1/28/16

midterms 1-3, parts of ch 4

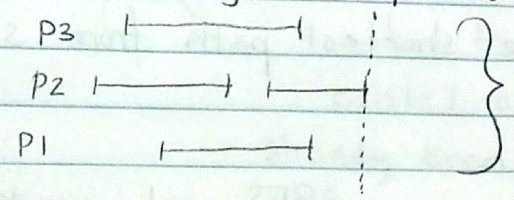
## Greedy Paradigm

ex: our scheduling problem where we only chose intervals based on how early it ended

we did not look at all the intervals only the ones that fit our condition

runtime for greedy solutions usually pretty quick but difficult to prove

ex: scheduling (multi-processors)




algorithm

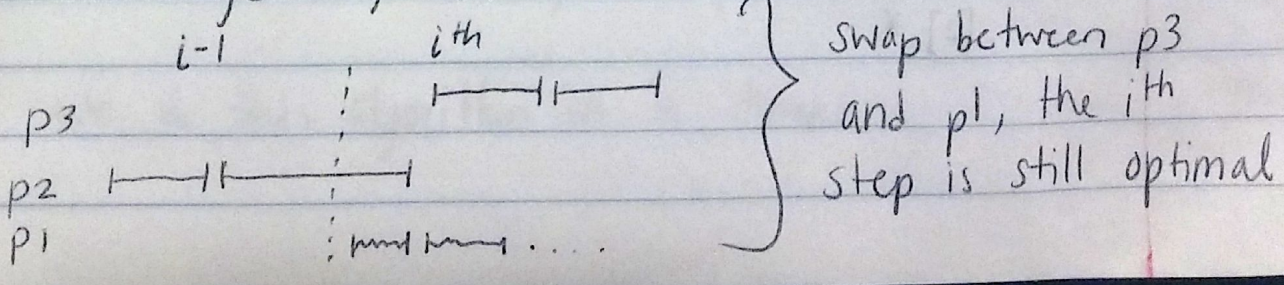
will sort intervals by start time

if a processor if free, assign the earliest starting interval to it
if all processors are busy, create a new processor

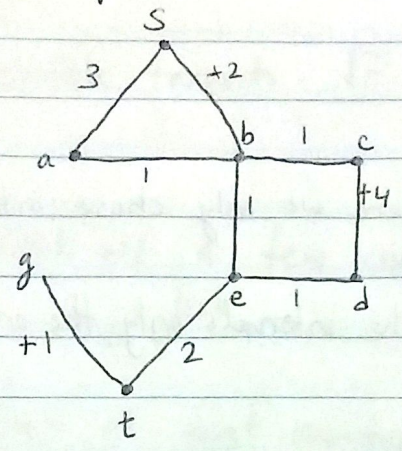
Proof:

up to the i-1 step, the algorithm is optimal by way of contradiction, assigning the $i^{th}$ interval to the wrong processor

"exchange argument"

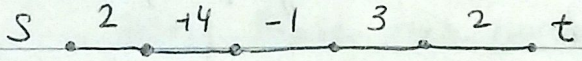


Graph (undirected) (edges are weighted)




What is the shortest path (assume no negative cycles)?

we're going to assume there are no negative edges but the algorithm is similar

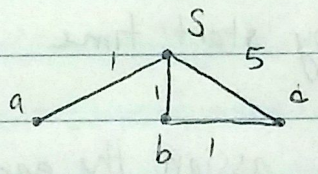



to get the distance we add up the edges

we're solving the shortest path from s to sinks

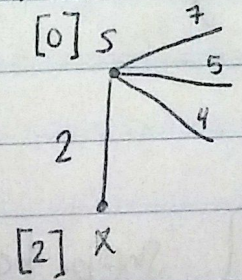
## BFS Tree

BFS only works for unweighted graphs




How can we solve using a greedy paradigm?

of all neighbors of s, pick the edge that is shortest. if I add a positive to all the edges the smallest path will be the one neighbor with the smallest edge

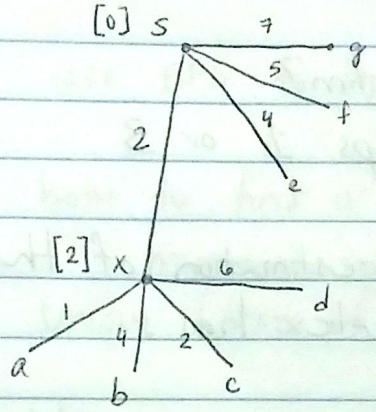


how to expand the algorithm?

this finds the shortest path between s and one neighbor

→ we have to define the next choice

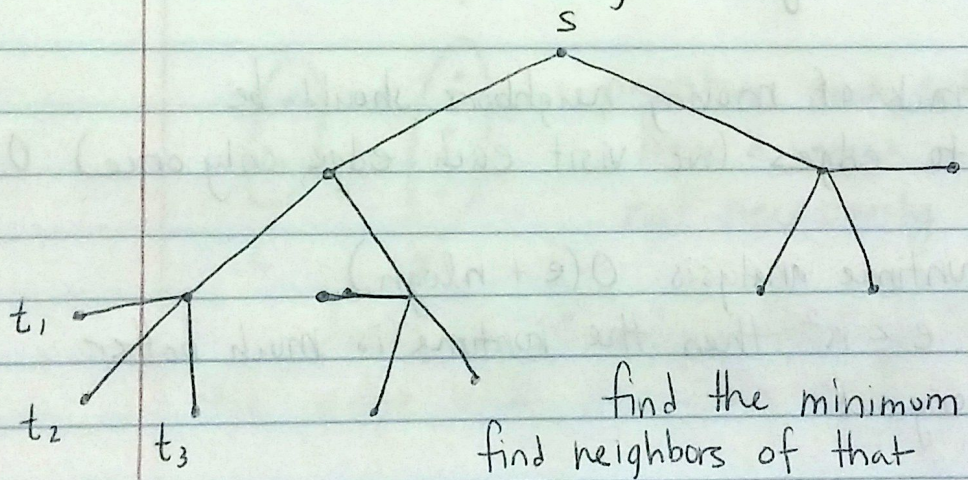
for each neighbor of the vertices we've visited, ask about their distance to s




general step

at any given time based on observed vertices/edges find their path

the shortest path from s-a has the length of 3 if there existed a shorter path it would have already been included in the set




"Dijkstra Shortest path"

find the minimum @ that level, find neighbors of that

Shortest path depends on weight not links

algorithm: find min t → x

fix its distance

move neighbors of x to frontier

we do this algorithm in n times

how long does it take to find min t?
loosely n units of time

fixing distance — constant
moving neighbors — $(n-1)$
$(n)(n-1)(1)$
cynical analysis: $O(n^2)$

Can we improve the algorithm?
→ possibly change steps 1 or 3

step 3 is an extreme overestimation of the number of neighbors each vertex has

- use a heap^ structure to keep track of the minimum $O(n \log n)$
  data

min heap

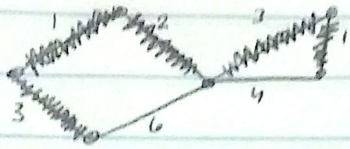
- keeping track of moving neighbors should be charged to edges (we visit each edge only once) $O(e)$

new runtime analysis: $O(e + n \log n)$
if $e < n^2$ then the runtime is much better than the original

Tip: solve new problems systematically

# Minimum Spanning Tree

mw refers to mst w/in graph




[remove edges until this graph becomes a tree]

① has to be a tree (no cycle) } definition of
② must connect all vertices } MST

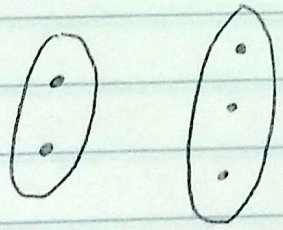
use BFS to find a spanning tree

how to find a minimum spanning tree?

minimum spanning tree is not always unique (think of a triangle w/ equal edges)

how many mst possible for graph w/ n vertices?

exponential $2^n$




partition: each set must have a vertex
not necessarily balanced

look at all edges that go L → R, take the edge with minimum weight