Lecture 10

Wednesday, February 7, 2024 2:47 PM

Review:

Single Source Shortest Path (SSSP) Algorithms

1) BFS $O(|V|+|E|)$
2) Dijkstra's Algorithm $O((|V|+|E|)\log|V|)$
3) Bellman-Ford Algorithm $O(|V| * |E|)$
4) DAG - SSSP - Algorithm $O(|V|+|E|)$

Greedy Algorithms:

1) Scheduling:

Thm: First finsh time is optimal

Proof: Exchange Argument

2) Compression (Huffman Codes)

Goal: Encode text with T letters from an alphabeth Γ with n letters and frequencies $\{f_i : i \in \Gamma\}$

Ex: $\Gamma = \{A, B, C, D\}$ $T = 100$

Sybol	Frequency f_i	Code1	Code2	Code3
A	80	00	0	0
B	10	01	1	11
C	5	10	10	100
D	5	11	11	101
	Cost:	200	110	130

Prefix-Problem: In Code 2, how to decode

$\boxed{10 = BA \text{ or } C\ ?}$

- B and C have same prefix

<u>Prefix-Free Property</u>:

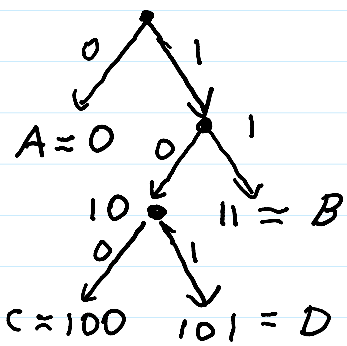
No codeword can be prefix of another

<u>Tree Representation</u>

Binary tree:

0 in i^{th} position $\Leftrightarrow$ go left in level i

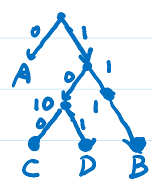
Codewords on leaves $\Longleftrightarrow$ prefix-free



<u>Full binary tree</u>:

every node has 0 or 2 children

Why Full?



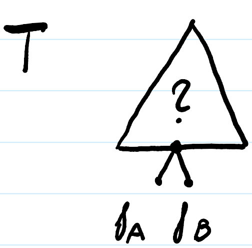
is not optimal

Q: Is the code $\{0, 11, 100, 101\}$ optimal?

How do we find optimal codes in general?

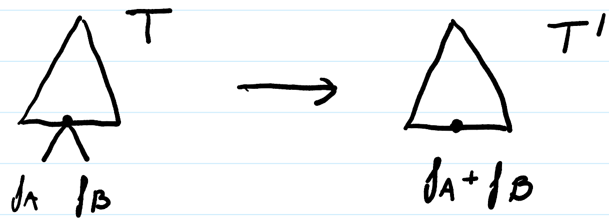
$$Cost(T) = \sum_{i=1}^{n} f_i \times \text{depth of } i$$

<u>Greedy</u>: smallest f_i should be the leaves with largest depth, say f_A and f_B $\Rightarrow$ Build tree bottom up



How to continue?

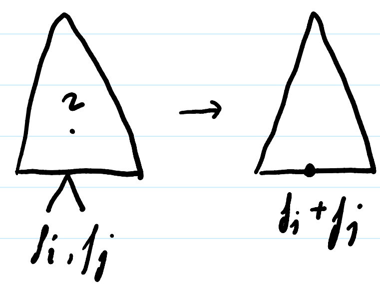
- new alphabeth with $n-1$ letters: $A' = A$ or B



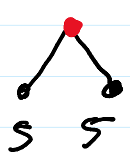
$$\text{cost}(T) = \text{cost}(T') + f_A + f_B$$

Huffman Code

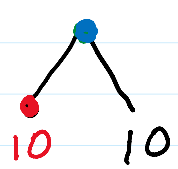
- Start with $F = \{f_1, \cdots, f_n\}$
- Find lowest two, f_i and f_j
- Remove f_i, f_j from F
- Add $f_1 + f_2$ to F
- Iterate



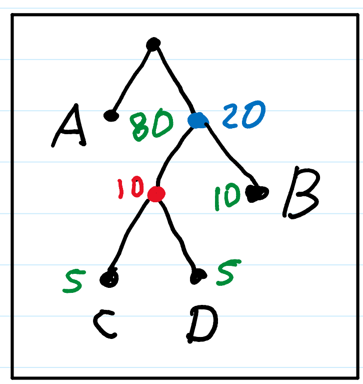
Example $f_1 = 80$, $f_2 = 10$, $f_3 = 5$ $f_4 = 5$

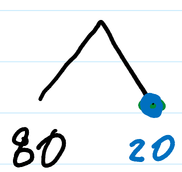


$\rightarrow \{ f_1 = 80,\ f_2 = 10,\ f_3' = 10 \}$



$\rightarrow \{ f_1 = 80,\ f_3' = 20 \}$





Huffman (f)

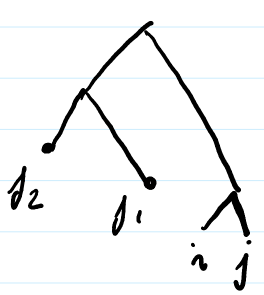
Input: $f[1, \cdots, n]$ frequencies
Output: encoding tree with n leaves

H = priority queue
For $i = 1, \cdots, n$: insert $(i, f[i])$
For $k = n+1, \cdots 2n-1$
    u = DeleteMin v = DeleteMin
    add (k, u) and (k, v) to E
    $f[k] = f[u] + f[v]$
    insert $(k, f[k])$

Correctness Proof:

Claim 1: Order $f_1 \le f_2 \le \cdots$. Then $\exists$ optimal encoding tree s.th. f_1 and f_2 are assigned to two syblings which are leaves of maximal depth

Pf: Choose optimal encoding tree
Choose leaf i of maximal depth
Full tree $\Rightarrow$ must have sybling j
depth(i) maximal $\Rightarrow$ j is leaf

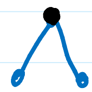


exchange f_1, f_2
with f_i, f_j
if needed

$\Rightarrow$ cost can only go down $\Longrightarrow$ new optimal tree with property from claim

Lemma: Huffman finds optimal tree

Pf by induction:

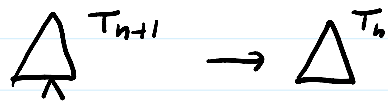
Base: $n=2$ trivial 

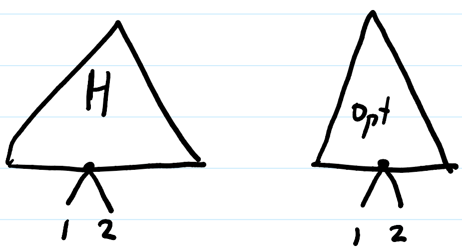
Pf: $n \to n+1$

T_{n+1} optimal. By Claim 1

$\Rightarrow \exists\, T_{n+1}$ s.th. T_{n+1} is optimal and 1 and 2 are leaves

Define T_n by pruning T_{n+1}





$$cost(H_{n+1}) = f_1 + f_2 + cost(H_n)$$

$$cost(T_{n+1}) = f_1 + f_2 + cost(T_n)$$

By induction

$$cost(H_n) \le cost(T_n)$$

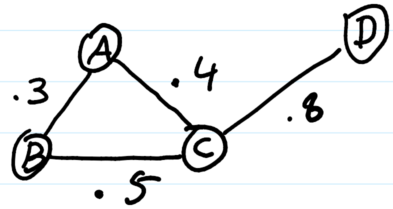
$$\Rightarrow cost(H_{n+1}) \le cost(T_{n+1})$$

$\Rightarrow H_{n+1}$ is optimal ∎

Kruskal and Prim

Given Graph

$G = (V, E)$



Goal: Find edge subset $T \subseteq E$ with minimal total weight $W(T) = \sum_{e \in T} w_e$ which keeps G

connected

Rem: We can choose T to have no cycles, i.e., to be a tree

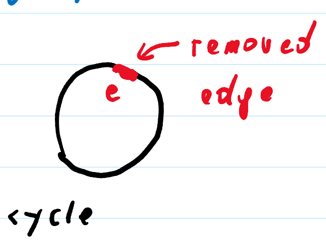
$\Rightarrow$ Problem become to find **minimum spanning tree (MST)**

Trees

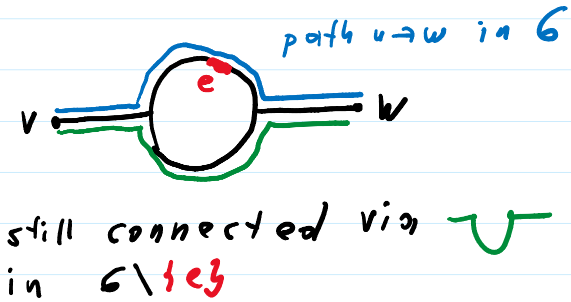
Def: An undirect connected graph without cycles

Claim 1: If G is connected and contains a cycle, removing any edge on a cycle can't disconnect it

Pf by Picture:

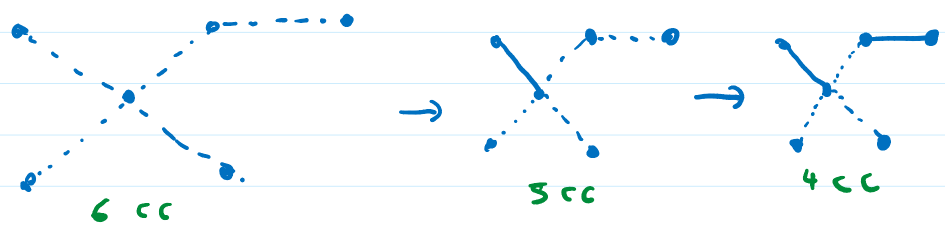


want to prove v connected to w in G $\Rightarrow$ still connected



Claim 2: A tree T with n nodes has $n-1$ edges

Pf: Remove all edges, add them back one by one.
Each time, we reduce # of connected components (cc) by 1



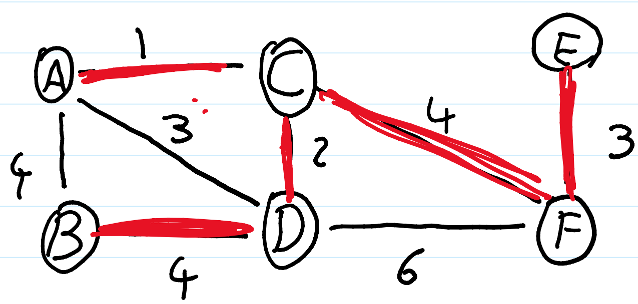
Note: we can never add an edge within a cc, since that would create a cycle.

Claim 3: If G is connected with n nodes and $n-1$ edges $\Rightarrow$ G is a tree

Pf: Follows from Claim 1+2

Greedy Algorithm (Kruskal)

- Choose edges in order of weights
- skip an edge if it creates cycle



- Prove it is optimal
- Find data structure to implement it

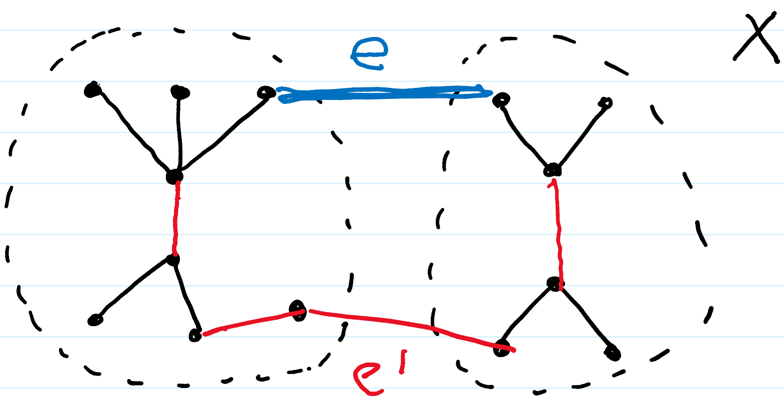
The Cut Property:

Suppose $X \subseteq E$ is part of a MST of G

Let $S \subseteq V$ s.th. X has no edge from S to $V \setminus S$

Let e be the lightest edge from S to $V \setminus S$

Then: $X \cup e$ is part of some MST



<u>PF:</u> let T be the MST s.th $X \subseteq T$

let $e' \in T$ s.th. it connects S to $V \setminus S$

add e to $T \Rightarrow$ creates cycle

remove $e' \Rightarrow$ still connected

$n-1+1-1$ edges $\Rightarrow$ new tree T'

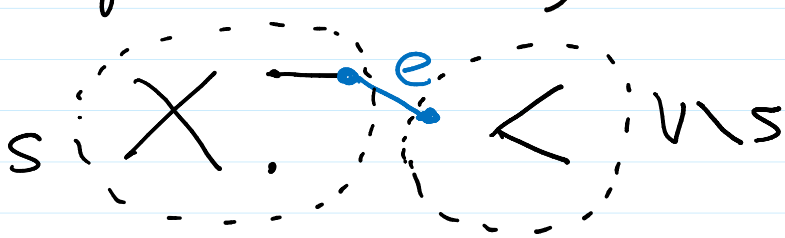
$W(T') = W(T) - w_{e'} + w_e \leq W(T)$ ■

<u>Rem:</u> The partition $S, \bar{S} = V \setminus S$ is called a cut

Prove Kruskal find MST

<u>Recall:</u> Adds lightest edge e not creating cycle

X edges at time $t \Rightarrow$



$\Rightarrow$ If X is part of MST, so is $X \cup \{e\}$

Proof (K finds MST)

By induction

Base Case $X = \emptyset$

$|X| = k \rightarrow |X| = k+1$ Cut property

Implementation of Kruskal

How to check for cycles?

Keep track of connected comp. (cc)

Disjoint Set Datastructure ("Union Find")

- makeset(x) makes singleton containing x
- Find(x) which set does x belong to
- union(x,y) merges sets contain. x,y

Kruskal(G,w)

```
For all v∈V  makeset(v)
X = { }
Sort edges in E by w(·)
∀ {u,v} ∈ E in that order
    if Find(u) ≠ Find(v)
        add {u,v} to X
        union(u,v)
return X
```

$n = |V|$, $m = |E|$

n makeset

sort $O(m \log m)$

$= O(m \log n)$

$2m$ Find

$n-1$ union

Running time

makeset $O(1)$ union, find $\log n$

total: $O((m+n)\log n)$

Master Algorithm

$X = \{\ \}$

repeat until $|X| = n-1$

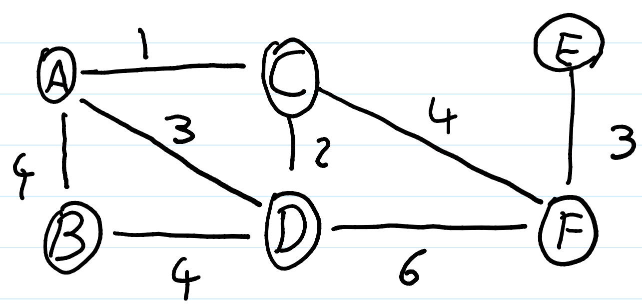
pick $S \subseteq V$ s.th. X contains no edge between S & $\bar{S}$

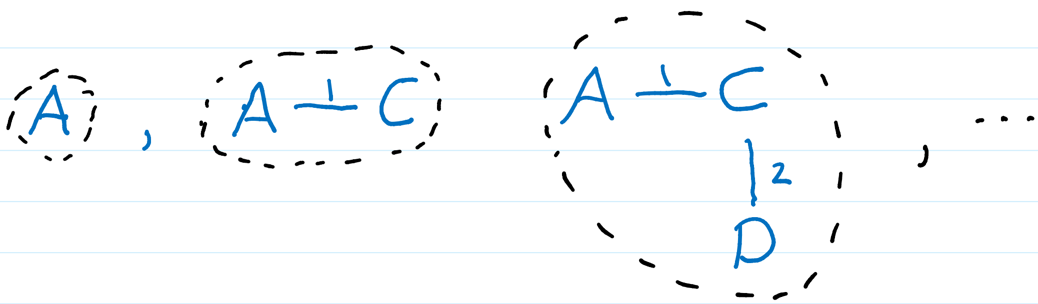
let e be minimum weight edge between S & $\bar{S}$

$X = X \cup \{e\}$

Prims Alg:

Choose S to be connected





Effectively, need to minimize

$$cost(v) = \min_{u \in S} w(v, u)$$

to decide which new vertex v to add

<u>Prim(G, w)</u>

$\forall u \in V$ $cost(u) = \infty$, $prev(u) = nil$

Pick any $u_0 \in V$

$cost(u_0) = 0$

$\forall v \in V$ insert key$(v, cost(v))$

while queue non empty

    v = DeleteMin

    $\forall \{v, u\} \in E$

        if $cost(u) > w(v, u)$

            $cost(u) = w(v, u)$

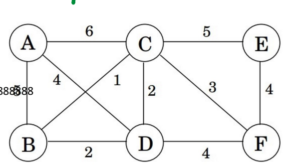
            $prev(u) = v$

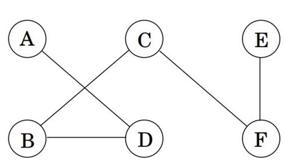
            DecreaseKey(u)

$O(n)$ insert, delete $O(m)$ decrease key

$\to O((n+m)\log n)$ running time

<u>Example</u>





Set S	A	B	C	D	E	F
$\{\}$	0/nil	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil
A		$5/A$	$6/A$	$4/A$	∞/nil	∞/nil
A, D		$2/D$	$2/D$		∞/nil	$4/D$
A, D, B			$1/B$		∞/nil	$4/D$
A, D, B, C					$5/C$	$3/C$
A, D, B, C, F					$4/F$	

← cost / pre