Dijkstra's Algorithm and Huffman Codes

COMP 215 Lecture 7

Dijkstra's Algorithm

- Floyd's algorithm was an order n^3 algorithm for finding all pairs of shortest paths.
- Overkill if we just need a shortest path from one particular node.
- The alternative is Dijkstra's algorithm.
- It is very very similar to Prim's algorithm.
- Let's do an example...

Dijkstra's Pseudocode

- Assume that we are finding paths from vertex v_1.
- Call the set of vertices already considered *Y*.
- We will maintain two arrays,
  - `touch[i]` = index of the vertex v in Y such that (v, v_i) is the last edge on the current shortest path from v_1 to v_i.
  - `length[i]` = length of the current shortest path from v_1 to v_i.

Dijkstra's Pseudocode

```
Void dijkstra(int n, number W[][], set_of_edges& F)
{
   index vnear;
   number min;
   edge e;
   index touch[2..n];
   number length[2..n];
   F = EMPTYSET;
   for (i = 2; i<=n i++) {
      touch[i] = 1;
      length[i] = W[1][i];
   }
      CONTINUED ON NEXT SLIDE...
```

```
    repeat (n - 1 times) {
       min = ∞;
       for (i = 2; i <=n; i++) {
          if (0 <= length[i] < min) {
             min = length[i];
             vnear = i;
          }
       }

       e = (touch[vnear],vnear);
       add e to F;

       for (i = 2; i<=n; i++) {
          if (length[vnear] + W[vnear][i] < length[i]) {
             length[i] = length[vnear] + W[vnear][i];
             touch[i] = vnear;
          }
       }
       length[vnear] = -1;
    }
}
```

Dijkstra Analysis

- Time complexity?
- What will *F* contain at the end of the algorithm?
- How do we retrieve the actual path?
- What if we wanted to know the length of the shortest paths?

Coding Theory

- Here are the 7-bit binary representations of the ascii codes for “a” and “q”.
  - “a” - `1100001`
  - “q” - `1110001`
- Why 7 bits? We would be happier with a 3 bit code.

Coding Theory

- OK, we couldn't represent everything we want to represent.
- Why not use a 3 bit code (or 2 or 1) for “a” and a longer code for “q”?

Prefix Codes

- No codeword for one character is the beginning of a codeword for another.
- If `00` is the codeword for “a”, `001` can't be the codeword for “b”.
- Allows us to decode with a simple left to right scan.
- Prefix codes can be represented as binary trees.
- Let's see an example...

The Problem...

- Find the prefix code (tree) that gives the shortest encoding of a given string.
- Notice that the number of bits used by a given binary tree is equal to:

$$\sum_{i=1}^{n} frequency(v_i)\, depth(v_i)$$

- So, we are looking for the tree that minimizes this.
- The solution is Huffman codes.
- Let's see how they work...

Huffman Pseudocode

- We need a node type to build our tree from:

```
struct nodetype
{
   char symbol;
   int frequency;

   nodetype* left;
   nodetype* right;
}
```

- The algorithm starts by creating one (leaf) node for each symbol.
- Those nodes are inserted into a heap.
- Lower frequency=higher priority.

Huffman Pseudocode

- We can then build the tree as follows:

```
nodetype* huffman(int n, priorityQueue PQ)
{
   nodetype *p, *q, *r;
   for (int i = 1; i <= n; i++) {
      remove(PQ,p);
      remove(PQ,q);
      r = new nodetype;
      r->left = p;
      r->right = q;
      r->frequency = p->frequency + q->frequency;
      insert(PQ,r);
   }
   remove(PQ,r);
   return r;
}
```

Huffman Analysis

- Initializing a heap requires $\Theta(n)$ time.
- Individual heap operations require $\Theta(\lg n)$ time.
- Correctness proof.