Lecture 8.1 & 8.2

- Anurag Sharma & Shymal Chandra

Greedy Algorithms

Greedy Uncle Scrooge

Greedy Algorithms

- Greedy algorithms, like dynamic programming are often used to solve optimization problems.
- A greedy approach arrives at the solution by making a sequence of decisions, each of which simply looks like the best decision at that moment.
- Each choice is the locally optimal choice, hoping to arrive at a globally optimal solution.
- The final solution however may not always be the optimal with the greedy approach.

Cont.

- Each iteration in the greedy algorithm consists of the following steps:

1. A selection procedure – chooses the next item according to some greedy criterion
2. A feasibility check – determines whether the decision is locally optimal, i.e. whether the chosen item can lead to a solution.
3. A solution check – determines whether the solution is reached or not.

Examples

- Problem: To minimize the number of coins while giving some change
- Example 1: From available coins 50c, 20c, 10c, 5c, 2c, 1c, give a change of 75c

1. Selection – pick coin with highest value
2. Feasibility – check if change value ≤ 75c
3. Solution – check if change value reached
   Greedy Solution: 50c + 20c + 5c (optimal)

Cont.

- Example 2: Assuming you only have coins 12c, 10c, 5c, 1c, how would you give a change of 16c?
- Greedy Solution: 12c + 1c + 1c + 1c + 1c (not optimal)
- Optimal solution would have been 10c + 5c + 1c

Huffman Code

- Huffman coding is an efficient method of compressing data without losing information.
- It uses a particular type of optimal prefix code for data compression.

Data Compression

- Even though capacity of secondary storage devices keeps getting larger and their cost keeps getting smaller, the devices continue to fill up due to increased storage demands.
- Thus, given a data file, it would be desirable to find a way to store the file as efficiently as possible.
- The problem of data compression is to find an efficient method for encoding a data file.

Encoding Data

- A common way to represent a file is to use a binary code – each character represented by a unique binary string, called the **codeword**
- A fixed-length binary code represents each character using the same number of bits
- Example: Using code A: 00, B: 01, C: 11 and given a file ABABCBBBC, the encoding is 00010001110101011
- Can you obtain more efficient encoding?

Cont.

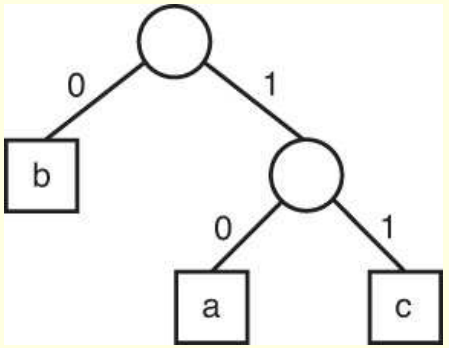
- A variable-length binary code can represent different characters using different numbers of bits
- Example: Using code A: 10, B: 0, C: 11 and given a file ABABCBBBC, the encoding is 1001001100011
- This encoding uses lesser bits than the previous

Prefix Codes

- A particular type of variable-length code is a prefix code.
- In prefix code, no codeword for one character constitutes the beginning of the codeword for another character, e.g. If codeword for A is 01, then another character B cannot have codeword as 011
- Every prefix code can be represented by a binary tree whose leaves are the characters to be encoded

Cont.

- Binary Tree for code A: 10, B: 0, C: 11



Parsing prefix codes

- Start at the first bit on ~~the left in the file and~~ the root of the tree, sequence through the bits, and go left or right down the tree depending on whether a 0 or 1 is encountered
- When a leaf is reached, obtain the character at that leaf.
- Return to the root and repeat the procedure starting with the next bit (or branch) ~~in sequence in the file~~

Generating prefix code

- Huffman developed a greedy algorithm that produces an optimal binary character code by constructing a binary tree corresponding to an optimal code
- A code produced by this algorithm is called a Huffman code

Huffman's Algorithm

- The basic approach is to get the frequency of each character used in a given a text file, and store these in a priority queue.
- In a priority queue, the element with the highest priority is the character with the lowest frequency in the file.
- The priority queue can be implemented as a linked list, but more efficiently as a heap.
-

Character	Frequency
A	15
B	5
C	12
D	17
E	10
F	25

Cont.

n = number of characters in the file;

Arrange n pointers to nodetype records in a priority queue PQ as follows: For each pointer p in PQ

$p\text{->}symbol$ = a distinct character in the file;
$p\text{->}frequency$ = the frequency of that character in the file;
$p\text{->}left = p\text{->}right$ = NULL;

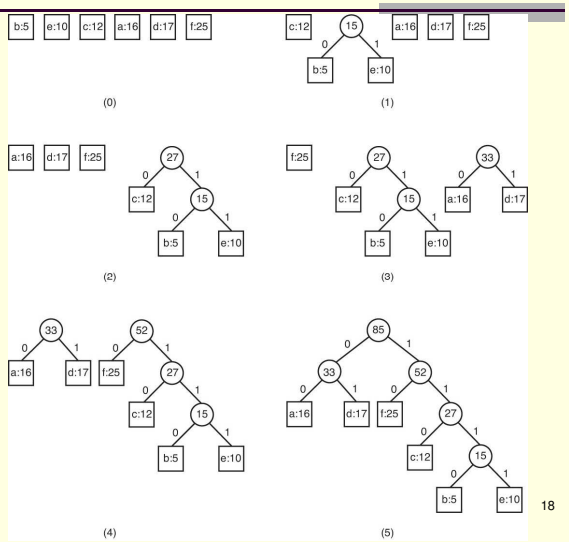
```
for (i=1;i<=n-1;i++) {  // There is no solution check; rather,
   remove(PQ,p);        // solution is obtained when i = n - 1.
   remove(PQ,q);        // Selection procedure.
   r = new nodetype;    // There is no feasibility check.
   r->left = p;
   r->right = q;
   r->frequency = p->frequency + q->frequency;
   insert(PQ,r);
}
remove(PQ,r);
return r;
```

Cont.

- Every iteration takes the two most priority items and join them together in left and right branch.
- The aim is to have lesser bits for most common characters and more bits for less common characters so as to minimize total number of bits.

Huffman tree construction

Character	Frequency
A	15
B	5
C	12
D	17
E	10
F	25



Final verdict

- Finally, parsing through the tree would result in the following optimal codes for each character:

Character	Frequency	Code
b	5	1110
e	10	1111
c	12	110
a	16	00
d	17	01
f	25	10

- Why not just 3 digits for each character? Try it!