Shortest Paths

(CLRS 24.0, 24.3)

- 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).

- Formal definition of shortest path:

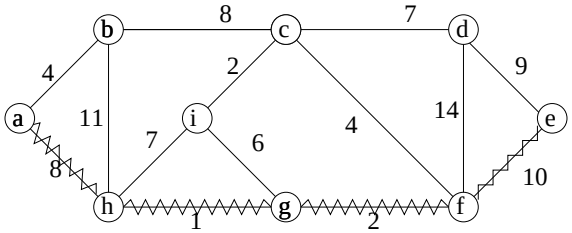
  $G = (V, E)$ weighted graph, directed or undirected.

  Weight of path $P =< v_0, v_1, v_2, \cdots, v_k >$ is $w(P) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$.

  Shortest path $\delta(u, v)$ from u to v has weight

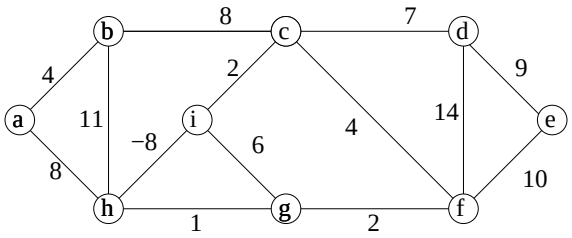
$$\delta(u,v) = \begin{cases} \min\{w(P) : P \text{ is path from } u \text{ to } v\} & \text{If path exists} \\ \infty & \text{Otherwise} \end{cases}$$

  Example: Shortest path from a to e (of length 21)



- Properties of shortest paths:
  - 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 $i < k$ $P' =< u = v_0, v_1, v_2, \cdots, v_i >$ is shortest path from u to v_i
  - No (unique) shortest path exists if graph has cycle with negative weight

    Example: If we change weight of edge (h, i) to -8, we have a cycle (i,h,g) with negative weight (-1). Using this we can make the weight of path between a and e arbitrarily low by going through the cycle several times



On the other hand, the problem is well defined if we let edge (h, i) have weight -7 (no negative cycles)

- **We will *only* consider graphs with non-negative weights.** 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.

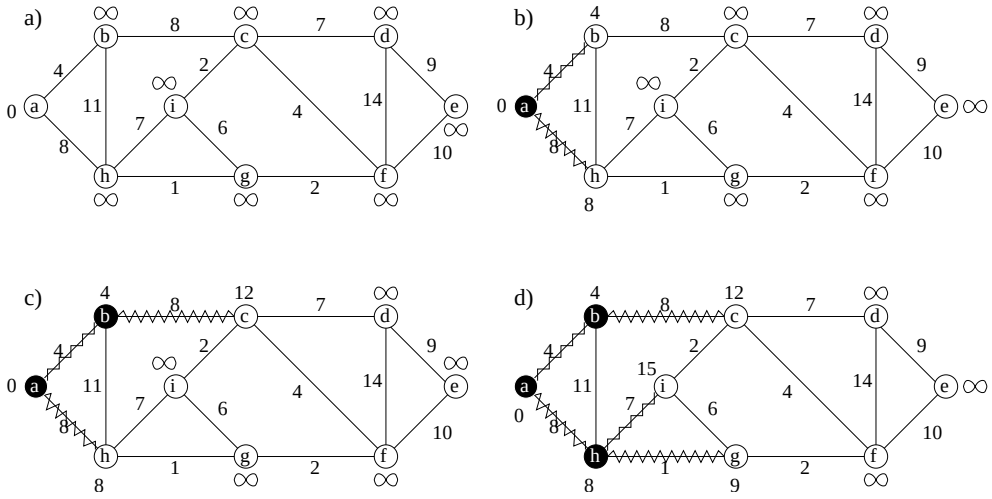
- Different variants of shortest path problem:
  - *Single pair shortest path*: Find shortest path from u to v
  - *Single source shortest path (SSSP)*: Find shortest path from source s to all vertices $v \in V$
  - *All pair shortest path (APSP)*: Find shortest path from u to v for all $u, v \in V$
- Note:
  - No algorithm is known for computing a single pair shortest path better than solving the ("bigger") SSSP problem
  - APSP can be solved by running SSSP $|V|$ times
    ⇓
    We concentrate on the SSSP problem.

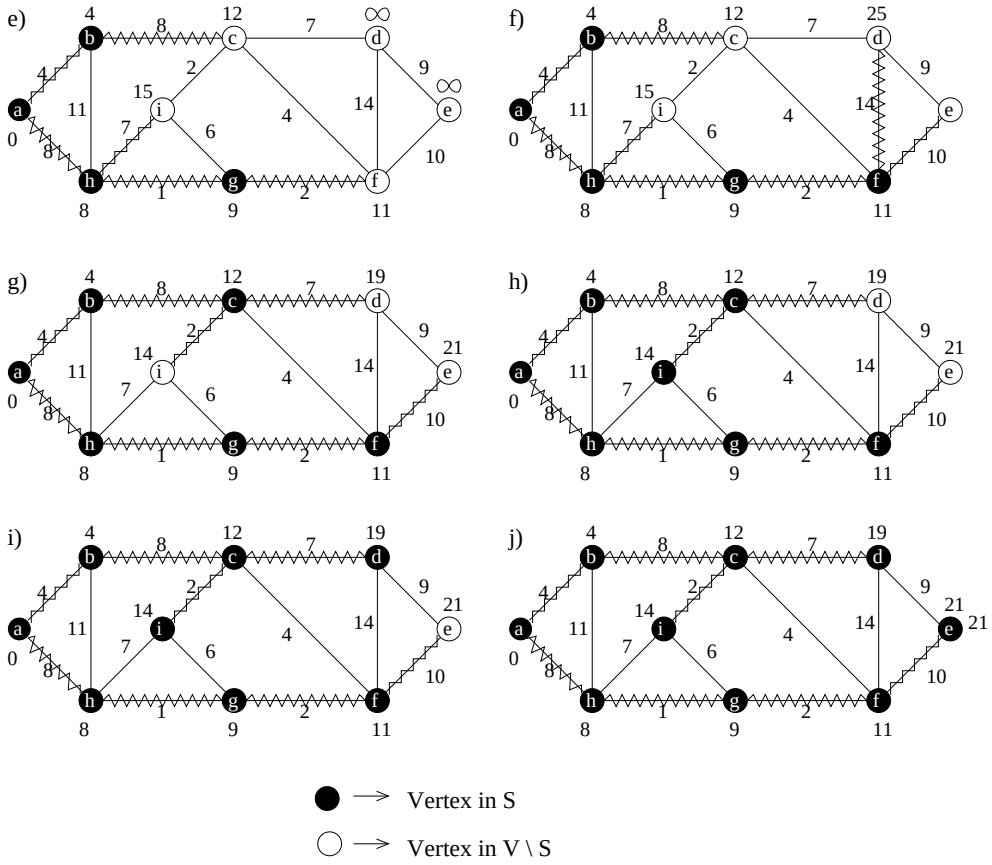
SSSP for non-negative weights—Dijkstra's algorithm

- 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 ∈ V DO
        d[v] = ∞
        INSERT(Q, v, ∞)
    S = ∅
    d[s] = 0
    CHANGE(Q, s, 0)
    WHILE Q not empty DO
        u = DELETEMIN(Q)
        S = S ∪ {u}
        FOR each e = (u, v) ∈ E with v ∈ V \ S DO
            IF d[v] > d[u] + w(u, v) THEN
                d[v] = d[u] + w(u, v)
                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:

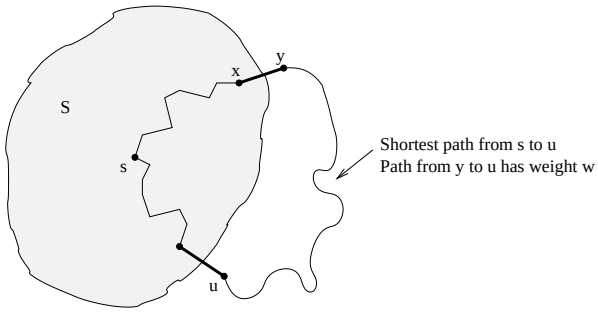
    Invariant trivially holds initially ($S = \emptyset$). To prove that invariant holds after one iteration of while-loop, given that it holds before the iteration, we need to prove that after adding u to S:

(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