CS 314 - Huffman codes + shortest paths

Note Title

2/15/2010

Announcements

Huffman codes

Goal: Transmit a message using as few bits as possible.

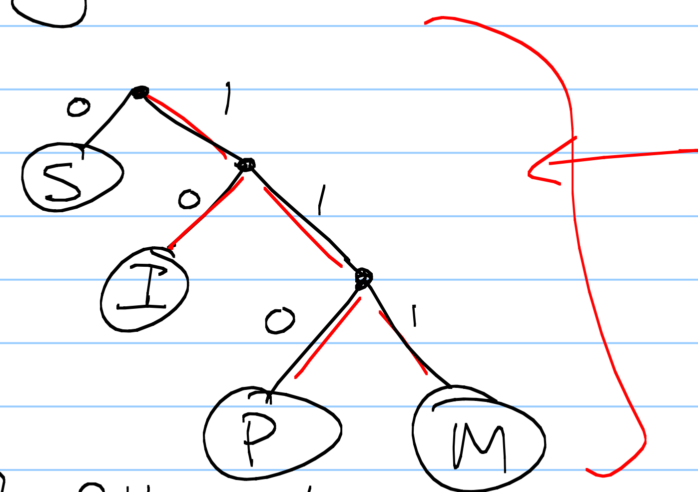
[Use frequency counts (so know the message ahead of time).

Huffman codes: pre-fix free

Why? So we can scan & decode — no ambiguity.

Can visualize as a binary tree

M : 111
I : 10
S : 0
P : 110



When reading string, just follow tree + output letter when you reach a leaf.

Ex: 0 10 1111 10

SIMP

Given n letters plus frequency counts for each letter, $f[1..n]$, find the code minimizing total length:

$$\sum_{i=1}^{n} f[i] \cdot \text{depth}(i) = \text{cost}(T)$$

Pseudocode:

→ Keep characters in min heap, w/ priority = to frequency.

L, R, & P keep track of left/right and parent indices.

```
BuildHuffman(f[1..n]):
  for i ← 1 to n
      L[i] ← 0; R[i] ← 0
      Insert(i, f[i])

  for i ← n to 2n − 1
      x ← ExtractMin()
      y ← ExtractMin()
      f[i] ← f[x] + f[y]
      L[i] ← x; R[i] ← y
      P[x] ← i; P[y] ← i
      Insert(i, f[i])

  P[2n − 1] ← 0
```

$O(n \log n)$

get two least frequent letters

sum frequencies

make them leaves

Proof of Correctness:

Lemma: Let x & y be the 2 least frequent characters. Then there is an optimal code where x & y are siblings and have maximum depth in the tree.

Did proof in class last time.

Thm: Huffman codes are optimal.

pf: Induction on # of letters.

Base case: $n = 1$ (or 2) ✓

IH: Given $< n$ characters, Huffman's alg. is optimal.

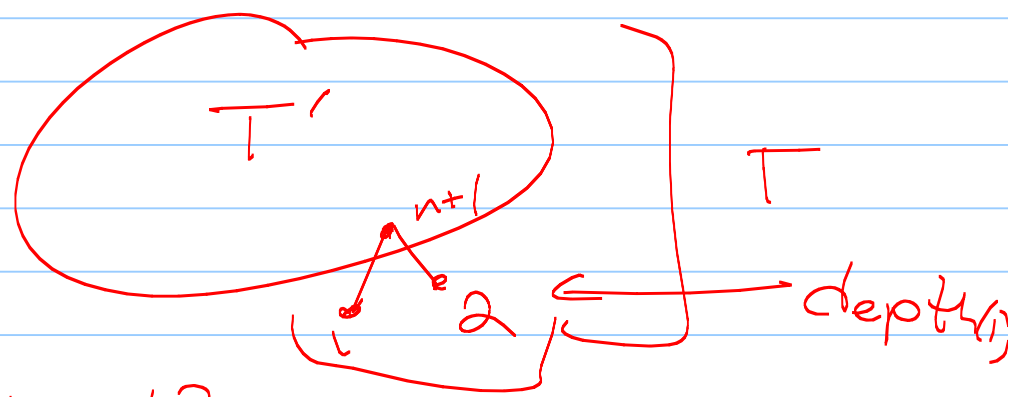
IS: n characters with frequency counts $f[1..n]$
wlog, assume $f[1]$ & $f[2]$ are least frequent.
By our prev. lemma, some optimal tree has 1 & 2 as siblings (at deepest level).

pf continued: Create $f[n+1] = f[1] + f[2]$.

Let T' be Huffman tree for $f[3..n+1]$.

→ By IH, cost(T') is smallest possible for any such tree.

Create T by removing leaf $n+1$ & replacing it w/ internal node w/ 2 children, 1 + 2.



Why is T optimal?

Claim: T is optimal for $f[1..n]$.

$$\mathrm{cost}(T) = \sum_{i=1}^{n} f[i]\cdot \mathrm{depth}(i)$$

$$= \underbrace{\sum_{i=3}^{n+1} f[i]\cdot \mathrm{depth}(i)} - f[n+1]\cdot \mathrm{depth}(n+1)$$

$$+ f[1]\cdot \mathrm{depth}(1) + f[2]\cdot \mathrm{depth}[2]$$

$$= \mathrm{cost}(T') + (f[1]+f[2])\cdot \mathrm{depth}(1)$$

$$- f[n+1]\cdot \underline{\mathrm{depth}(n+1)}$$

(know $f[1]+f[2]=f[n+1]$, and $\mathrm{depth}(1) = \mathrm{depth}(n+1)+1$)

$$= \mathrm{cost}(T') + f[1] + f[2].$$

So $\boxed{cost(T) = cost(T') + f[1] + f[2]}$.

Suppose T was <u>not</u> optimal.

In optimal tree remove 1 & 2 & get a Huffman tree for 3..n+1.

If do that, get a tree for 3..n+1 that is better than T'.

contradiction

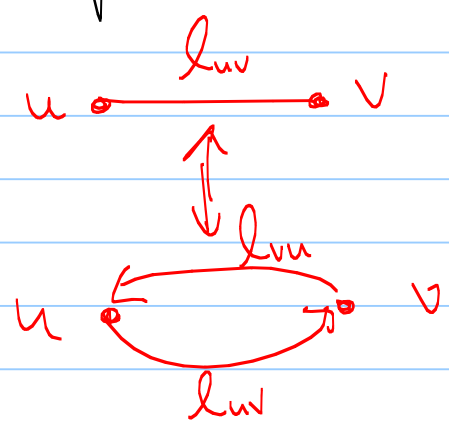
$\blacksquare$

Shortest paths in a graph. (4.4)

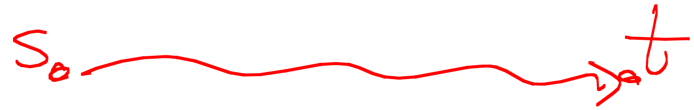
Suppose we have $G = (V, E)$ and each edge $e \in E$ has a length l_e.

Here, we'll assume G is directed: $u \longrightarrow v$.

If given undirected graph, how could we adapt to directed model?

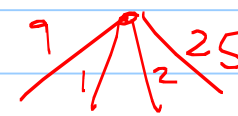


$$O(2|E|) = O(|E|)$$

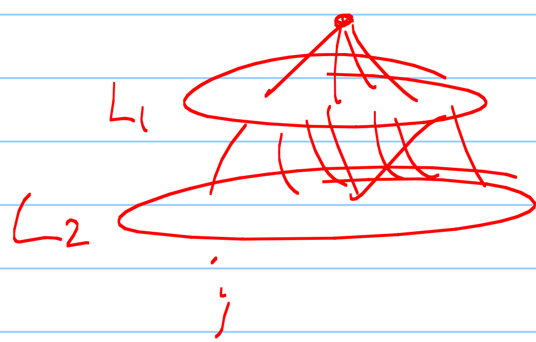


Goal: Given two veritces, find shortest path between them.

Why? Mapquest!



Ideas?



distance 1

BFS - right idea

We'll actually do something harder:

Given a source vertex s, compute shortest path from s to every other vertex.

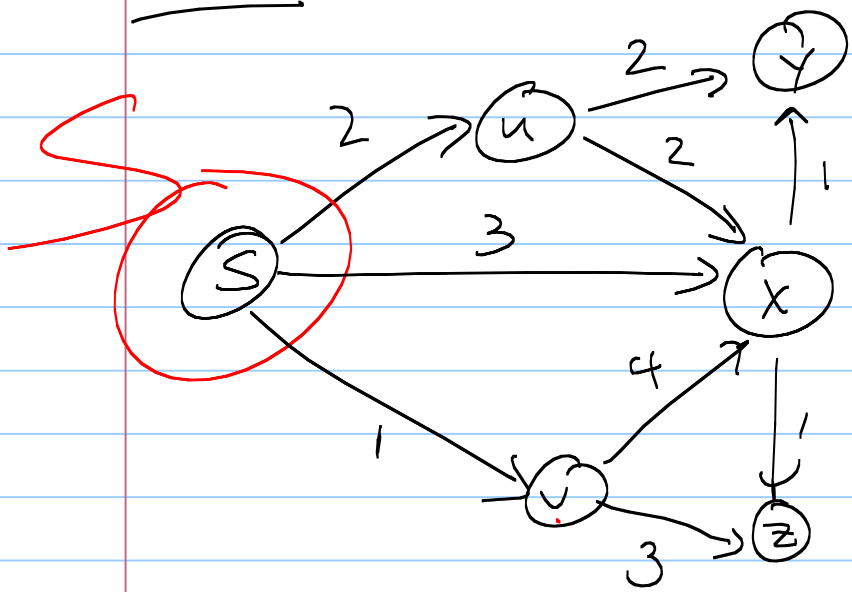
The reason — if we don't explore everything, we don't know if we've missed a shorter path.

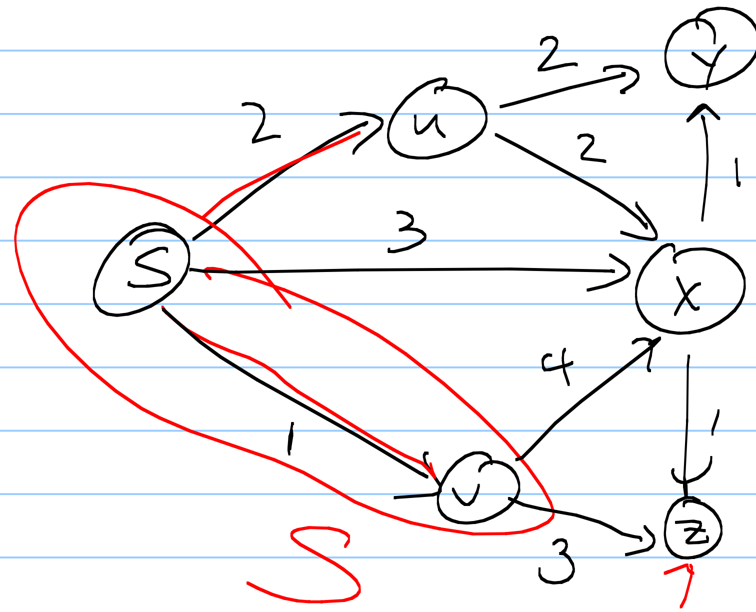
Greedy idea:

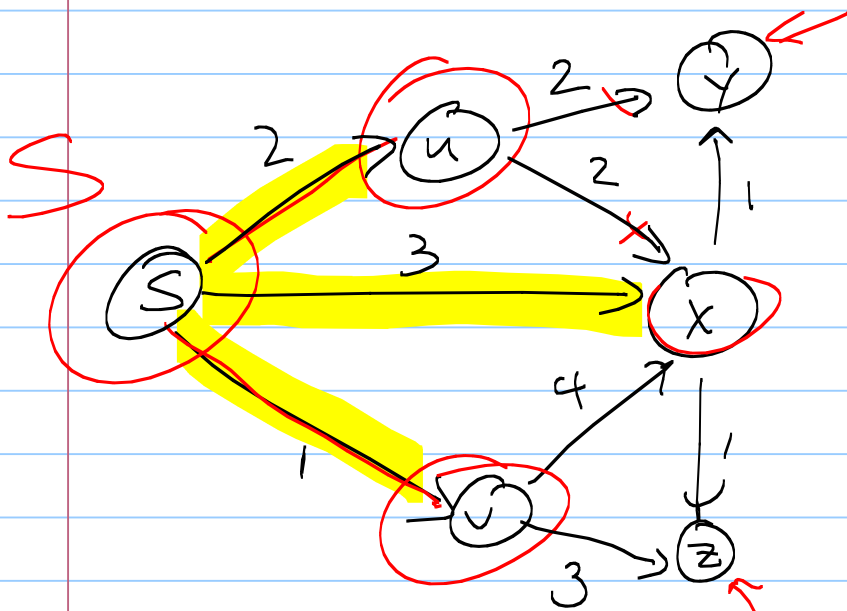
Start with a set S.
(initially $S = \{s\}$)

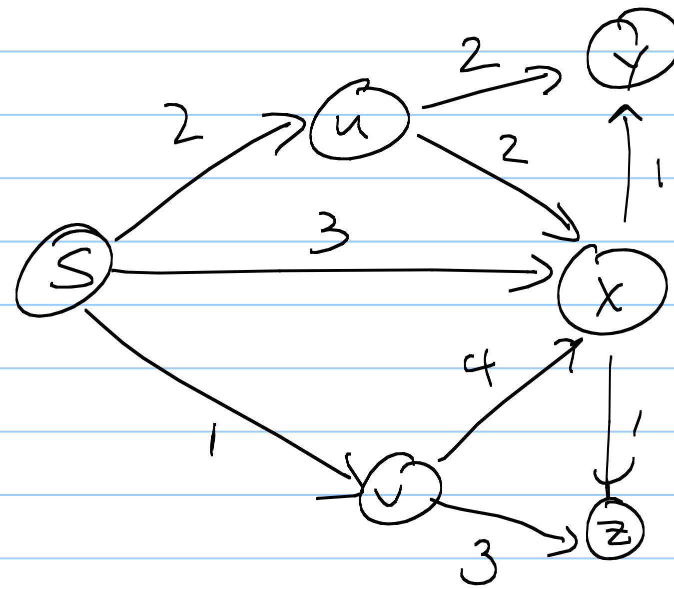
At each step, grow out from s,
taking next shortest path from s to
a new vertex & adding that to S.

Ex:






S
s
u
v
x
y
z
2
2
2
3
1
4
1
1
3


s
u
v
x
y
z
2
2
2
3
1
4
1
1
3