Lecture 13: Dijkstra's Algorithm

Review

- Single-Source Shortest Paths on weighted graphs
- Previously: $O(|V| + |E|)$-time algorithms for small positive weights or DAGs
- Last time: Bellman-Ford, $O(|V||E|)$-time algorithm for **general graphs** with **negative weights**
- Today: faster for **general graphs** with **non-negative edge weights**, i.e., for $e \in E$, $w(e) \geq 0$

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:
  - Grow a sphere centered at source s
  - Repeatedly explore closer vertices before further ones
  - But how to explore closer vertices if you don't know distances beforehand? :(
- **Observation 1:** If weights non-negative, monotonic distance increase along shortest paths
  - i.e., if vertex u appears on a shortest path from s to v, then $\delta(s, u) \leq \delta(s, v)$
  - Let $V_x \subset V$ be the subset of vertices reachable within distance $\leq x$ from s
  - If $v \in V_x$, then any shortest path from s to v only contains vertices from V_x
  - Perhaps grow V_x one vertex at a time! (but growing for every x is slow if weights large)
- **Observation 2:** Can solve SSSP fast if given order of vertices in increasing distance from s
  - Remove edges that go against this order (since cannot participate in shortest paths)
  - May still have cycles if zero-weight edges: repeatedly collapse into single vertices
  - Compute $\delta(s, v)$ for each $v \in V$ using DAG relaxation in $O(|V| + |E|)$ time

Dijkstra's Algorithm

- Named for famous Dutch computer scientist **Edsger Dijkstra** (actually Dÿkstra!)

11 August 1982
prof. dr. Edsger W. Dÿkstra
Burroughs Research Fellow

- **Idea!** Relax edges from each vertex in increasing order of distance from source s
- **Idea!** Efficiently find next vertex in the order using a data structure
- **Changeable Priority Queue** Q on items with keys and unique IDs, supporting operations:

`Q.build(X)`	initialize Q with items in iterator `X`
`Q.delete_min()`	remove an item with minimum key
`Q.decrease_key(id, k)`	find stored item with ID `id` and change key to `k`

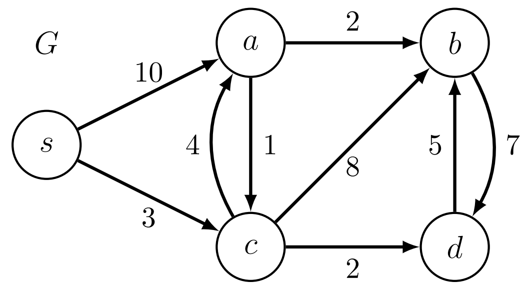
- Implement by **cross-linking** a Priority Queue Q' and a Dictionary D mapping IDs into Q'
- Assume vertex IDs are integers from 0 to $|V| - 1$ so can use a direct access array for D
- For brevity, say item `x` is the tuple `(x.id, x.key)`

- Set $d(s, v) = \infty$ for all $v \in V$, then set $d(s, s) = 0$
- Build changeable priority queue Q with an item $(v, d(s, v))$ for each vertex $v \in V$
- While Q not empty, delete an item $(u, d(s, u))$ from Q that has minimum $d(s, u)$
  - For vertex v in outgoing adjacencies $\text{Adj}^+(u)$:
    - If $d(s, v) > d(s, u) + w(u, v)$:
      - Relax edge (u, v), i.e., set $d(s, v) = d(s, u) + w(u, v)$
      - Decrease the key of v in Q to new estimate $d(s, v)$

- Run Dijkstra on example

Example

Delete	$d(s,v)$				
v from Q	s	a	b	c	d
s	**0**	∞	∞	∞	∞
c		10	∞	**3**	∞
d		7	11		**5**
a		**7**	10		
b			**9**		
$\delta(s,v)$	0	7	9	3	5



Correctness

- **Claim:** At end of Dijkstra's algorithm, $d(s,v) = \delta(s,v)$ for all $v \in V$
- **Proof:**
  - If relaxation sets $d(s,v)$ to $\delta(s,v)$, then $d(s,v) = \delta(s,v)$ at the end of the algorithm
    * Relaxation can only decrease estimates $d(s,v)$
    * Relaxation is safe, i.e., maintains that each $d(s,v)$ is weight of a path to v (or ∞)
  - Suffices to show $d(s,v) = \delta(s,v)$ when vertex v is removed from Q
    * Proof by induction on first k vertices removed from Q
    * Base Case ($k = 1$): s is first vertex removed from Q, and $d(s,s) = 0 = \delta(s,s)$
    * Inductive Step: Assume true for $k < k'$, consider k'th vertex v' removed from Q
    * Consider some shortest path π from s to v', with $w(\pi) = \delta(s,v')$
    * Let (x,y) be the first edge in π where y is not among first $k' - 1$ (perhaps $y = v'$)
    * When x was removed from Q, $d(s,x) = \delta(s,x)$ by induction, so:

$$\begin{aligned}
d(s,y) &\le \delta(s,x) + w(x,y) && \text{relaxed edge } (x,y) \text{ when removed } x \\
&= \delta(s,y) && \text{subpaths of shortest paths are shortest paths} \\
&\le \delta(s,v') && \text{non-negative edge weights} \\
&\le d(s,v') && \text{relaxation is safe} \\
&\le d(s,y) && v' \text{ is vertex with minimum } d(s,v') \text{ in } Q
\end{aligned}$$

    * So $d(s,v') = \delta(s,v')$, as desired $\square$

Running Time

- Count operations on changeable priority queue Q, assuming it contains n items:

Operation	Time	Occurrences in Dijkstra
`Q.build(X)` $(n = \lvert\texttt{X}\rvert)$	B_n	1
`Q.delete_min()`	M_n	$\lvert V\rvert$
`Q.decrease_key(id, k)`	D_n	$\lvert E\rvert$

- Total running time is $O(B_{|V|} + |V| \cdot M_{|V|} + |E| \cdot D_{|V|})$
- Assume pruned graph to search only vertices reachable from the source, so $|V| = O(|E|)$

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

- If graph is **dense**, i.e., $|E| = \Theta(|V|^2)$, using an Array for Q' yields $O(|V|^2)$ time
- If graph is **sparse**, i.e., $|E| = \Theta(|V|)$, using a Binary Heap for Q' yields $O(|V| \log |V|)$ time
- A Fibonacci Heap is theoretically good in all cases, but is not used much in practice
- We won't discuss Fibonacci Heaps in 6.006 (see 6.854 or CLRS chapter 19 for details)
- You should assume Dijkstra runs in $O(|E|+|V| \log |V|)$ time when using in theory problems

Summary: Weighted Single-Source Shortest Paths

Restrictions		SSSP Algorithm	
Graph	Weights	Name	Running Time $O(\cdot)$
General	Unweighted	BFS	$\lvert V\rvert + \lvert E\rvert$
DAG	Any	DAG Relaxation	$\lvert V\rvert + \lvert E\rvert$
General	Non-negative	Dijkstra	$\lvert V\rvert \log \lvert V\rvert + \lvert E\rvert$
General	Any	Bellman-Ford	$\lvert V\rvert \cdot \lvert E\rvert$

- What about All-Pairs Shortest Paths?
- Doing a SSSP algorithm $|V|$ times is actually pretty good, since output has size $O(|V|^2)$
- Can do better than $|V| \cdot O(|V| \cdot |E|)$ for general graphs with negative weights (next time!)