Shortest Paths

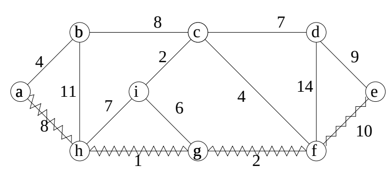
(CLRS 24.0 : 24.3)

The problem

- We discussed that BFS finds shortest paths if the length of a path is defined to be the number of edges on it.
- In general we have weights on edges and we are interested in shortest paths with respect to the sum of the weights of edges on a path.
- Example: Finding shortest driving distance between two addresses (lots of www-sites with this functionality). Note that weight on an edge (road) can be more than just distance (weight can e.g. be a function of distance, road condition, congestion probability, etc).
- Definition: Given a graph $G = (V, E)$ directed and weighted. Given two vertices u, v, the shortest path from u to v is a path of smallest overall weight from u to v. If v is not reachable from u a path does not exist.
- The *weight* of the shortest path from u to v is often denoted as $\delta(u, v)$:

$$\delta(u,v) = \begin{cases} \min\{w(P) : P \text{ is path from } u \text{ to } v\ \}, & \text{if path exists} \\ \infty, & \text{otherwise} \end{cases}$$

- Example: Shortest path from a to e (of length 21). Note that the graph below is undirected and has only positive weights:



SP variants

- *Single pair shortest path*: Find shortest path from u to v. This problem seems easier than SSSP, but in fact all known algorithms for this problem have the same worst-case running time as best SSSP algorithms.
- *Single source shortest path (SSSP)*: Find shortest path from source s to all vertices $v \in V$.
- *All pair shortest path (APSP)*: Find shortest path from u to v for all $u, v \in V$. APSP can be solved by running SSSP $|V|$ times, once for every vertex in G as source. Faster solutions are possible, as we'll see.
- Single-destination shortest paths: Given u, find SP from all vertices in G to u —- this is the reverse of SSSP and can be solved by reversing the graph.

Properties of shortest paths

1. Shortest paths are not necessarily unique, that is, there may be more than one path of weight $\delta(u, v)$ from u to v.

2. Subpaths of shortest paths are shortest paths: If $P =< u = v_0, v_1, v_2, \cdots, v_k = v >$ is shortest path from u to v then for all i $(i < k)$ we have $P' =< u = v_0, v_1, v_2, \cdots, v_i >$ is shortest path from u to v_i; also P” $=< v_i, v_{i+1}, \cdots, v_k = v >$ is shortest path from v_i to $v_k = v$; and for any $i \leq j$, $< v_i, \cdots, v_j >$ is the shortest path from v_i to v_j.

3. Negative cycles and negative edges: If G has a negative weight cycle reachable from u, then the shortest path from u to v is not well defined; the reason is that we can make the weight of a path from u to v arbitrarily low by going through the cycle several times.

   On the other hand, SPs are well defined if G has negative edges but no negative cycles.

4. In general it is possible to find shortest paths if the graph has negative edge weights (but no negative cycles), but the algorithms that are known for this more general case are slower.

5. Can a shortest path have cycles? No! Any shortest path has to be simple (i.e. cannot contain cycles). If p had cycles, the path obtained by removing the cycle would be smaller weight than p so p could not have been a shortest path to start with.

6. How many edges at most on a shortest path? Answer: $|V| - 1$. The longest shortest path in terms of number of edges would have $|V| - 1$ edges and visit all vertices in G.

SP summary

-
-

SSSP for G with non-negative weights: Dijkstra's algorithm

- Dijkstra's algorithm for SSSP is a greedy algorithm:

- Idea: Grow set (tree) S of vertices we know the shortest path to; repeatedly add new vertex v that can be reached from S using one edge. v is chosen as the vertex with the minimal path weight among paths $< s = v_0, v_1, \cdots v_i, v >$ with $v_j \in S$ for all $j \leq i$

- Implemented using priority queue on vertices in $V \setminus S$.

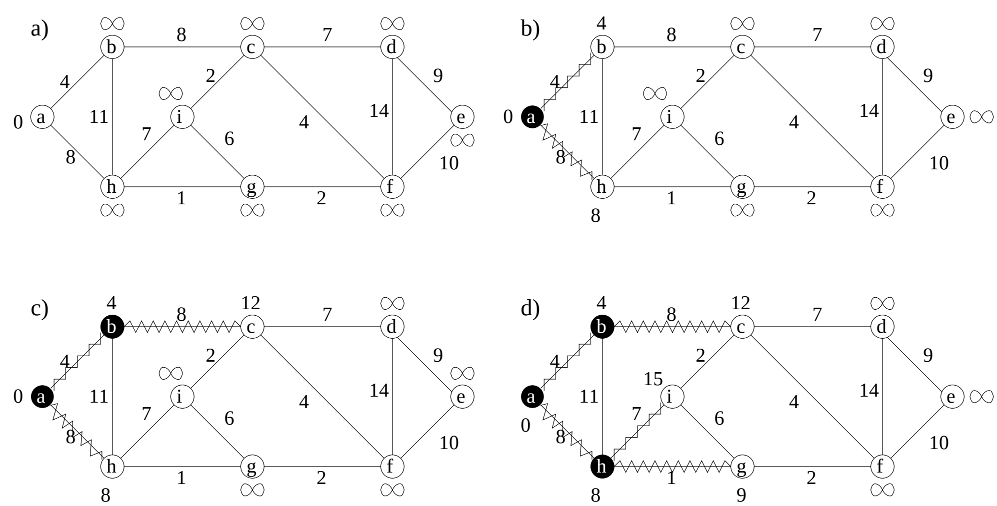
- Algorithm computes *shortest path tree* (stored using parent[v]) which can be used to find actual shortest paths

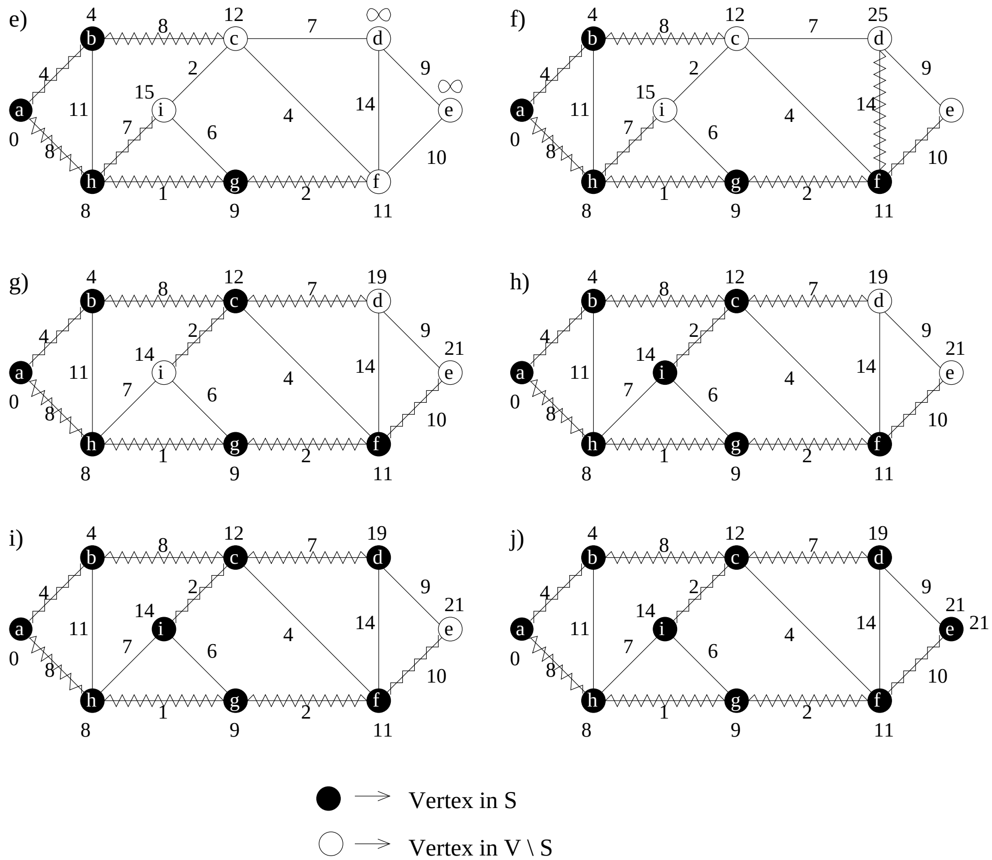
- Algorithm works for directed graphs as well

Dijkstra(s)

  FOR each $v \in V$ DO
    $d[v] = \infty$
    $\textsc{Insert}(Q, v, \infty)$
  $S = \emptyset$
  $d[s] = 0$
  $\textsc{Change}(Q, s, 0)$
  WHILE Q not empty DO
    $u = \textsc{Deletemin}(Q)$
    $S = S \cup \{u\}$
    FOR each $e = (u, v) \in E$ with $v \in V \setminus S$ DO
      IF $d[v] > d[u] + w(u, v)$ THEN
        $d[v] = d[u] + w(u, v)$
        $\textsc{Change}(Q, v, d[v])$
        parent$[v] = u$

- Example:





- Analysis:
  - We perform $|V|$ Insert's
  - We perform $|V|$ Deletemin's
  - We perform at most one Change for each of the $|E|$ edges
    $\Downarrow$
    Assuming edge-list representation and the priority queue implemented with a heap $O((|V| + |E|) \log |V|) = O(|E| \log |V|)$ running time.

- Correctness:
  - We prove correctness by induction on size of S
  - We will prove that after each iteration of the while-loop the following *invariants* hold:
    (I1) $v \notin S \Rightarrow d[v]$ is length of shortest path from s to v among all paths from s to v that contain only vertices from S.
      * (I2)] $v \in S \Rightarrow d[v] = \delta(s, v)$

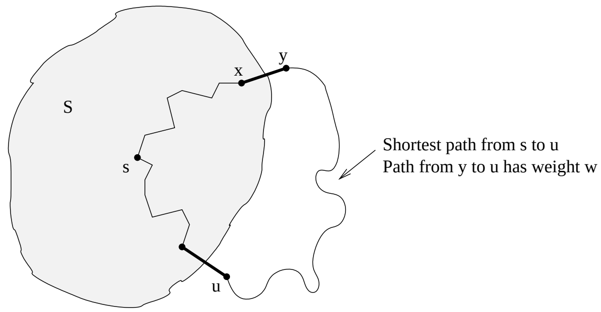
    $\Downarrow$
    When algorithm terminates ($S = V$) we have solved SSSP
  - Proof:
    Invariant trivially holds initially ($S = \emptyset$). To prove that invariant holds after one iteration of while-loop, given that it holds before the iteration, we need to prove that after adding u to S:

(I1) $d[v]$ correct for all $(u, v) \in E$ where $v \notin S$

- Easily seen to be true since $d[v]$ explicitly updated by algorithm (all the new paths to v of the special type go through u)

(I2) $d[u] = \delta(s, u)$

- Assume by contradiction that $d[u] > \delta(s, u)$, that is, the found path is not the shortest.
- Consider shortest path to u and edge (x, y) on this path where $x \in S$ and $y \notin S$ (such an edge must exist since $s \in S$ and $u \notin S$)



- We know that $\delta(s, u) = \delta(s, y) + \delta(y, u) = \delta(s, y) + w$
- We know that $d[y] = \delta(s, y)$ by (I1)
- Therefore $\delta(s, u) = d[y] + w$
- We chose u such that $d[u]$ was minimized $\Rightarrow d[y] > d[u]$
- Therefore $d[u] > \delta(s, u) = d[y] + w > d[u] + w \Rightarrow w$ must be $< 0 \Rightarrow$ contradiction since all weights are non-negative