week4_lec7

Continuation of Greedy Algorithms

Greedy Techniques are best suited to finding a solution for the immediate situation

- Greedy Choice Property : This property states that the globally optimal solution can be obtained by making a locally optimal solution which gives us the greedy approach. It is determined and impacted by prior decisions, not by future decisions.
- Optimal Substructure: The main goal here is to solve subproblems and build up solutions to solve larger problems.

  Structure → Optimal solution to problem += Optimal solution to subproblems

Main Ideas

- Activity Selection
- Huffman Codes

Activity Selection

Problem : There is a set of activities S = $a_1, a_2, ..., a_n$ and each activity a_i needs a resource during the period $[s_i, f_i)$ which is a half open interval that is specified by a start time and a finish time. Select the maximal number of activities from S that do not intersect times.

The greedy choice: The activity with the least finish time (say a_0 belongs to some optimum solution).

Proof: Let A be a maximum size subset of mutually compatible activities from S. We then order the activities in monotonically increasing order of the finish times. Let a_k be the first activity in A.

If

$a_k = a_0$: done.

else

construct $B = A - ([a_k] \cup [a_0])$

Huffman Codes

The Huffman Coding Algorithm's main concept is to employ fewer bits for more frequently occurring characters. It uses variable length codes to compress data storage.

To read a single character from a file, the system usually reads 8 bits at a time. This is a waste of time. One contributing reason for this is that certain characters are used more frequently than others. The simple solution is to use longer binary codes for characters and groupings of characters that are less common.

Note : Using Variable length encoding means that no codeword can be a prefix of another codeword i.e. one word cannot start with how another ends.

Problem : Given the frequencies $f_1, f_2, ..., f_n$ of n symbols, we want a tree whose

Greedy Approach : The ideal tree must have the two symbols with the fewest frequencies at the bottom. They act as the lowest internal node's offspring. We back up this claim by stating that we replace the two symbols with the lowest value, and the encoding definitely improves.

Optimum Substructure property: The cost of the tree is sum of the frequencies of all leaves and internal nodes apart from the root of the tree.

Example :

Character	Frequency
a	12
b	2
c	7
d	13
e	14
f	85

As a result, we generate a binary tree for each character that additionally retains its frequency of occurrence. Find the two binary trees with the lowest frequencies at their

nodes in the list. Each leaf node corresponds to a letter in the code when you build the tree. We go from the root to the leaf node to find the code.

1. For each move to the left, append 0 to the code.
2. For each move to the right, append 1 to the code.

$$\text{cost of tree} = \sum_{i=1}^{n} f_i \cdot (\text{depth of } i\text{th symbol in tree})$$

Character	Code
a	001
b	0000
c	0001
d	010
e	011
f	1

Now, we try to see and understand how many bits we saved to appreciate the algorithm.

- First, calculate the number of bits originally used to store the data and subtract from that the number of bits used to store data using the algorithm.
  Since we have 6 characters assume each character to be stored using 3-bit code.
  Since, there are 133 characters, we multiply total frequencies by 3 and get total bits 3 * 133 = 399.
- using Hauffman

Character	Code	Frequency	Total Bits
a	001	12	36
b	0000	2	8
c	0001	7	28
d	010	13	39
e	011	14	42
f	1	85	85

- TOTAL: 238
- Saved = 399 - 238 = 161 which is about 40% of the storage space.

Algorithm for Hauffman Coding

```
Build a priority queue H out of the frequencies
for k = n+1 to 2n-1:
    i = deleteMin(H)
    j = deleteMin(H)
    Create a node k with children i,j
    f[k] = f[i] + f[j]
    insert(H,k)
```

Entropy

It relates to the information in the system and how to optimum encode it. We realize the relation:

more compressible = less random = more predictable

Suppose there are n possible outcomes, with probabilities $p_1, p_2, ..., p_n$. Assume that $p_i's$ are all of the form $(1/2)^k$ and are observed frequencies.

Average number of bits needed to encode a single draw from the distribution is:

$$\sum_{i=1}^{n} p_i \log \frac{1}{p_i}$$

The most uniform distributions minimize this value. This is called the entropy of the distribution.