, i.e., it has to satisfy the problem's constraints
- *locally optimal*, i.e., it has to be the best local choice among all feasible choices available on that step
- *irrevocable*, i.e., once made, it cannot be changed on subsequent steps of the algorithm

2

File Compression

- Given a file which is represented as a string of characters, the file ***compression*** problem is to compress the file as much as possible in such a way that the original file can be easily reconstructed.
- Each character is represented using a sequence of bits called the *encoding* of the character.
- There exist two types of encodings:
  - *Fixed-length* encoding: Each character is represented using the same number of bits.
    - The size of the file would only depend on the number of characters in the file.
  - *Variable-length* encoding: Characters with large frequencies are assigned short encodings, whereas characters with small frequencies are assigned long encodings.
    - Let $C = \{c_1, c_2, ..., c_n\}$. Then, $\forall c \in C$, $f(c)$ represents the frequency of character c in the file

3

File Compression (cont…)

- **Variable-Length Encoding Properties**
  - Prefix Codes: The encoding of one character must not be the prefix of the encoding of another character.
    - For example, if we assign the encodings 11 and 110 to the letters "*a*" and "*b*", there will be an ambiguity as to whether 11 is the encoding of "*a*" or is the prefix of the encoding of the letter "*b*".
  - Decoding Process: Consists of scanning a sequence of bits until an encoding of some character is found. This process is repeated until all the sequence has been scanned.

4

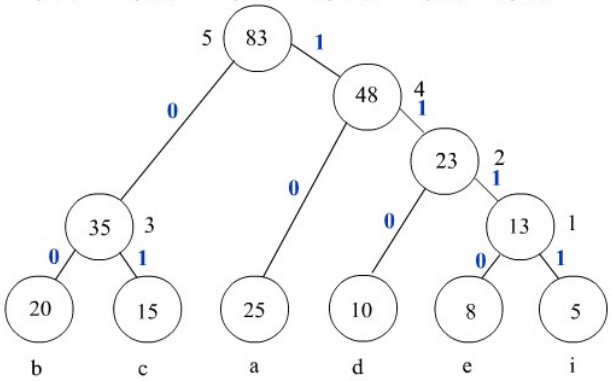
Huffman Algorithm

- **Basic Idea of Huffman Algorithm**
  - Full Binary Tree Representation
    - Each internal node has exactly two branches labeled by 0 and 1
    - A sequence of 0's and 1's on a path from the root to a leaf corresponds to a character encoding.
    - The leaves in this tree correspond to the characters $\{c_1, c_2, ..., c_n\}$.
  - Greedy Approach to File Compression: Let F be a forest consisting of n roots, each corresponding to a character $c \in C$, where the frequency $f(c)$ is stored in its corresponding root.The algorithm repeats the following steps until F contains only one tree
    - Select two roots r_i and r_j with minimum frequencies.
    - Create a new node v whose frequency is the sum of the frequencies of r_i and r_j, and make r_i and r_j the children of v.
  - Note here that the code constructed by this algorithm satisfies the prefix constraint and minimizes the size of the compressed file as well.

5

Huffman Example

- Example:
  - Suppose that the letters a, b, c, d, e and i appear in a file with the following frequencies: $f(a) = 25, f(b) = 20, f(c) = 15, f(d) = 10, f(e) = 8, f(i) = 5$.



6

Comparison

$$Size = \sum_{c \in C} \#\text{bits}(c) \,.\, f(c)$$

- 3-bit Fixed Codes: $3(25+20+15+10+8+5) = 249$ bits.
- Huffman: $2(25+20+15)+3(10)+4(8+5) = 202$ bits.
- Saving $(249-202)/249 \approx 18.88\%$

7

Huffman Algorithm

Algorithm HUFFMAN

Input A set $C = \{c_1, c_2, ..., c_n\}$. of n characters and their frequencies $f(c_i), 1 \le i \le n$.

Output A Huffman tree F.

1. Insert all characters into a min-heap H according to their frequencies.
2. Initially F is a forest of n roots, each representing a character $c \in C$;
3. **for** $k \leftarrow 1$ to n - 1
4. $r_i \leftarrow$DELETEMIN(H)
5. $r_j \leftarrow$DELETEMIN(H)
6. $f(v) \leftarrow f(r_i) + f(r_j)$ {v is a new node}
7. INSERT(H, v)
8. *Make* r_i *and* r_j children of v in F
9. **end for**

8

Huffman Algorithm Complexity

Algorithm HUFFMAN

Input: A set $C = \{c_1, c_2, ..., c_n\}$. of n characters and their frequencies $f(c_i), 1 \le i \le n$.

Output: A Huffman tree F.

1. Insert all characters into a min-heap H according to their frequencies.
2. Initially F is a forest of n roots, each representing a character $c \in C$;
3. **for** $k \leftarrow 1$ to n - 1
4. $r_i \leftarrow$DELETEMIN(H)
5. $r_j \leftarrow$DELETEMIN(H)
6. $f(v) \leftarrow f(r_i) + f(r_j)$ {v is a new node}
7. INSERT(H, v)
8. *Make* r_i *and* r_j children of v in F
9. **end for**

- **Step 1 takes $\Theta(n)$.**
- **Steps 4 – 8 cost $O(\log n)$.**
- **The overall time taken by the for loop is $O(n \log n)$.**
- **Thus, the time complexity of the algorithm is $O(n \log n)$.**

9

End of Lecture

10