

Week 7 Day 2 Lecture Notes

Prep:

- Implementation 2 due tonight @midnight.
- 1 Day Grace period with no point reduction.
- Read DPV Chapter 5.2

Minimum Spanning Tree and Hoffman Coding.

- Review Prims and Kruskal's Algorithm. Every edge can be constructed with a cut that the edge is cheapest across that cut. Used to prove correctness.

Prims Algorithm:

Prims algorithm Review: It randomly picks a node to start with.

Prims Algorithm Pseudocode:

More efficient implementation

procedure prim(G, w)

Input: A connected undirected graph $G = (V, E)$ with edge weights w_e

Output: A minimum spanning tree defined by the array prev

for all $u \in V$:

$\text{cost}(u) = \infty$

$\text{prev}(u) = \text{nil}$

Pick any initial node u_0

$\text{cost}(u_0) = 0$

$H = \text{makequeue}(V)$ (priority queue, using cost-values as keys)

while H is not empty:

$v = \text{deletemin}(H)$

$S = S \cup \{v\}, T = T \cup (\text{prev}(v), v)$

 for each $\{v, z\} \in E$:

 if $\text{cost}(z) > w(v, z)$:

$\text{cost}(z) = w(v, z)$

$\text{prev}(z) = v$

$\text{decreasekey}(H, z)$

Maintaining the costs for attaching z to S

Analysis: $O((|V| + |E|) \log |V|)$ - assuming using binary heap for priority queue

- each node is inserted and deleted once from the priority queue ($O(|V| \log |V|)$)
- Each edge is checked once, leading one possible decreasekey ($O(|E| \log |V|)$)

Can we improve the runtime?

Binary Priority queue. stores the cost of reaching every node from the original node u

Kruskal's Algorithm: Sorts edges in increasing weight order.

- Adds the cheapest edges one at a time (as long as the edge doesn't create a cycle)
- To accomplish this we check if there is already a path between the nodes that the edge we are checking connects.
- Runtime: Sorting time + cycle through all the edges. Simplified as $O(|E|^2)$

Union-By-Rank Efficient Kruskal:

Efficient Implementation of Kruskal's

- Use Union-by-rank data structure to maintain disjoint sets
- Each set contains the nodes of a particular connected component
- Initially each node is in a component by itself

procedure kruskal(G, w)

Input: A connected undirected graph $G = (V, E)$ with edge weights w
Output: A minimum spanning tree defined by the edges X

for all $u \in V$:

makeset(u) \leftarrow Place every node in its own connected component. $O(|V|)$

$X = \{\}$

Sort the edges E by weight $\leftarrow |E| \log |E|$

for all edges $\{u, v\} \in E$, in increasing order of weight:

if $\text{find}(u) \neq \text{find}(v)$: \leftarrow If u and v are not in the same connected component $O(\log |V|)$

add edge $\{u, v\}$ to X

union(u, v) \leftarrow Merge the two connected components of u and v . $O(C)$

- Total running time:

$$\underbrace{O|V|}_{\text{makeset.}} + \underbrace{O(|E| \log |E|)}_{\text{Sorting}} + \underbrace{O(|E| \log |V|)}_{\text{For loop.}}$$

41

- Compared to the original we maintain a tree structure.

- Maintain disjoint sets.
- **Runtime:** $O(V) + O(|E|\log|E|) + O(|E|\log|V|)$
- **Much faster than E^2 .**

**** The information above is not required material for final review. It's to help enhance your knowledge about the class and Algorithm improvement.****

Huffman Coding

Loss-Less Compression:

- Assume we have 100,000 characters
- How can we encode the data without any loss while storing data in a binary form which is much cheaper than storing the ascii characters?

Fixed Length Code:

Fixed length code

- A simple binary code that use the same number of bits to represent each character
- In our previous example, we have 4 characters
- We can encode each letter use $\log_2 4 = 2$ bits

A	00
B	01
C	10
D	11

- Message: A A B B C D
- Code: 00 00 01 01 10 11
- Code: 11|01|00|10|01|
- Message: D B A C B

- For simple code we can represent 4 characters in a binary bit.
- A = 00, B = 01, C = 10, D = 11.

- Because we have 2 bits for every character we compress the data from original 8 bits down to 2 bits.

Coding Efficiency:

Data sample where the frequency of each amount is different.

- Make higher frequency characters use shorter representation. (Similar to how English words that are used allot are usually very short words)
- **But how can we encode and decode if we have variable bit length?**

Variable length codes

Symbol	Codeword
A	0
B	100
C	101
D	11

- Message: A A B B C D
- Code: 0 0 100 100 101 11
- Code: 11 01 00 10 01 1
- Message: D A B B D

Total	130
A	70 × 1 = 70
B	30 × 3 = 90
C	25 × 3 = 75
D	5 × 2 = 10

- Total # of bits for encoding this sample?

260
245

How do we avoid Ambiguity? The code must be pre-fix Free. (No code could be the prefix of another code)

Example: If we have the code A = 00. B = 001. C = 101 and C = 110.

Since A=0 and B starts with the Prefix of 00.

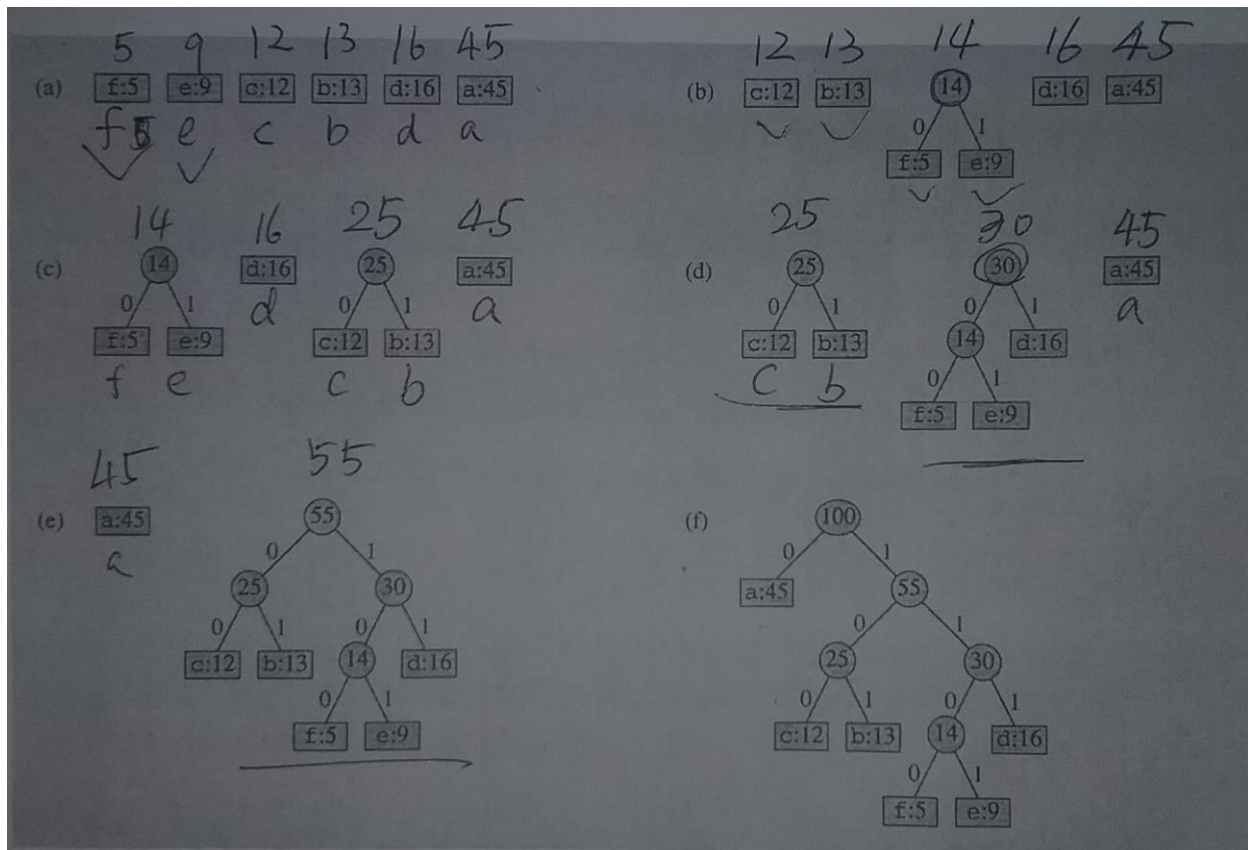
Solve Ambiguity:

****NO LETTER CAN HAVE THE PREFIX OF ANY OTHER LETTER.****

Binary Tree Representation:

- Every Character must be a leaf node.
- If a character is not a leaf node then it would be the pre-fix of another character.
- So every character has to be a leaf node to prevent Prefix Ambiguity.

Decoding is easy with the prefix tree: See picture below for Tree Traversal



Optimal Prefix Coding Problem:

How do we determine the optimal Binary Tree to minimize the average code length?

Optimal Tree Must be a Full Binary Tree:

Full Binary Tree: Every Node (except leaf nodes) have two children.

Think Greedily.

Start by creating leaves for the least-frequent characters.

- See Picture of Tree* showing the most expensive leaves first.
- For each combination add a parent node that contains both.

Ended on Recursive view for solving*

Next time:

- Next Lecture will start with proving the correctness of Huffman's Algorithm
- Implementation 2 Due tonight.
- Continue Reviewing lectures
- Read DPV Chapter 7.1

End of Week 7 Day Notes

~Information composed by Notetaker Scott Russell for CS 325 **DAS** student