1 Huffman codes CLRS §16.3/ KT §4.8

Recall from last time:

1.1 Intro to compression

Huffman coding is probably the most common compression technique today. E.g. used in: zip files, mp3 files, common formats of image files (specifically in cases where we don't want to lose any data).

1.2 Prefix codes

Recall: Prefix codes allow for unambiguous parsing when we have variable length codewords.

Definition A **prefix code** for a set S is a function

$$\gamma : S \to \{0,1\}^*$$

such that for all $x, y \in S$, $x \neq y$, $\gamma(x)$ is not a prefix of $\gamma(y)$.

Definition The **average bits per letter** of a prefix code γ is

$$\mathbf{ABL}(\gamma) = \sum_{x \in S} f_x \, |\gamma(x)|$$

Goal: given alphabet S and frequencies f, we want to find an optimal prefix code γ; i.e. minimize $ABL(\gamma)$.

We can use binary trees to represent prefix codes!

Example

ABL(T) =

Can this encoding be made more efficient?

Definition A binary tree is **full** if

Claim. The binary tree corresponding to an optimal prefix code is full.

Proof.

1.3 Greedy attempt 1: Shannon-Fano 1949

Create tree top-down, splitting S into

$$f_a = 0.32, \quad f_e = 0.25, \quad f_k = 0.2, \quad f_r = 0.18, \quad f_u = 0.05$$

1.4 Greedy attempt 2: Huffman encoding 1952

- *Observation 1.* Lowest frequency symbols should be
- *Observation 2.* The lowest level always contains
- *Observation 3.* The order in which symbols appear

Claim 1. There is an optimal prefix code with tree T^* where

Create tree bottom-up.

Example

$$f_a = 0.32, \quad f_e = 0.25, \quad f_k = 0.2, \quad f_r = 0.18, \quad f_u = 0.05$$

1.5 Algorithm

```
 if |S| = 2:
     return
Let y and z be
S' =
Remove y and z from
Insert new
T' =
T =
Return
```

Time complexity

- Naive implementation
- Use priority queue to store symbols

1.6 Proof of correctness/optimal

Claim 2. ABL(T) =

Proof.

Claim 3. The Huffman code achieves the minimum ABL of any prefix codes.

Proof.

2 Graph Algorithms

2.1 Intro - Examples, notation, applications

Graph search/graph exploration problems

Example applications. Google pagerank, Facebook social graphs, Google maps, internet routing, biological network (protein-protein interactions), puzzles and games

Example

Representing graphs

2.2 Breadth-First Search (BFS) (CLRS §22.2)

A motivating example for BFS: Pocket cube ($2 \times 2 \times 2$ Rubik's cube)

Example

BFS

Input:

Output:

Idea:

Example

```
level = { s:0 }
parent = { s:None }
i = 1
frontier = [s]
while frontier not empty:
    next = [ ]
        for u in
            for v in
                if v not in
                    level[v] =
                    parent[v] =
                    next.append
        frontier =
            i
```

Runtime

Remarks.

- For all v, the path

- The set of vertices we explored

Example

2.3 Depth-First Search (DFS) (CLRS §22.3)

Idea:

Example

```
u.color =
for v in
    if v.color ==
        v.parent
        DFS-VISIT
u.color =
time
u.f
```

We usually use DFS to learn something about the graph (as opposed to a vertex as in BFS). We therefore usually use:

```
for u in
    u.color =
    u.parent =
for u in
    if u.color ==
        DFS-VISIT
```

Runtime.

2.3.1 Edge Classification

Definition

- **Tree edge**

- **Forward edge**

- **Backward edge**

- **Cross edge**

How can we detect what type of edge (u, v) *is?*

- Tree edge:
- Forward edge:
- Backward edge:
- Cross edge:

To distinguish forward from cross edge,

- In forward edge
- In cross edge

Example

Theorem 22.10 In a DFS of an undirected graph G, every edge is either

In other words,

Proof.

Corollary. An undirected graph is acyclic if and only if

Proof.

What about directed graphs?

Theorem 22.9 (white-path theorem) The vertex v is a descendant of u in the DFS forest if and only if

Proof idea.

Lemma 22.11 A directed graph is acyclic if and only if

Proof.

3 Topological Sort (CLRS §22.4)

Tasks to be performed where some tasks must be completed before others. E.g. prerequisites for courses. Input is represent as

Example

Algorithm.
Input:

1. Run

2. Output

Runtime.

Theorem. The given algorithm outputs

Proof.