LEC13 Dijkstra's Algorithm notes

4.1 https://github.com/GUMI-21/MIT6.006_note

Review

- Single-Source Shortest Paths on weighted graphs
  SSSP
- Previously
  O(|V| + |E|) for small positive weights or DAGs *BFS*
  Bellman-Ford, O(|E||V|)-time for general graphs with detecting negative wegiths
- Today
  faster for gengeral graphs with non-negative edge weights.
  $e \in E, w(e) >= 0$

Restrictions		SSSP Algorithm		
Graph	Weights	Name	Running Time $O(\cdot)$	Lecture
General	Unweighted	BFS	$\|V\| + \|E\|$	L09
DAG	Any	DAG Relaxation	$\|V\| + \|E\|$	L11
General	Any	Bellman-Ford	$\|V\| \cdot \|E\|$	L12
General	Non-negative	Dijkstra	$\|V\| \log \|V\| + \|E\|$	L13 (Today!)

Non-negative Edge Weights

- idea: Generalize BFS approach to weighted graphs

Observation1

if weights >= 0, then distance increase along Shortest Paths.
a examle: if s -> u -> v is the SP => $\delta(s, u) <= \delta(s, v)$

Observation2

we can solve SSSP if given order of vertices in increasing distance.
The idea here is if I can construct a DAG in linear time, or means construct a topological order.
-if I konw the ordering of the increasing distance of verteices, then I can use *DAG relaxation*.

Dijkstra's Algorihm

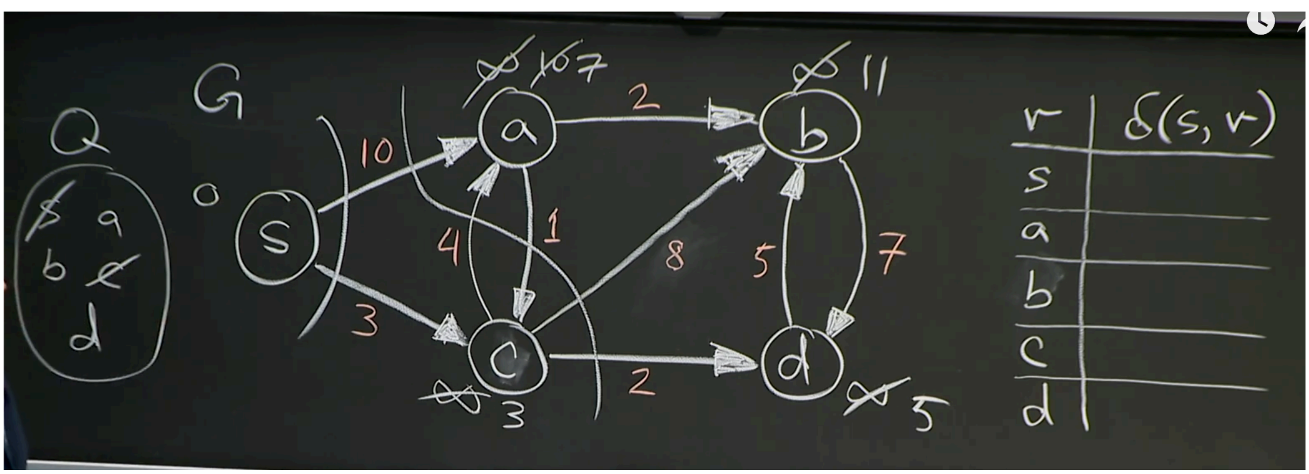
Ideas

- idea1
  Relax edges from vertices in increasing distance from source.
- idea2
  Find next vertex efficiently using a Datastruct *Changable Priority Queue*
  -Q.build(X)
  -Q.delete_min()
  -Q.decrease_key(id, k)
  Priority Queue Q' cross-link with Dict D

Algorithm

-Set d(s,v) = infinity for v in V, set d(s,s) = 0
-Build CPQ Q with an item(v,d(s,v)) for each v in V
While *Q not empty*, delete (u,d(s,u)). from Q that has *minimum estimate distance*
For v in Adj+(u):
if d(s,v) > d(s,u) + w(u,v):
Relax edge(u,v): set d(s,v) = d(s,u) + w(u,v)
Decrease key of vertex v in Q to new d(s,v)

A examle

1. Q(S) = 0 (estimate distance) and others are infinity, so S is minimun, then relax edge outgoing from S => a= 10,c = 3, delete S in Q.
2. Q(c) = 3 is minimum, then relax outgoing edges from c and delete c in Q.
3. then recure this function
   

   answeri of SSSP: 0,7,9,3,5

Correctness

- Claim: d(s,v) = $\delta(s, v)$ for all v \in V at end

- Proof:
  -if ever reaxation sets d(s,v) = $\delta(s, v)$, still true at end. beacuse relaxation only decreases d(s,v), but safe: length of some path (*triangle inequality*)
  -Suffices to show that d(s,v) = $\delta(s, v)$ When v removed from Q
- Proof by induction on first k vertices removed from Q
  Base case: k = 1, d(s,s) = 0
  Inductive Step: Assume true for k < k'
  v' be k'th vertex.
  set x & y in the shortest path from s to v'
  d(s,y) <= $\delta(s, x) + w(x, y) = \delta(s, y) <= \delta(s, v') <= d(s, v') <= d(s, y)$
  because we are popuping minimum from priority queue. => v' = y d=\delta

Running time

build once Q
delete minimum in Q |V| times
for every possible edge, we need to relax and decrese the key in our queue.

- assume build B time, *delete M time*, *decrease D time*
  then take O(B+|V|M + |E|D) time

Priority Queue Q' on n items	Q Operations $O(\cdot)$			Dijkstra $O(\cdot)$ $n = \|V\| = O(\|E\|)$
	build(X)	delete_min()	decrease_key(id, k)	
Array	n	n	1	$\|V\|^2$
Binary Heap	n	$\log n_{(a)}$	$\log n$	$\|E\| \log \|V\|$
Fibonacci Heap	n	$\log n_{(a)}$	$1_{(a)}$	$\|E\| + \|V\| \log \|V\|$

  Fibonacci Heap can look at chapter 19 in CLRS O(|E| + |V|log|V|)
  choose which datastruct to use to build priority Queue need to check the graph is sparse or dense.
  sparse: V colse to E => use Binary Heap => O(|V|kig|V|)
  dense: V less than E => use Array => O(|V|log|V| + |E|log|V|)