Cheat Sheet- Quiz 2

Wednesday, September 25, 2019 12:09 AM

Better Master Theorem

General Form: $T(n) = aT(\frac{n}{b}) + f(n)$

where, $a \geq 1$, $b > 1$, $f(n) = \Theta(n^k \log^p n)$

Need 2 values:

① $\log_b a$

② k

Case 1: if $\log_b a > k$ then $T(n) \in \Theta(n^{\log_b a})$

Case 2: if $\log_b a = k$

Sub-Case 1: if $p > -1$, $T(n) \in \Theta(n^k \log^{p+1} n)$
Sub-Case 2: if $p = -1$, $T(n) \in \Theta(n^k \log(\log(n)))$
Sub-Case 3: if $p < -1$, $T(n) \in \Theta(n^k)$

Case 3: if $\log_b a < k$

Sub-Case 1: if $p \geq 0$, $T(n) \in \Theta(n^k \log^p(n))$
Sub-Case 2: if $p < 0$, $T(n) \in O(n^k)$

Ex. $T(n) = 2T(\frac{n}{2}) + 1$

$a = 2$
$b = 2$
$f(n) = \Theta(1)$

$\log_2 2 = 1$, $k = 0$

$b = 2$

$f(n) = \Theta(1)$

$= \Theta(n^0 \log^0 n)$

$k = 0, \quad p = 0$

Case 1

$T(n) \in \Theta(n)$

Ex. $T(n) = 4T(\frac{n}{2}) + n$

$a = 4$

$b = 2$

$\log_2 4 = 2, \quad k = 1, \quad p = 0$

$\log_2 4 = 2 > k = 1$

Case 1:

$T(n) \in \Theta(n^2)$

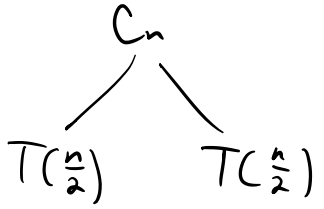
Recurrence Trees

Mergesort - show cost at each level, # of levels to get cost of alg.

$$T(n) = \begin{cases} c & n = 1 \\ 2T(\frac{n}{2}) + cn & n > 1 \end{cases}$$

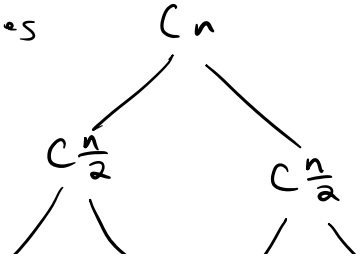
c = Constant

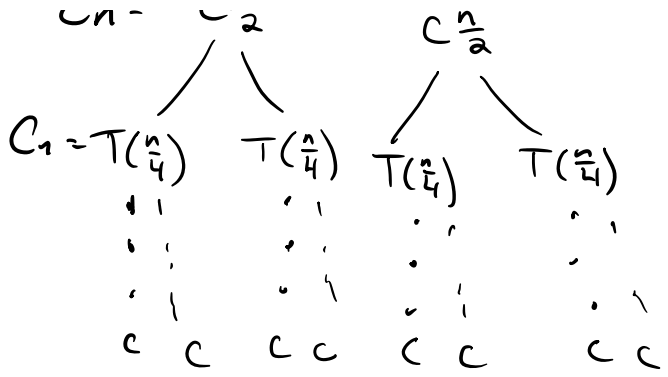
Root of tree is Top-Level Recurrence



Each $T(\frac{n}{2})$ contributes to cost, replace w/ $c\frac{n}{2}$

$cn =$





How Many levels in tree and Cost of each level?

Level i below root has 2^i nodes, each with cost of $C(\frac{n}{2})$

Cost of level = $2^i \times C(\frac{n}{2}) = Cn$

levels = $\log_2 n + 1$ +1 for root

Cost of level = Cn

Total Cost: $Cn \log_2 n + Cn$

$\Theta(n \log n)$

Unrolling

Plug in a few terms to see pattern, write as Summation.

Ex. $T(n) = 2T(\frac{n}{2}) + cn$

$T(\frac{n}{2}) = 2T(\frac{n}{4}) + \frac{c}{2}$

$T(n) = 2[2T(\frac{n}{4}) + \frac{cn}{2}] + cn$

$= 4T(\frac{n}{4}) + 2cn$

$T(\frac{n}{4}) = 2T(\frac{n}{8}) + \frac{cn}{4}$

$T(n) = 8T(\frac{n}{8}) + 3cn$

$T(n) = 16T(\frac{n}{16}) + 4cn$

$T(n) = 2^k T(\frac{n}{2^k}) + kcn$

Base Case: $T(1) = 1$: Where recursion stops when $\frac{n}{2^k} = 1$

Solve for k: $2^k = n$

# of unrolls to BC $\longrightarrow$ $k = \log_2(n)$

Substitute

$T(n) = 2^{\log_2 n} * T(1) + (\log_2 n)nc$

$= n + n\log_2 nc$

$\boxed{T(n) = \theta(n\log(n))}$

Dijkstra's Algorithm - Single Source Shortest Path

- Include destination and solve for a for specific vertex.
- Shortest paths for all other verts.

Given $G = (V, E)$

Let S be set of solved verts
For each $u \in S$, store $d(u)$

Init $S = \{s_0\}$ so source is solved
$d(s_0) = 0$

While $S \neq V$
Select vert $v \notin S$ with at least one edge from S, where:
$d'(v) = \min_{e=(u,v),\ u \in S}(d(u) + E_{u,v})$
add to S
Set $d(v) = d'(v)$
End while

Pseudocode -

Dijkstras (G, s) G is $G(V, E)$, S is Source
Initialize dist, prev $\forall v \in G$ vertex:
dist(s) = 0 dist: path length to vert
dist(v) = ∞ $\forall$ v - s prev: Parent vert
Q = min priority queue
Q.add(V) // add all verts to Q
While Q ! empty // Priority Queue uses dist to vertex as key
u = Q.pop()
for each v in u.adj.
d = dist(u) + e_{uv}
if d < dist(v)
dist(v) = d
prev(v) = u

return G // return graph with dist and prev set for verts.

and prev set for verts.

Analyzation

Always true that Dijkstras finds shortest path to vertex V.

Proof of Correctness:

Let p_v be the shortest path to vert v

Induction - Prove: For set S with starting vertex S_0, for each $u \in S$, the path P_u is the shortest S_0 to u path.

Base Case: $|S| = 1$. Since we have $S = \{S_0\}$ and our $dist(S_0) = 0$ we have shortest path to S_0. thus base case holds.

Inductive Hypothesis: $|S| = k$. Assume claim holds for $|S| = k$, there are k solved verts in solved set for $k \geq 1$

Inductive Step: We grow $|S|$ to $k+1$ by adding vertex v.

k+1 refers to what vertex is solved next.

Greedy Algo Stays ahead

Interval Scheduling -

Set of requests = $\{1, 2, 3, \ldots, n\}$, the i^{th} request has start time s_i and finish time f_i. Subset of requests is compatible if no two overlap in time $f_i \leq s_i$.

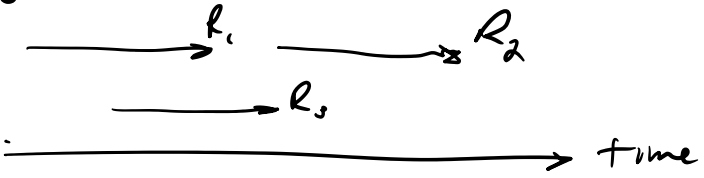
Compatible if no two overlap in time
$f_i \leq s_i$

Optimize: Accept largest set of requests, max size of subset.

Greedy Decision Rule:
- Select Earliest Start time: Won't produce optimal

R_1 selected
Opt: $R_2 + R_3$ selected

- Shortest time interval:
$f_i - s_i$ minimize



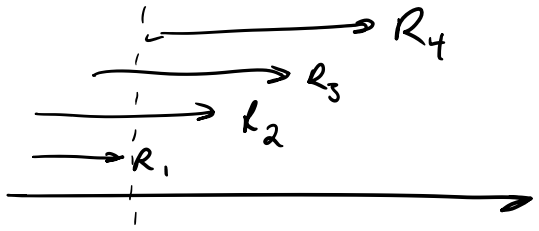
R_3 Selected
opt: $R_1 + R_2$ selected

- Solution that works
select request w/ smallest finish time

Algo - R = Set of requests
A = Set of accepted requests

While R not empty
Choose request $i \in R$ with smallest f_i
Add request i to A
Delete all requests from R not

Add request i to A
Delete all requests from R not compatible with i, $s_i < f_i$
End while
return A



Select R_1
Delete R_2, R_3
Select R_4

Analyzing Algo

- Correctness and Optimality
- Runtime

Correctness

Prove that A is compatible set of requests

Optimality

Show algo produces interim solution that is as good as optimal solution.

There is a # of requests satisfied in optimal solution i.e same #, different requests

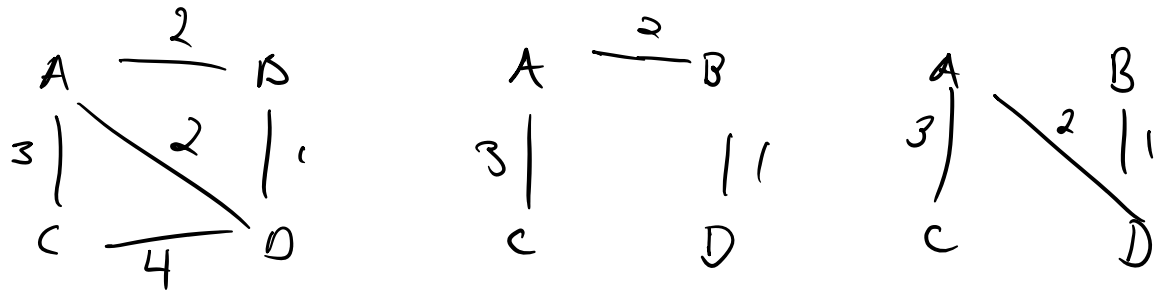
- Given a connected, undirected graph $G = (V, E)$ V are vertices, E are edges, a spanning tree is a subgraph that is undirected tree, contains

V are vertices, E are edges, a spanning tree is a subgraph that is undirected tree, contains all verts of G.

- In a weighted graph, the weight of subgraph is sum (Edges)

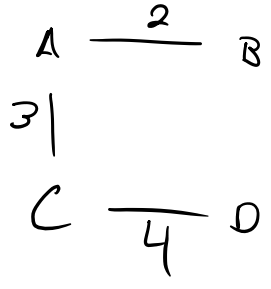
- A min spanning tree (MST) is a spanning tree with min weight.

Ex.



MST
6

MST
6



Spanning tree but not min

There can be multiple MSTs in graph

And spanning trees that aren't MST

Applications
- Running cable between cities
- Find clusters by removing low cost edges

Algos for MSTs

- Kruskal's and Prim's
  - Both use greedy strategy of growing tree

- Both use greedy strategy of growing tree one edge at a time, selecting edge w/ min cost.

- Both manage set of edges A where A maintains LI prior to each iteration, A is a subtree of some MST.

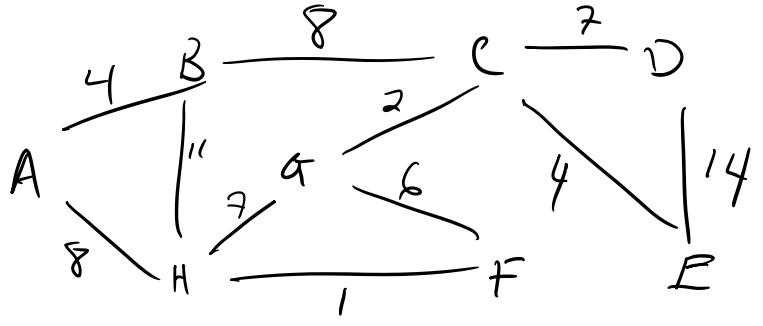
- At each step, determine (u,v) that can be added to A s.t. A U (u,v) is also a subset of an MST. Safe edge.

Kruskals

```
Kruskal's (G)
  R = G.E      // R is min priority queue, G.E
  A = ∅              is all edges in G.
                // A is solution set of edges, initially Null
  While R ! empty
      remove min(u,v) edge from R
      if (u,v) doesn't create cycle in A
          add (u,v) to A

  return A
```

Loop until no more edges to inspect.



- Selects min from all possible edges.

Prim's:
- Always forms single tree, adds edge to tree
- Greedy Algo
- Selects min that connects to existing tree.

Vertex:
Cost: edge cost to vert
Parent: Vert at other end of edge.

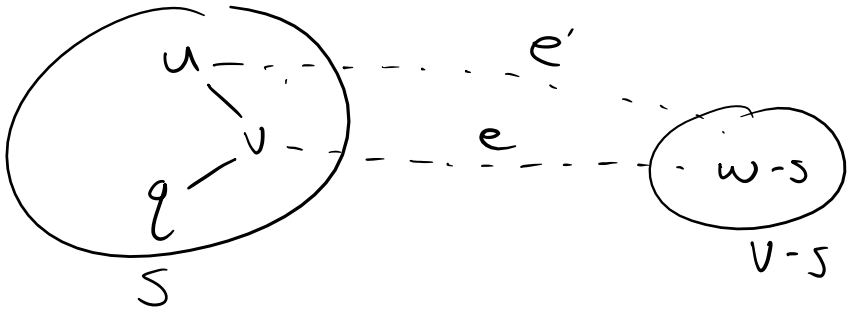
```
Prim's(G):          // G(V, E)
    v.cost = ∞
    v.parent = NuLL
    Q = G.V         // min priority queue of all verts. Cost is key
    G.root.cost = 0 // define starting of tree that is built
    While Q ! empty:
        u = Q.pop() // remove min cost vert from Queue
        for each v ∈ u.adj:
            if v ∈ Q and e_u,v < v.cost
                v.parent = u
                v.cost = e_u,v
```

Theory of MST Algos

What is a "safe" edge?

Cut Property: - Assume distinct edge cost
- Let S be set of verts, not empty and S ≠ V

- Let S be set of verts, not empty and $S \neq V$
- Let edge $e = (v, w)$ be min cost edge $v \in S$ and $w \in V - S$



Every MST contains edge e

<u>Proof:</u> Let T be a spanning tree that doesn't contain e. Show T is not min possible cost.

<u>Exchange Argument</u> - Identify edge e' in T more expensive than e and exchange e for e', which results in spanning tree cheaper than T.

Ends of e are in S and $V-S$. need e' that also has ends in S and $V-S$ so there's still a path v, w Let $w' \in S$ and $v' \in V-S$ s.t. $e' = (v', w')$ so in Spanning tree T.

Exchange e for $e' \Rightarrow$ edge-set

$$T' = T - \{e'\} \cup \{e\}$$