Shortest Paths

Ira A. Fulton Schools of Engineering
Arizona State University

Shortest Paths: Part 1

- Shortest Path Problem
- Dijkstra’s Shortest Path Algorithm
- Analysis

Definitions

- A path in a directed graph is a sequence of vertices $p = \langle v_0, v_1, \ldots, v_k \rangle$ s.t. there is an edge connecting v_i to v_{i+1} for all i

- A path in an undirected graph is a sequence of vertices $p = \langle v_0, v_1, \ldots, v_k \rangle$ s.t. there is an edge connecting v_i and v_{i+1} for all i

Definitions

- The weight of a path p = <v_0, v_1, ..., v_k> is

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- The shortest-path weight from vertex u to vertex v is

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \xrightarrow{p} v\} & (\exists p)(u \ldots v) \\ \infty & otherwise \end{cases}$$

Applications: Navigation System

- Suppose you want to find the distance or travel time from one POI (point of interest) to another POI, you can ask Google:

- https://www.google.com/maps

- If you need to know how to go to an address, you can use an app on your phone.

- Why doesn't your phone ask for the source?

Vertex attributes, Initialization

- For each node, v, we have two attributes: $v.d$, $v.\pi$ representing the current distance (from the source) and the current predecessor of vertex v.

- $\boldsymbol{v.d}$ will not increase in the process.

- We initialize distance from the source to ∞

Initialize-Single-Source(G, s)

1. for each vertex v ∈ V[G] do
2. v.d = ∞
3. v.p = nil
4. s.d = 0

Relax(u, v, w)

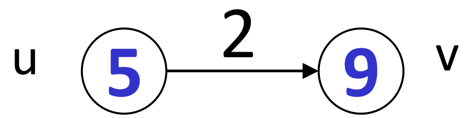
We adjust (if it reduces) the distance at vertex v whenever we traverse an edge (u, v).

Relax(u, v, w)

1. if (v.d > u.d + w(u, v)) then
2. v.d = u.d + w(u, v)
3. v.p = u

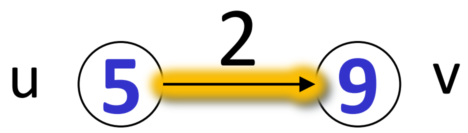
Relax (u, v, w): Illustration

Before:



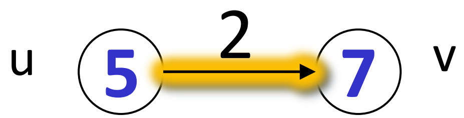
Relax (u, v, w): Illustration

Relax:



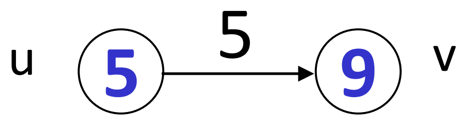
Relax (u, v, w): Illustration

Result:



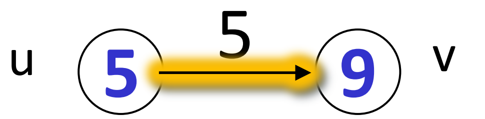
Relax (u, v, w): Illustration

Before:



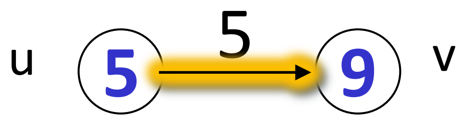
Relax (u, v, w): Illustration

Relax:



Relax (u, v, w): Illustration

Result:



Ira A. Fulton Schools of Engineering

Arizona State University

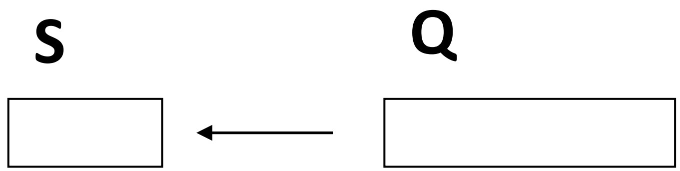
Shortest Paths: Part 2

- Shortest Path Problem
- Dijkstra’s Shortest Path Algorithm
- Analysis

Dijkstra's Algorithm

The idea of the algorithm is the following:

- Use a greedy algorithm
- Use a priority queue Q, where the key of vertex v is the currently computed distance from the source.
- Processed vertices are removed from Q and stored in a set S

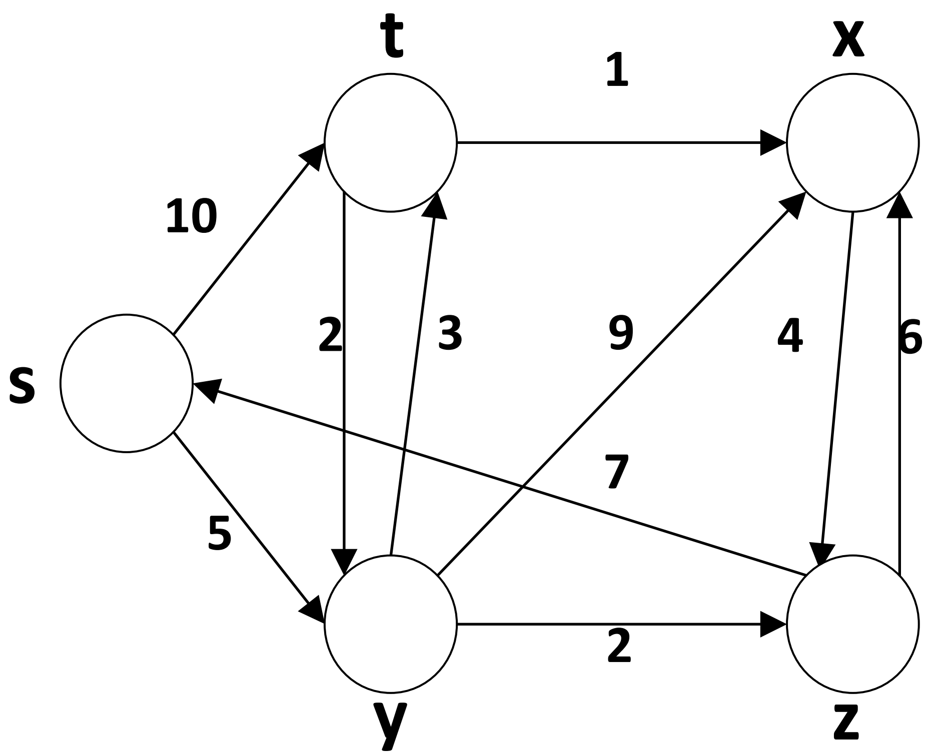


Dijkstra's Algorithm

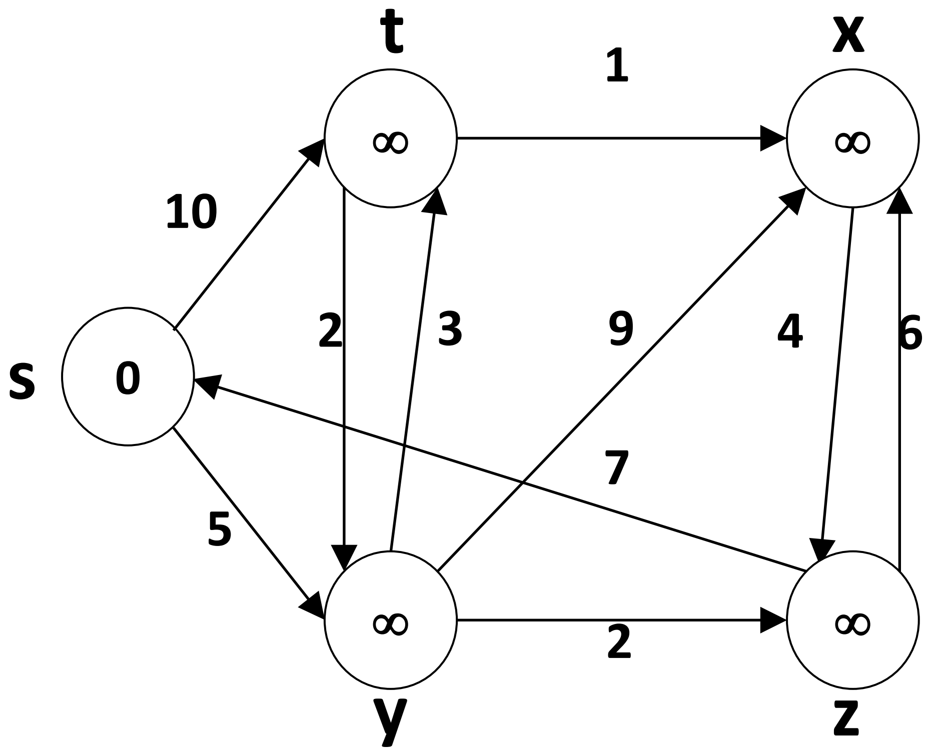
Dijkstra(G, w, s)

```
Initialize-Single-Source(G, s)
S = ϕ
Q = ϕ
for each vertex u ∈ G.V
    Insert(Q, u)
while Q ≠ ϕ do
    u = Extract-Min(Q)
    S = S ∪ {u}
    for each vertex v ∈ u.adj do
        Relax(u, v, w)
        if the call of Relax decreases v.d
            Decrease-Key(Q, v, v.d)
```

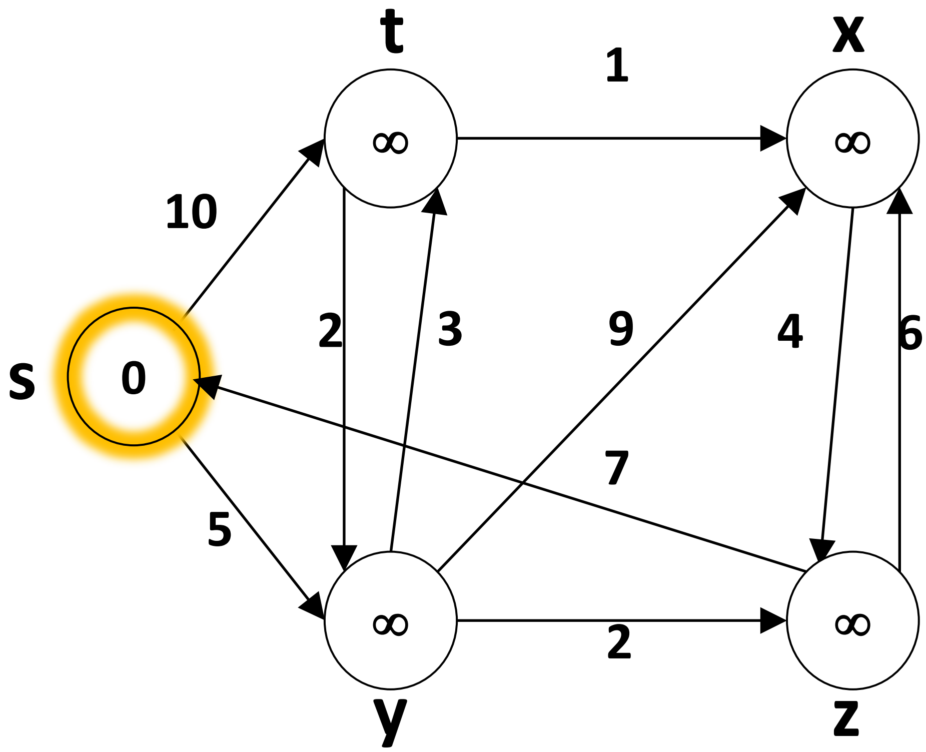
Running Example



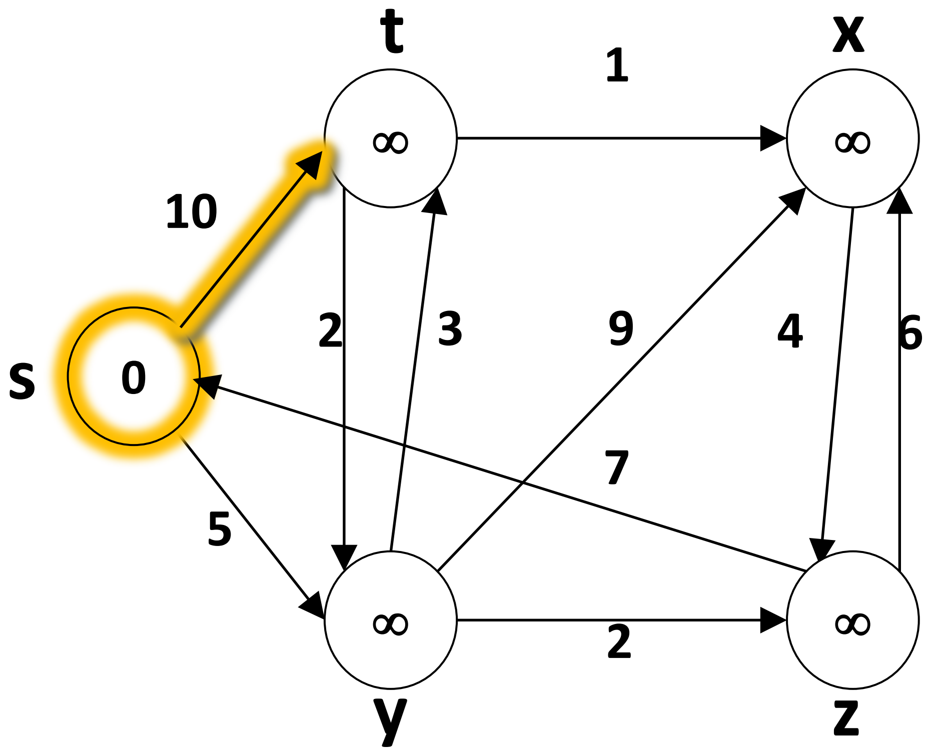
Running Example



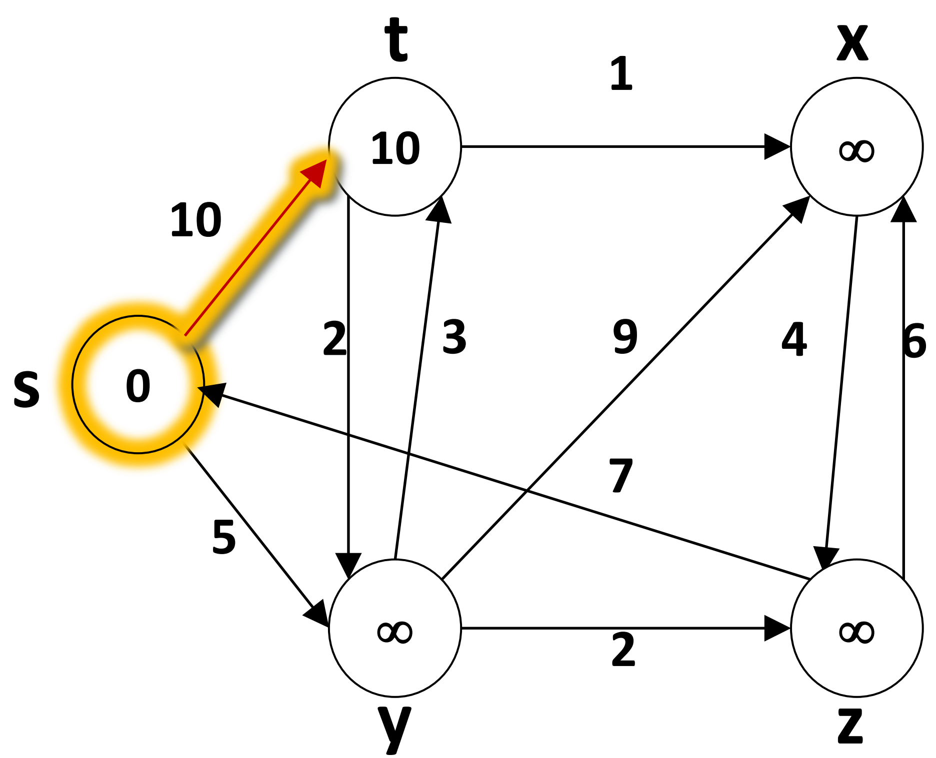
Running Example



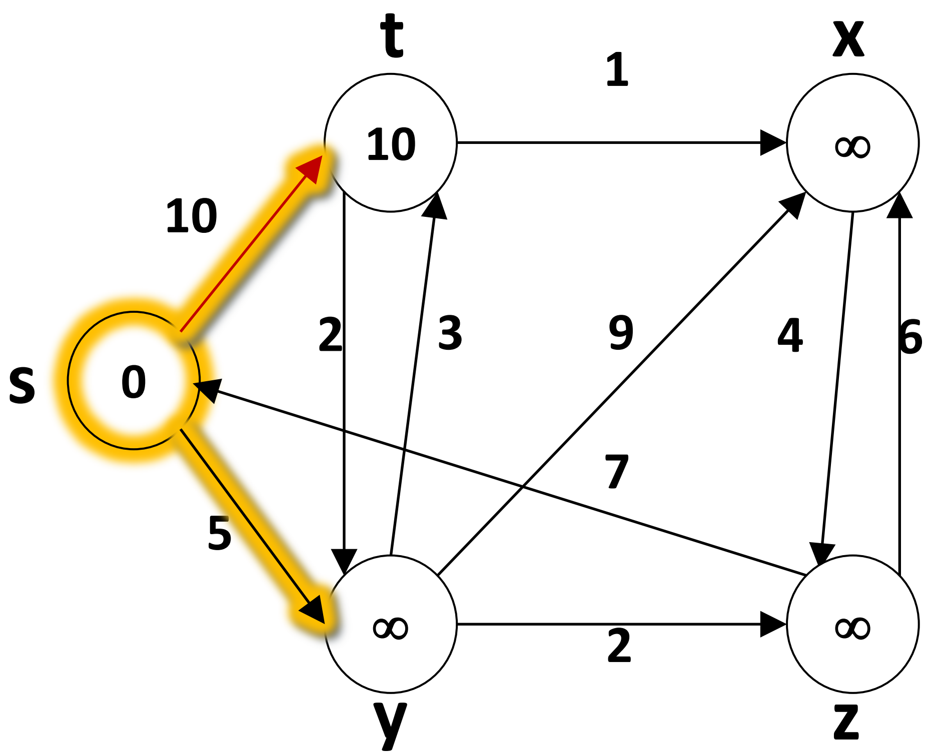
Running Example



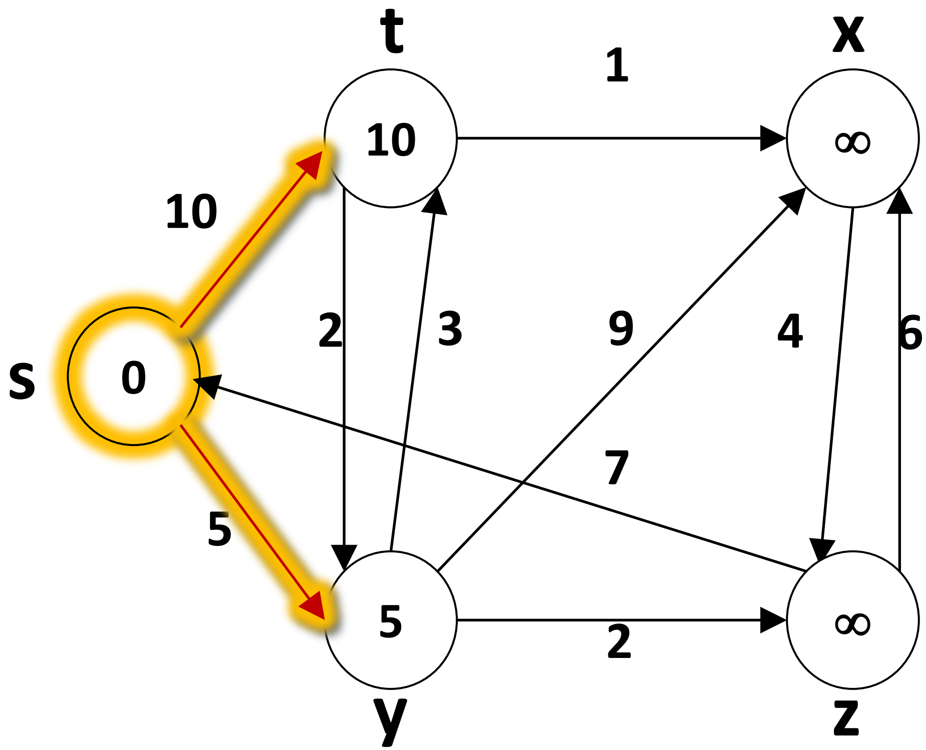
Running Example



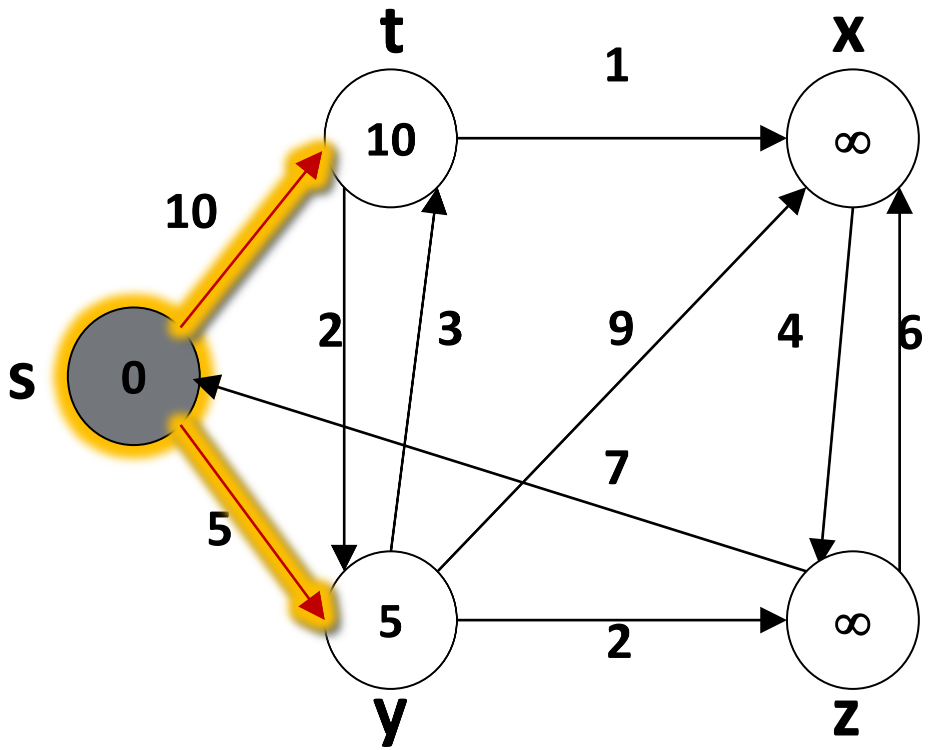
Running Example



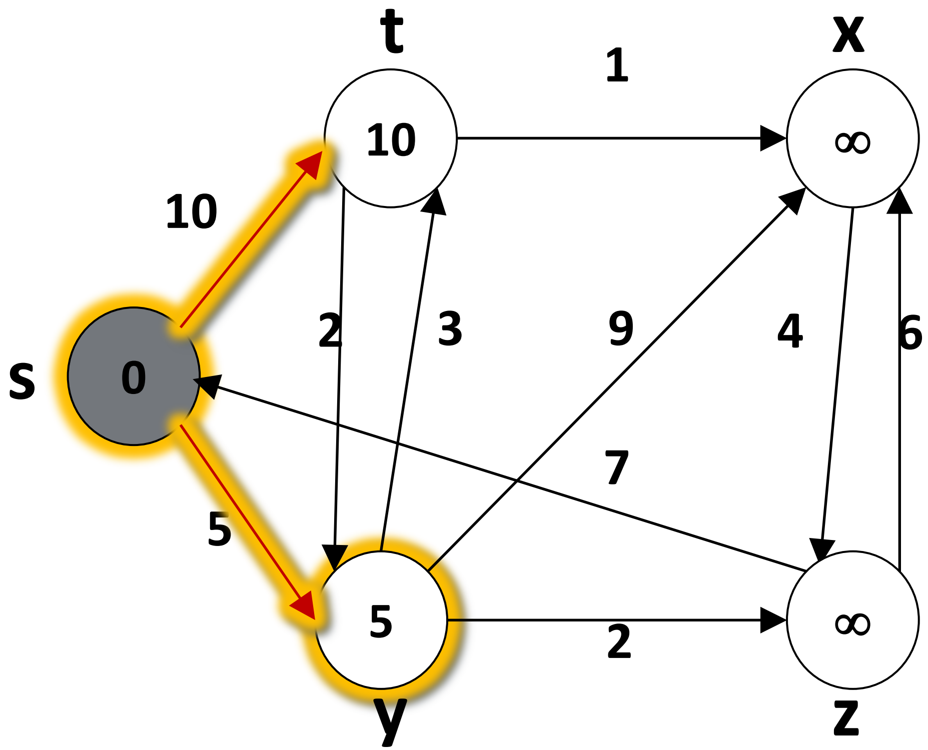
Running Example



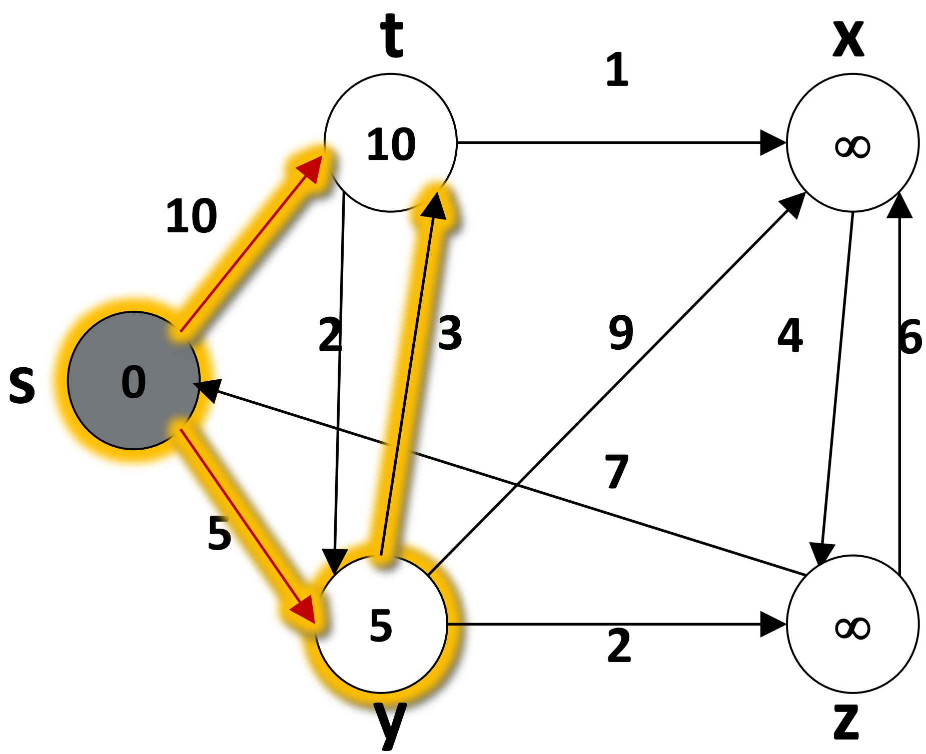
Running Example



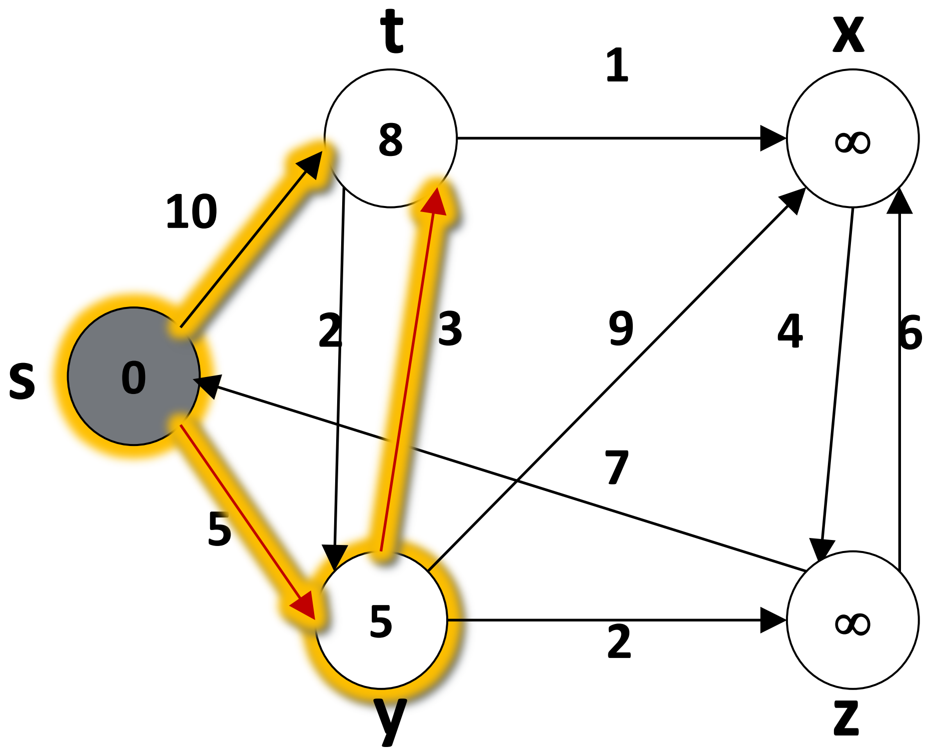
Running Example



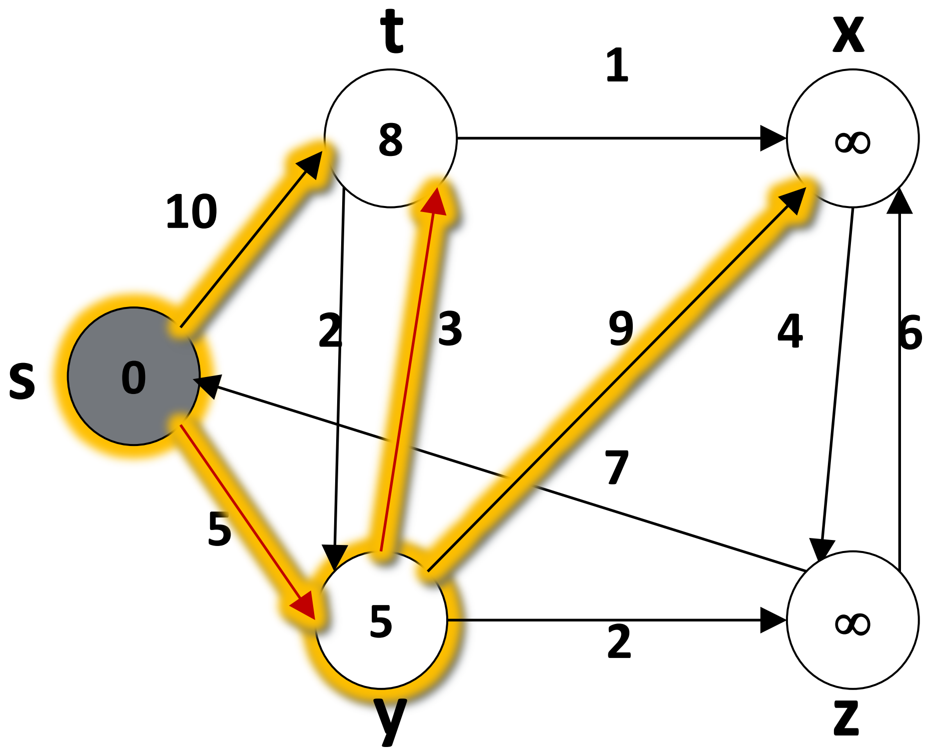
Running Example



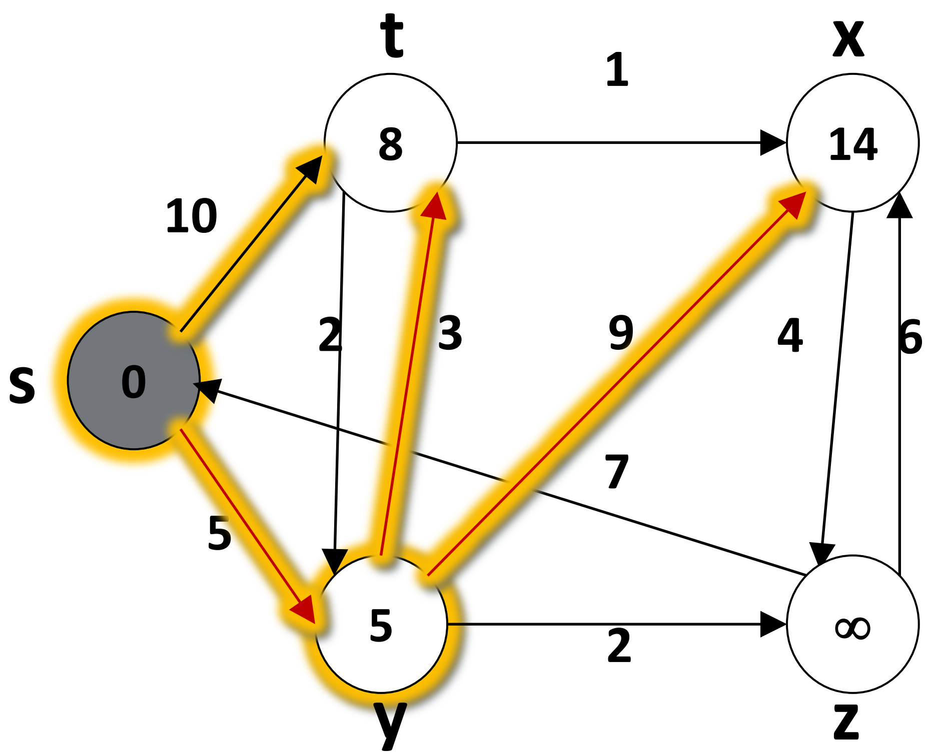
Running Example



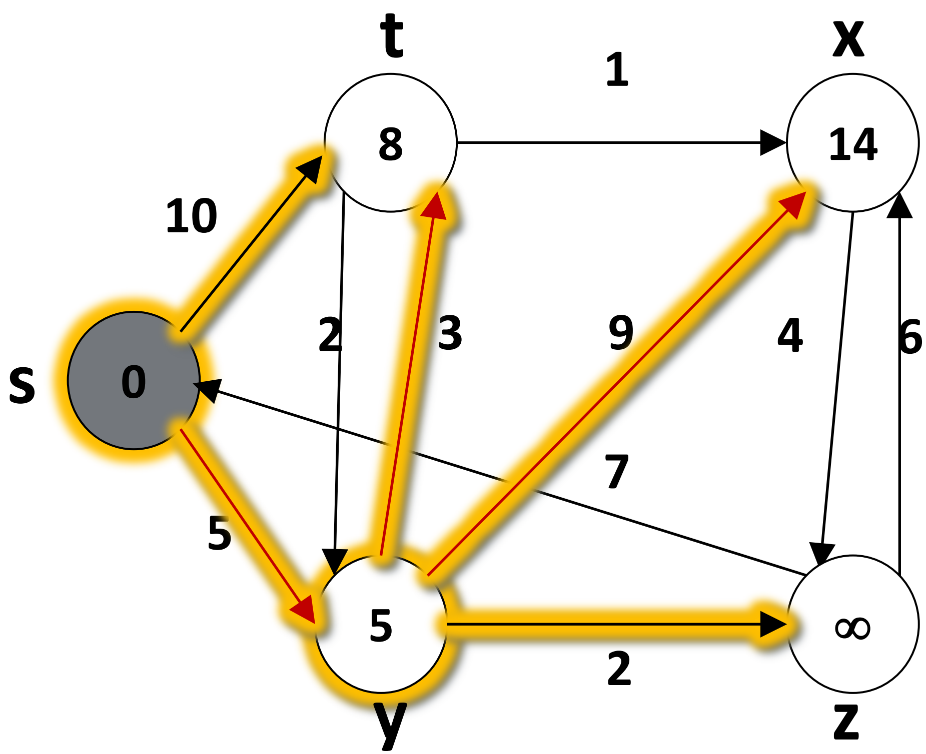
Running Example



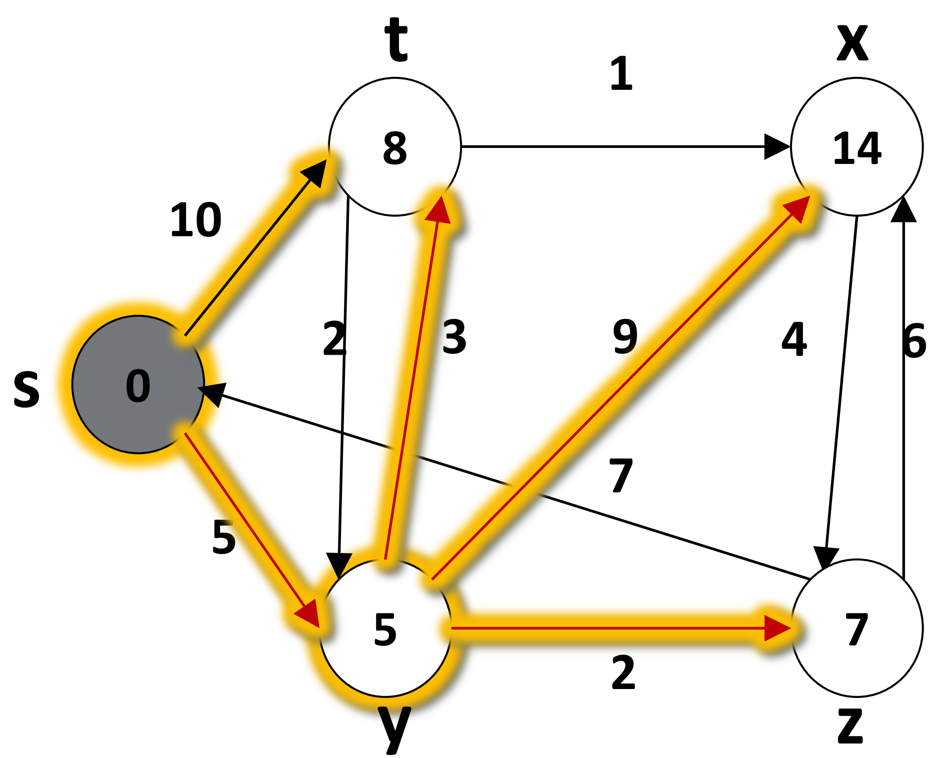
Running Example



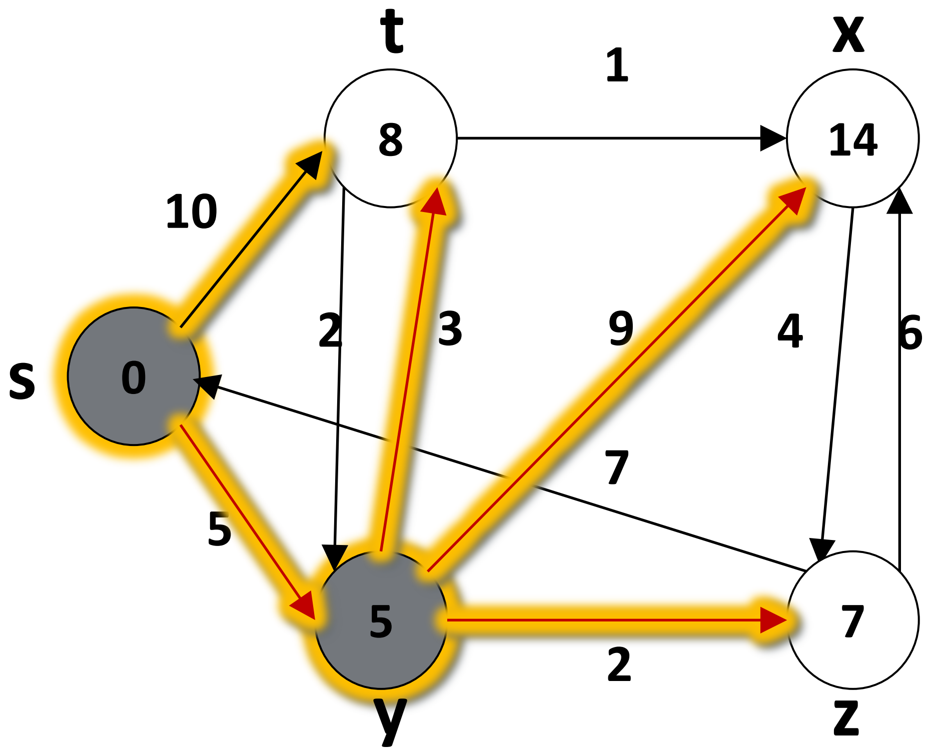
Running Example



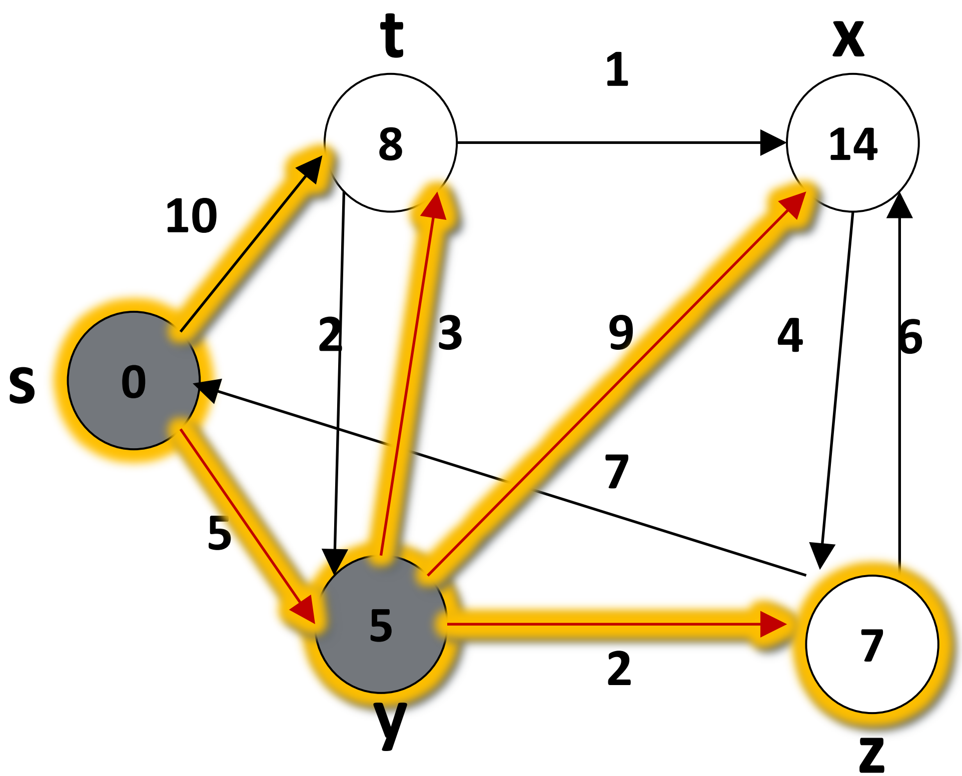
Running Example



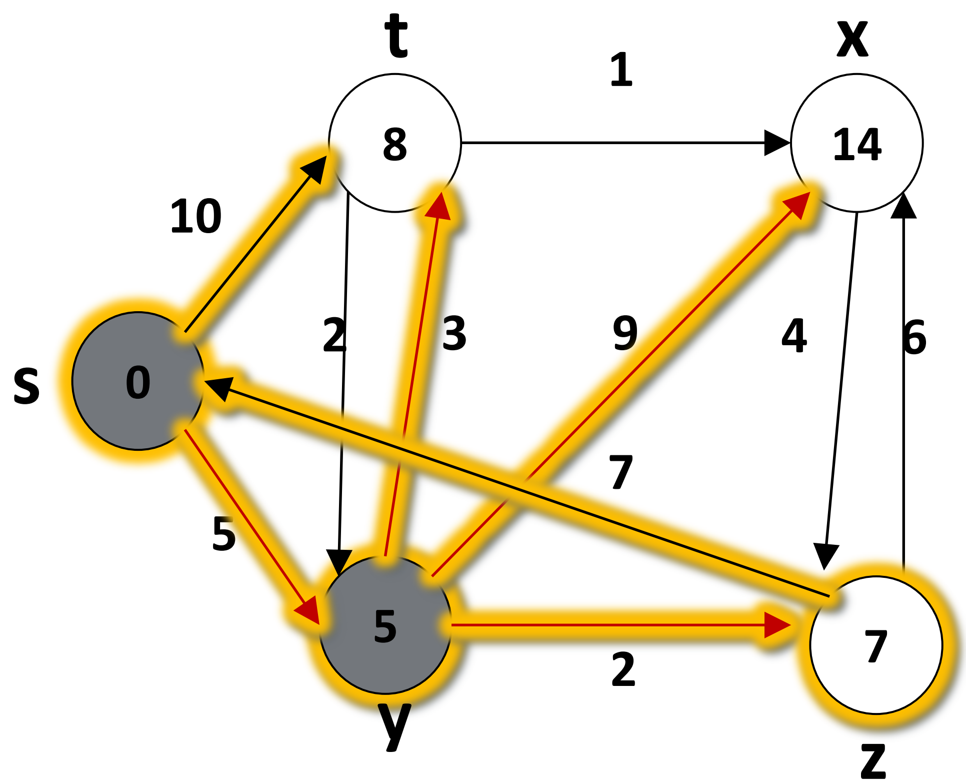
Running Example



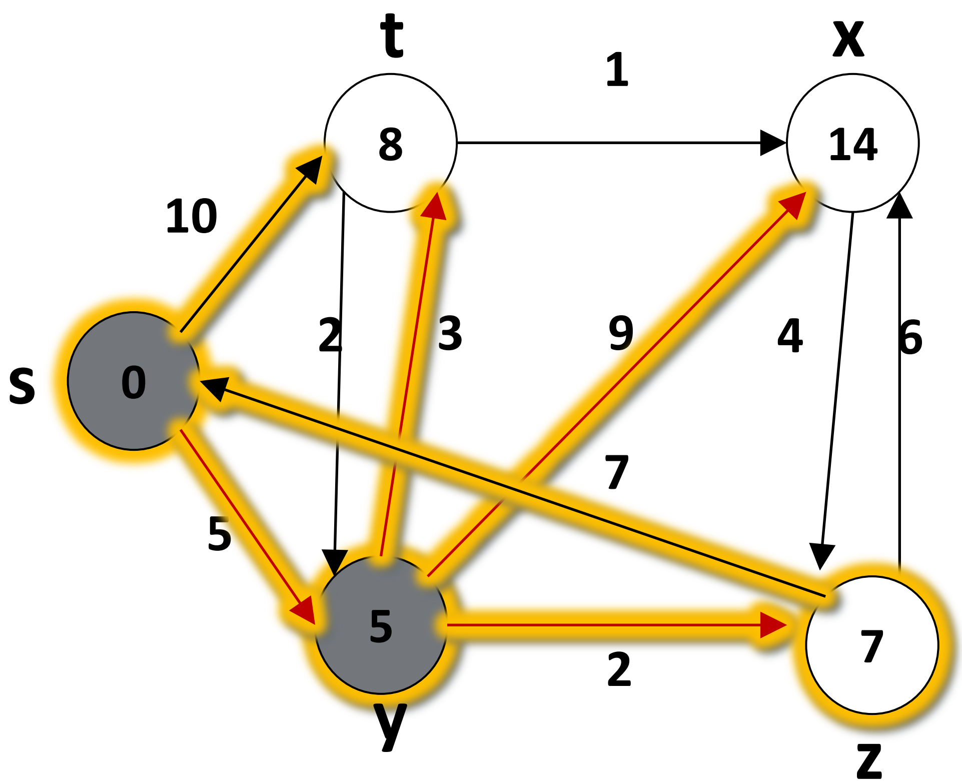
Running Example



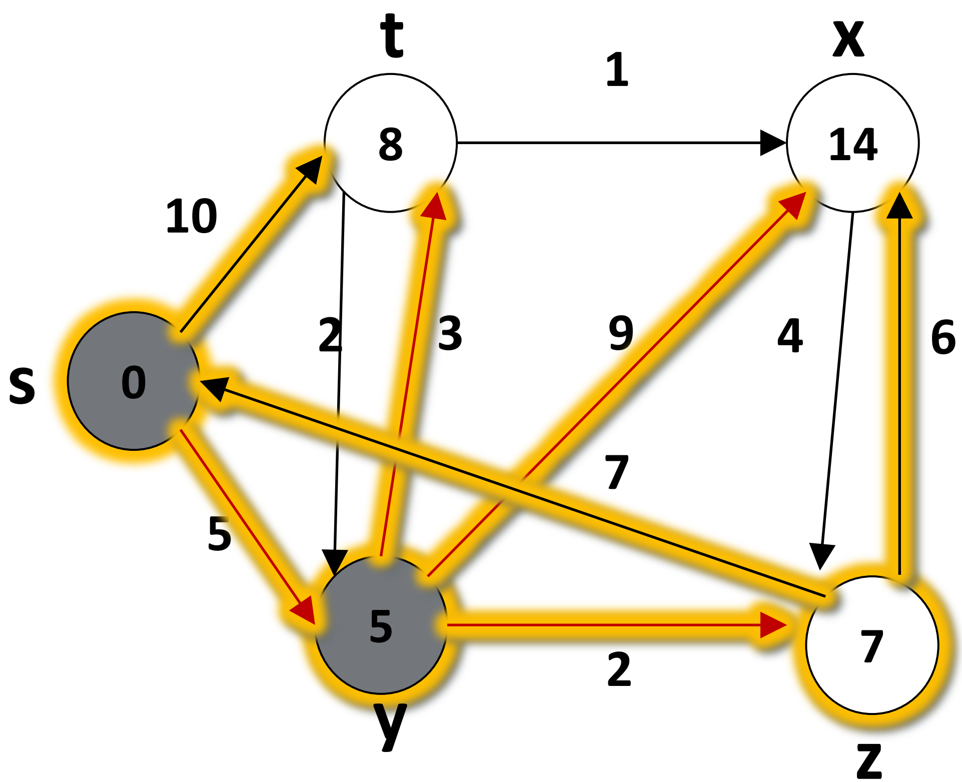
Running Example



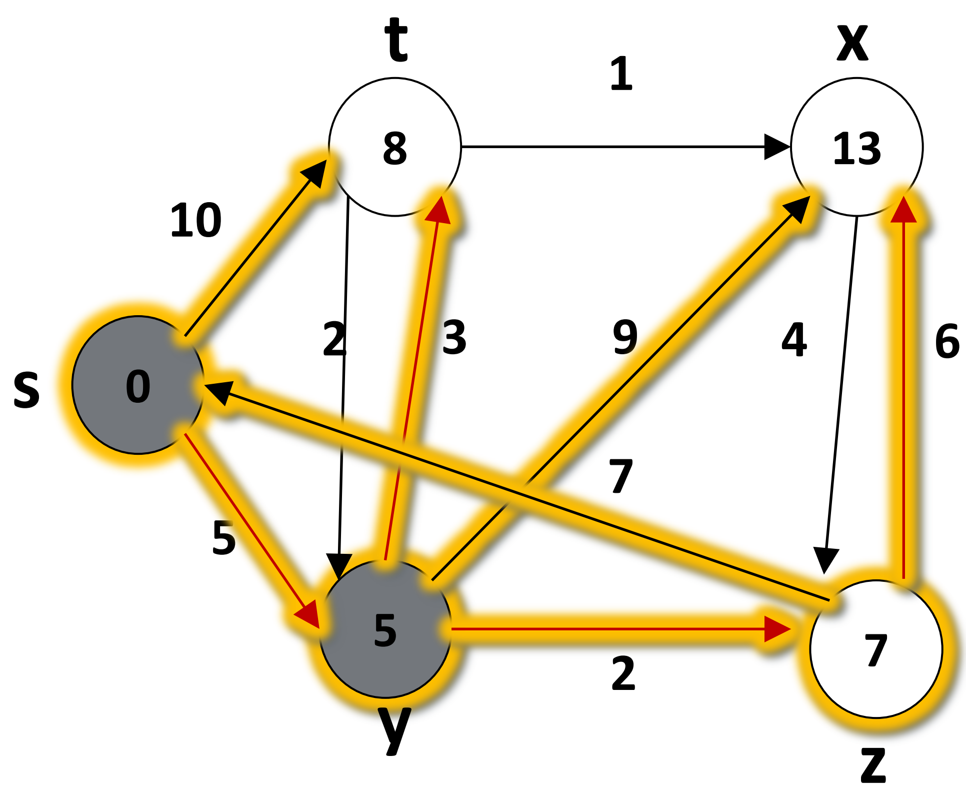
Running Example



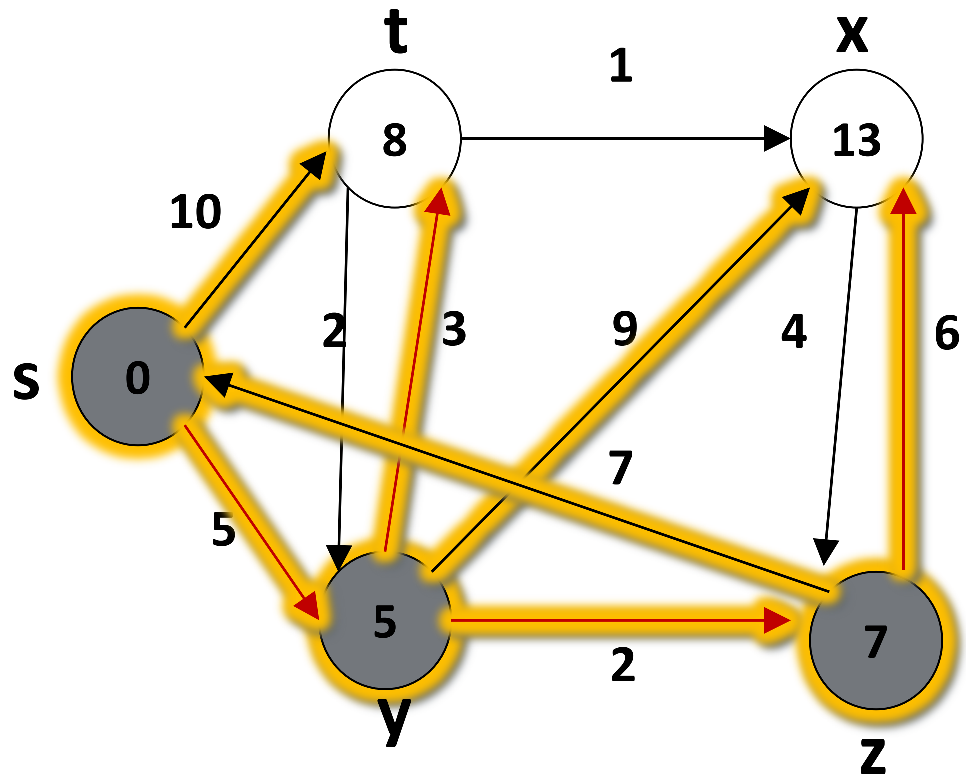
Running Example



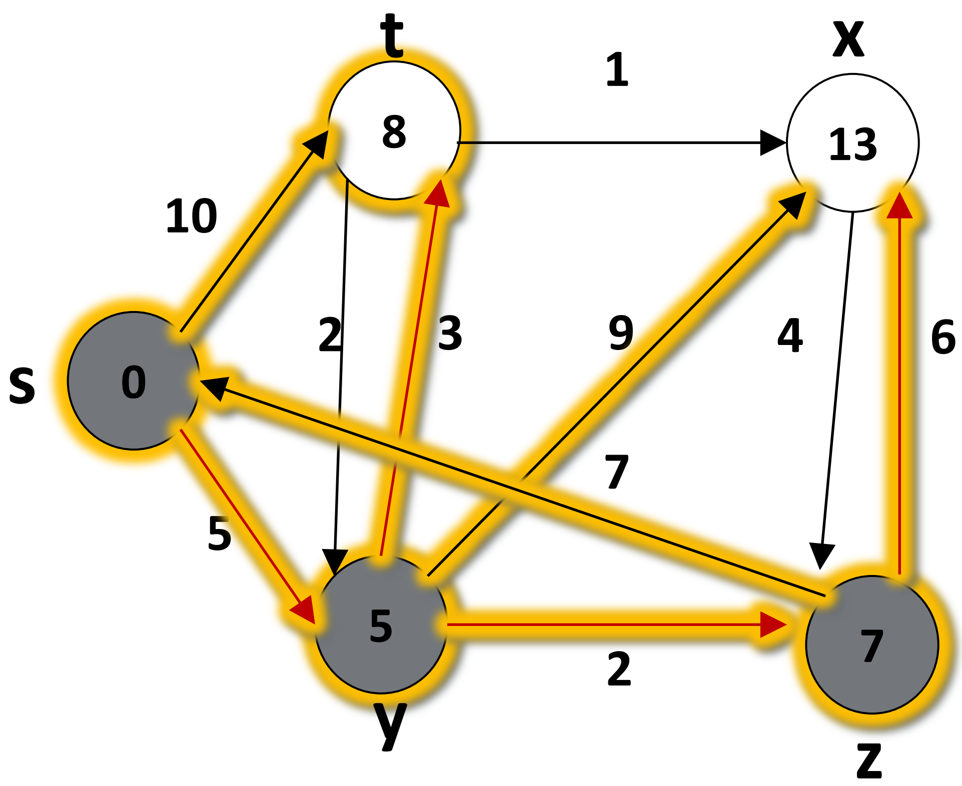
Running Example



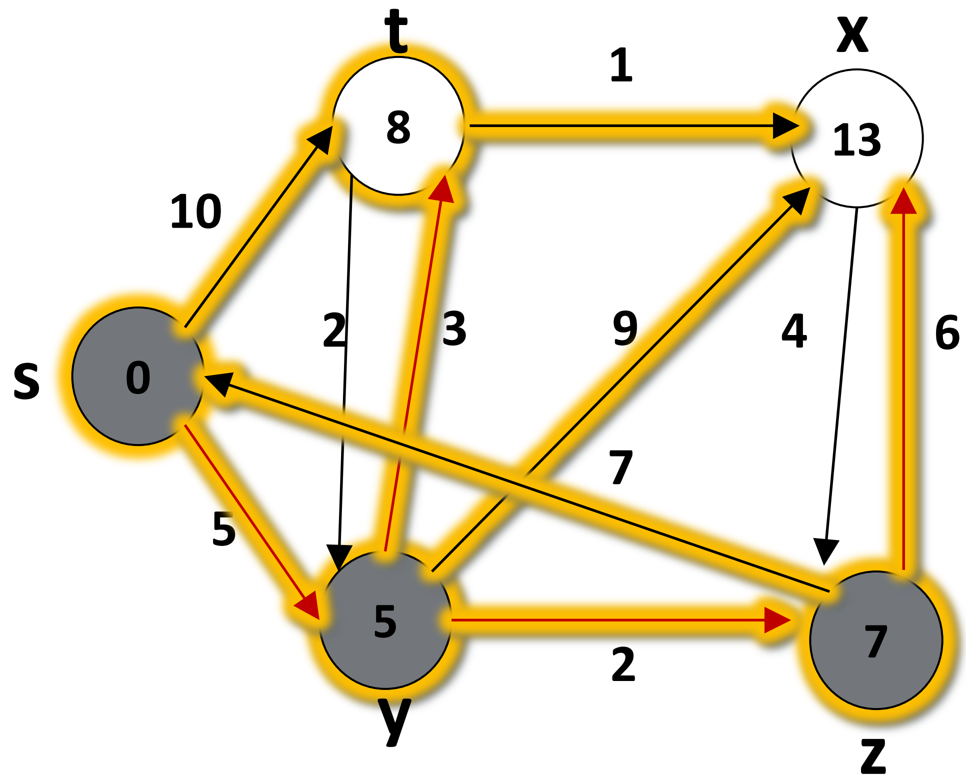
Running Example



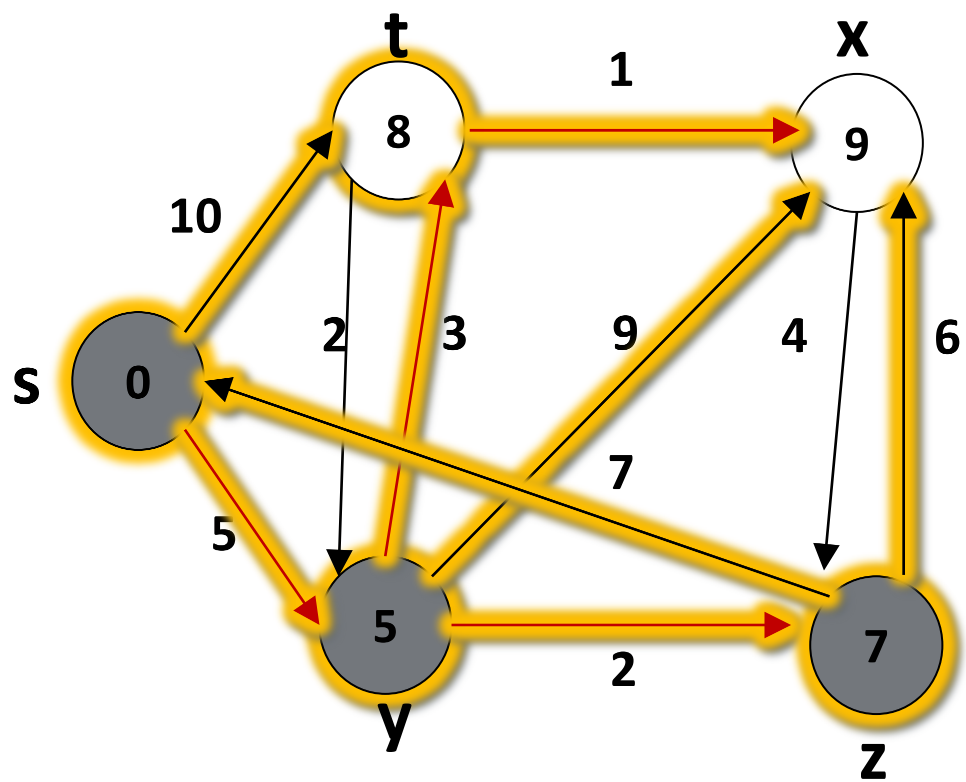
Running Example



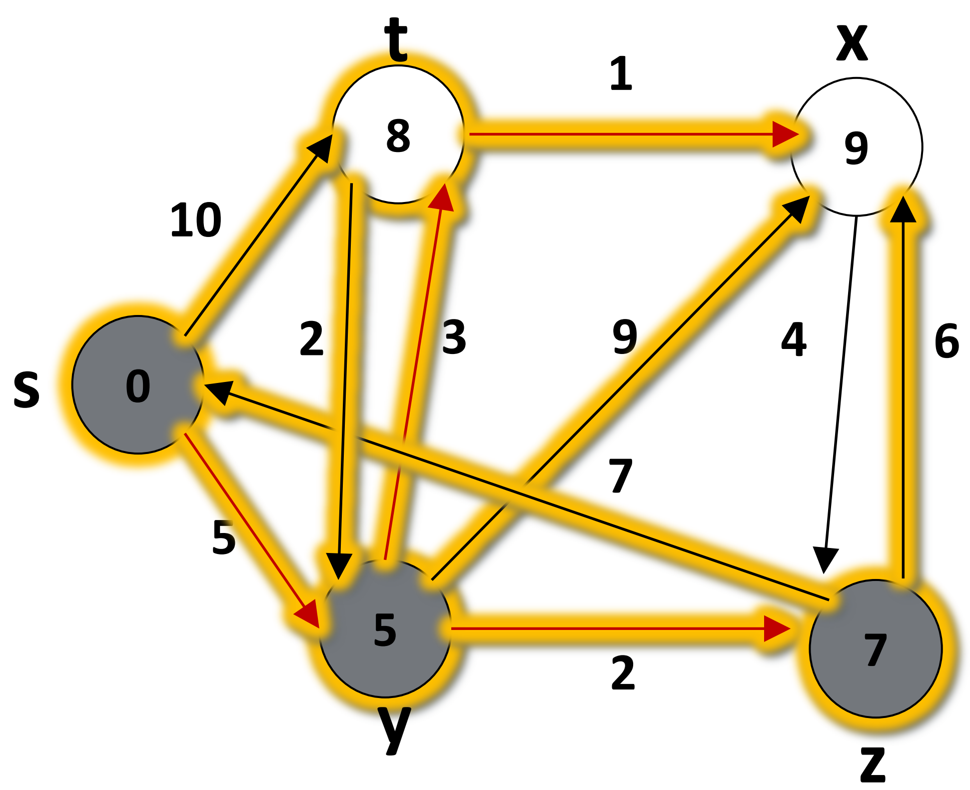
Running Example



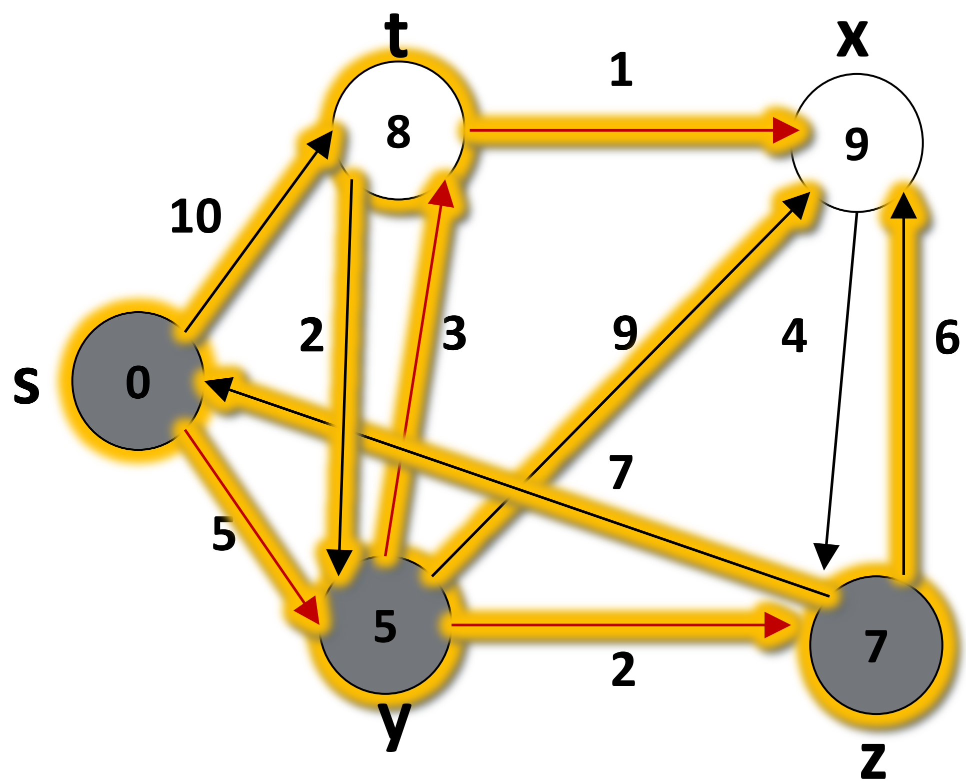
Running Example



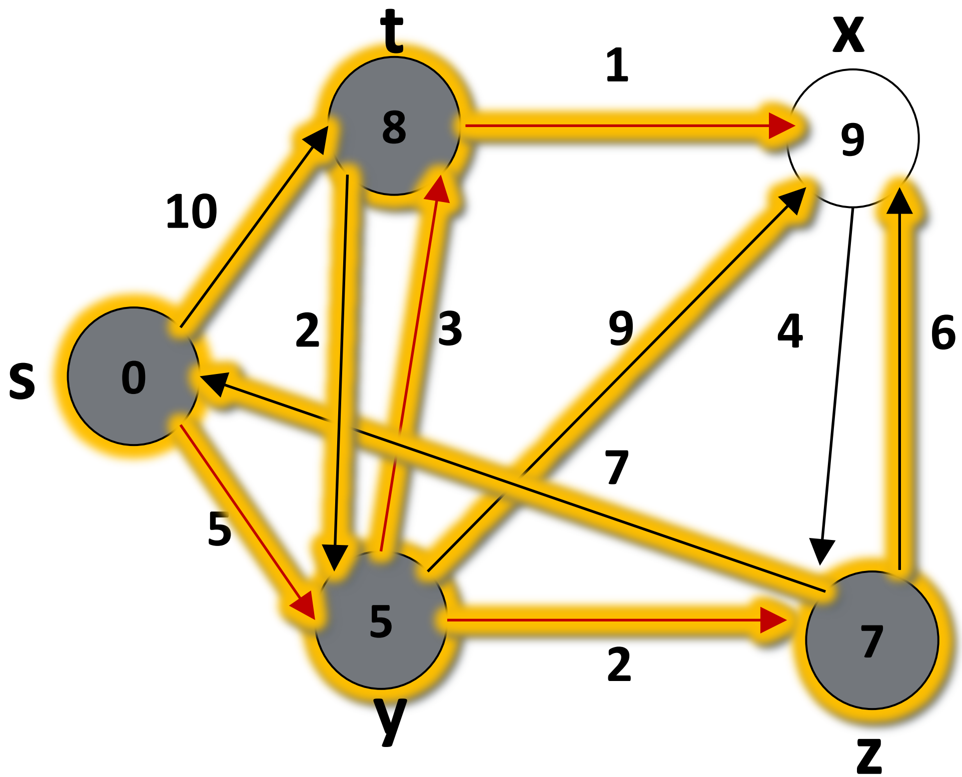
Running Example



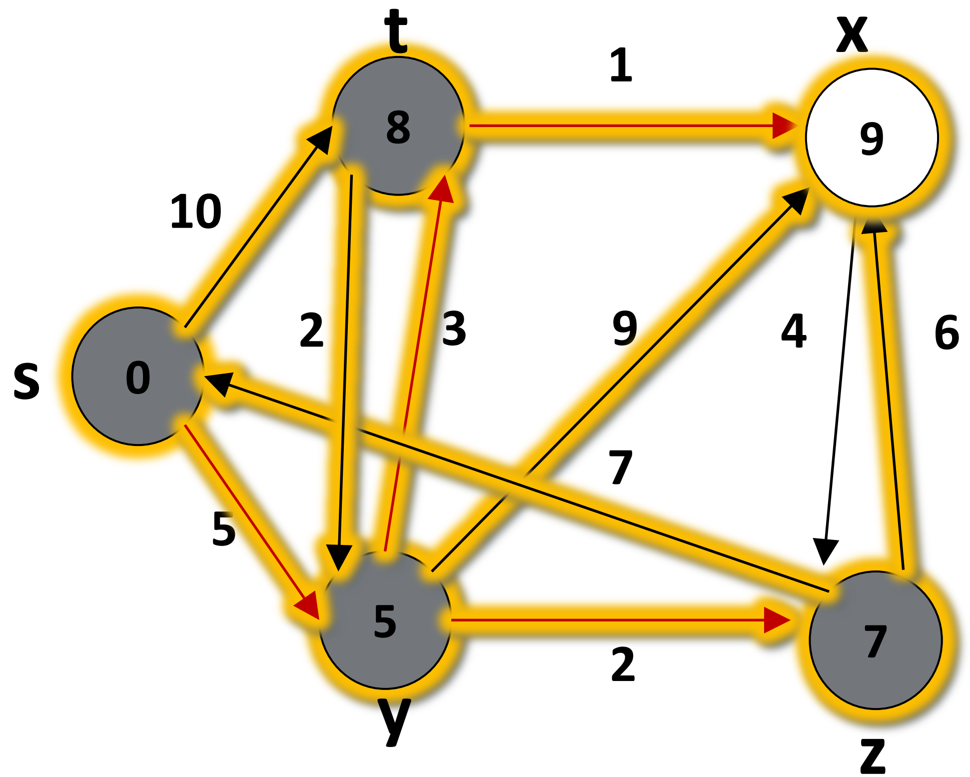
Running Example



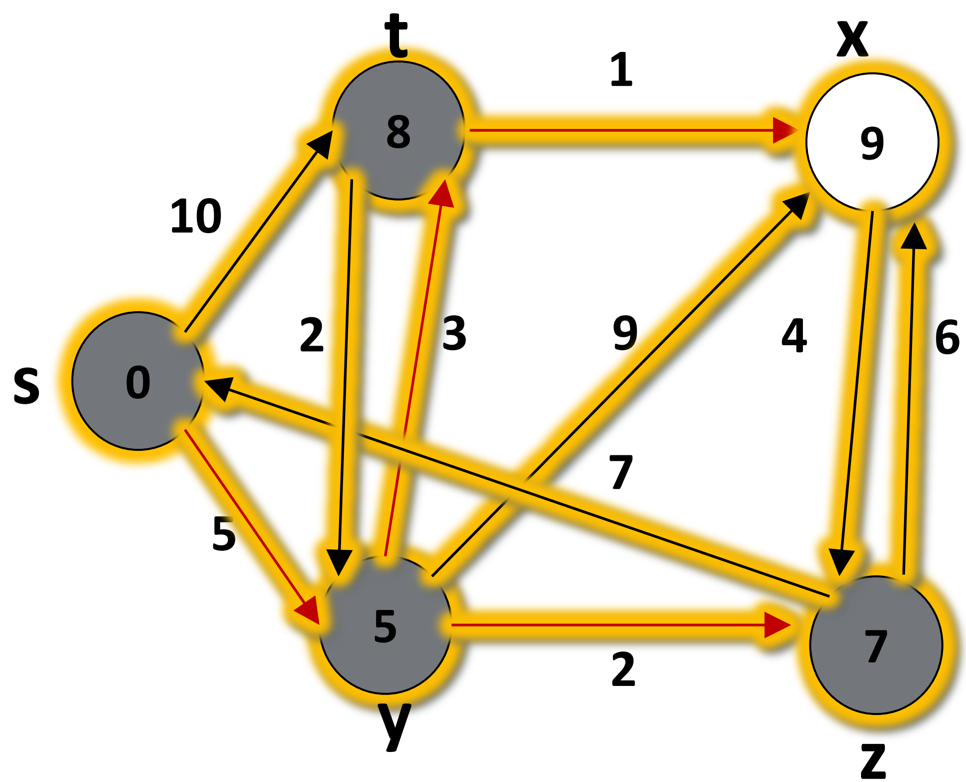
Running Example



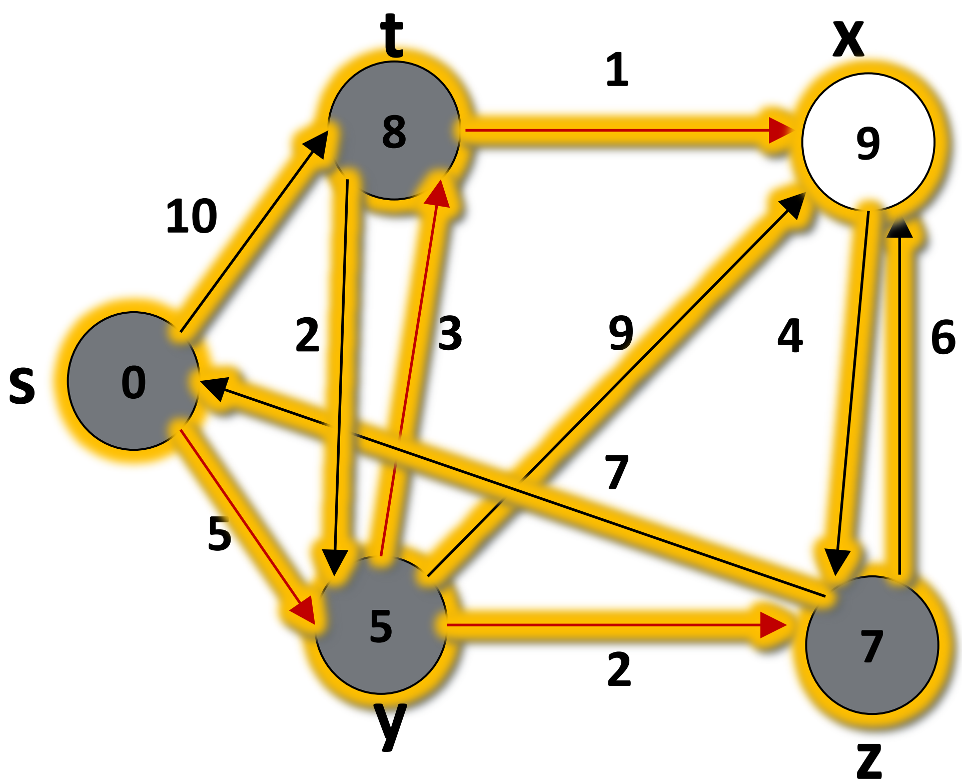
Running Example



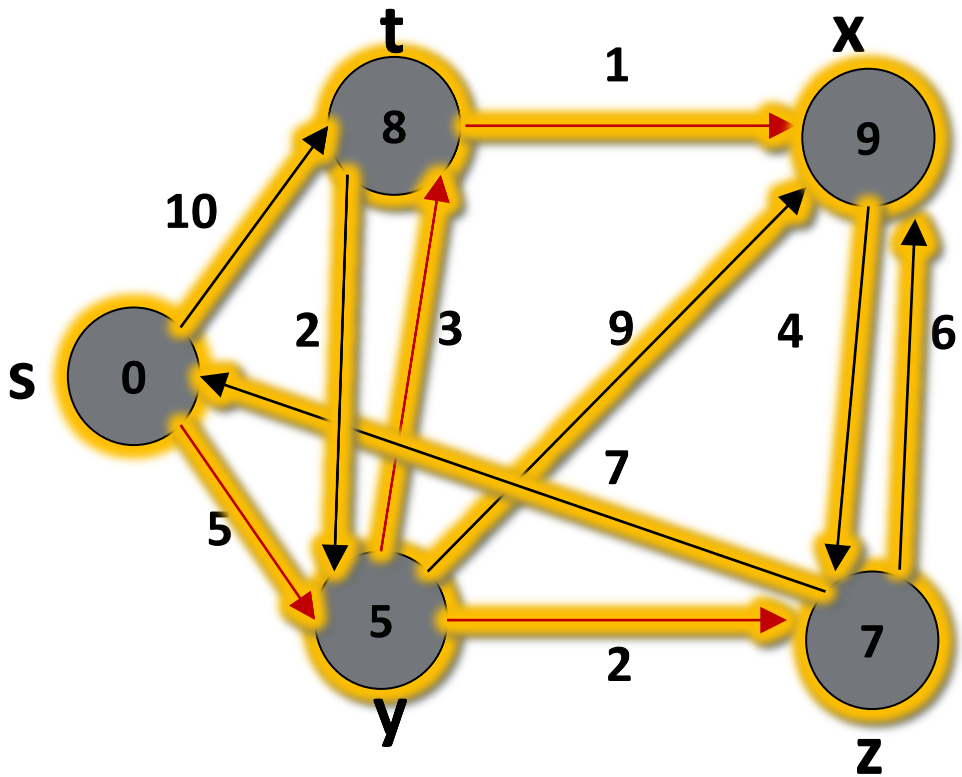
Running Example



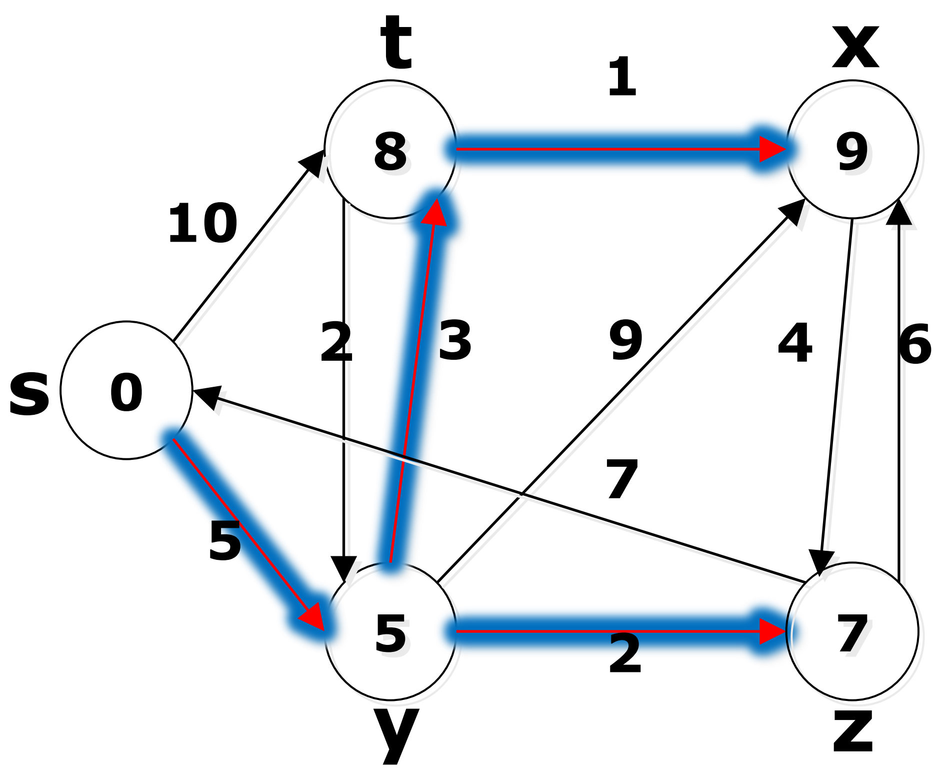
Running Example



Running Example



Running Example



Ira A. Fulton Schools of
Engineering
Arizona State University

Shortest Paths: Part 3

- Shortest Path Problem
- Dijkstra’s Shortest Path Algorithm
- Analysis

Running Time of Dijkstra's Algorithm

- **The running time depends on how we implement Q**
- **If we use a binary heap to implement the priority Q**
  - Insertion takes $O(\log|V|)$
  - Extract-Min takes $O(\log|V|)$
  - v.d := u.d+w(u, v) in Relax decrease-key: $O(\log|V|)$
  - The total running time is $O((|E|+|V|)\cdot\log|V|)$
    $= O(|E|\cdot\log|V|)$ if the graph is connected. If we use a
- **If we use Fibonacci heap to implement the priority Q**
  - The total running time is $O(|V|\cdot\log|V| + |E|)$

Fibonacci Heap Data Structure

- Fredman, Michael Lawrence; Tarjan, Robert E. "Fibonacci heaps and their uses in improved network optimization algorithms" (PDF). Journal of the Association for Computing Machinery. 34 (3): 596–615.

Correctness of Dijkstra's Algorithm

The algorithm produces correct output.

Partial Correctness

- If the preconditions are satisfied and
- if the program terminates,
- then the postconditions are satisfied.
- Core: loop invariant and induction

Termination

- If the preconditions are satisfied
- then the algorithm terminates in finite steps
- strictly decreasing order on finite set

Correctness of Dijkstra's Algorithm

Partial correctness and termination

Precondition:

weighted graph $G = (V, E, W)$ and $s \in V$ and $w \geq 0$

Postcondition:

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

Loop-invariant:

$(\forall u)\ (u \in S \Rightarrow d[u] = \delta(s, u))$

$S = V$ when exiting the loop

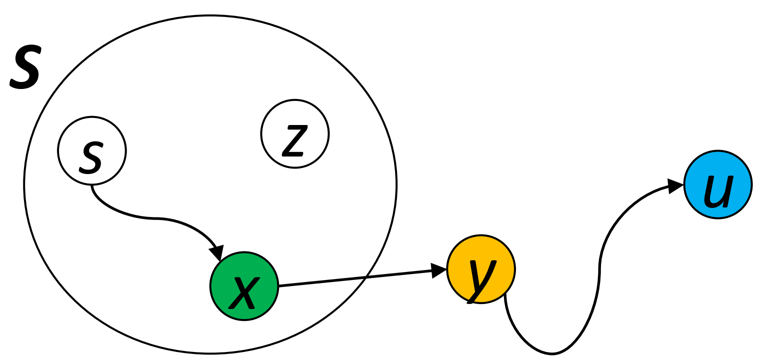
Therefore the postcondition is true.

Correctness of Dijkstra's Algorithm

- Use mathematical induction:
- Step 1: For iteration 1, $S = \{s\}$ and $d[u] = \delta(s, u)$
- Step 2: Assume that it is true when S has k elements, and we wish to prove that it is still true when another element u is added to S.
- Assume that $d[u] \neq \delta(s, u)$. We will derive a contradiction.
- Since $d[s] = \delta(s, s)=0$, $u \neq s$.
- Since u is added to S, there is a shortest path p from s to u.
- Let y be the first vertex on path p that is NOT in S.

Correctness of Dijkstra's Algorithm

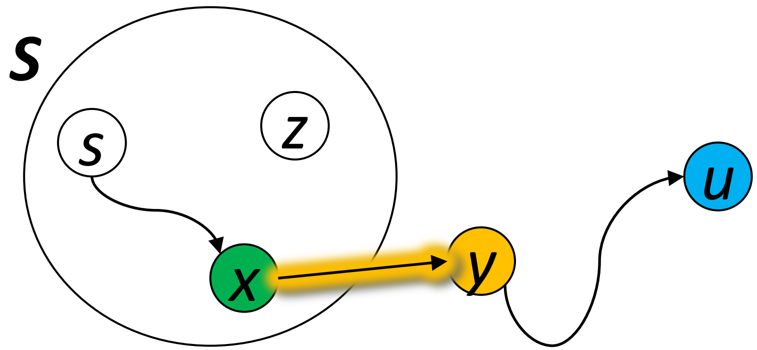
- Let y be the first vertex on path p that is not in S.
- We claim that $d[y] = \delta(s, y)$ when we add u to S.



Shortest path from s to u.

Correctness of Dijkstra's Algorithm

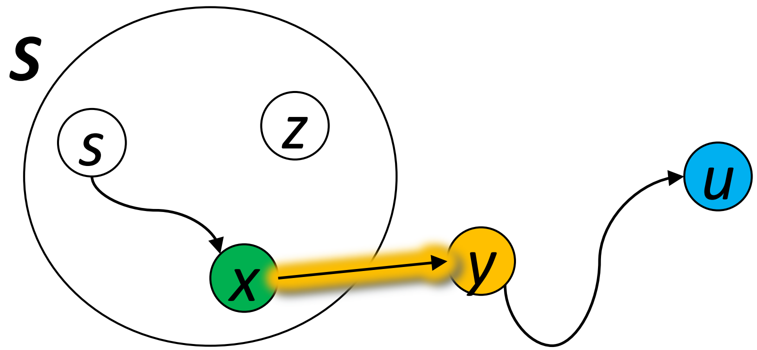
- Let y be the first vertex on path p that is not in S.
- We claim that $d[y] = \delta(s, y)$ when we add u to S.
- Let x be the predecessor of y on path p. Since y is the first vertex on p that is not in S, x is in S. At the time x is added to S, we have relaxed along edge (x, y). Hence $d[y]= \delta(s, x)+w(x, y)= \delta(s, y)$. If $y=u$, we have a contradiction. In the rest, we consider the case where $y \neq u$.



Shortest path from s to u.

Correctness of Dijkstra's Algorithm

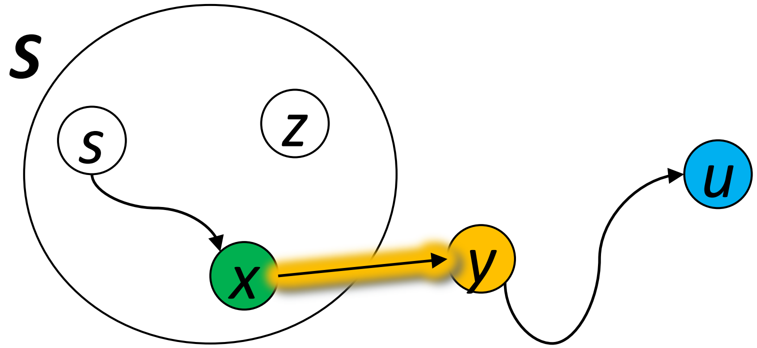
- Let y be the first vertex on path p that is not in S.
- We proved that $d[y]=\delta(s, y)$when we add u to S.
- Since y is ahead of u on path p, we have $\delta(s, y)\leq\delta(s, u)$.



Shortest path from s to u.

Correctness of Dijkstra's Algorithm

- Let y be the first vertex on path p that is not in S.
- We proved that $d[y]=\delta(s, y)$when we add u to S.
- Since y is ahead of u on path p, we have $\delta(s, y)\leq\delta(s, u)$.
- Since u is added to S before y, $d[u]\leq d[y]=\delta(s, y)$.
- Now we have $d[u]\leq d[y]=\delta(s,y)\leq\delta(s, u)\leq d[u]$.
- Therefore, **all the above inequalities must be equalities**. Hence, we have $d[u]=\delta(s, u)$. <u>This is a contradiction</u>.



Shortest path from s to u.

Dijkstra’s Algorithm on Undirected Graphs

- The algorithm works equally well on directed graphs and undirected graphs.
- Same Asymptotic time complexity.

Single-Pair Shortest Paths

- When you use a navigation system, you ask for a single-pair shortest paths, also known as s-t shortest paths, where s is the source (your current location) and t is the destination.
- We can modify Dijkstra’s algorithm slightly to compute s-t shortest paths.
- The key in this modification is: STOP as soon as the vertex t is deleted from the min-Heap.

Dijkstra’s Algorithm

Dijkstra-ST(G, w, s)

```
Initialize-Single-Source(G, s)
S = ϕ
Q = ϕ
for each vertex u ∈ G.V
    Insert(Q, u)
while Q ≠ ϕ do
        u = Extract-Min(Q)
        if (u==t) STOP
        S = S ∪ {u}
        for each vertex v ∈ u.adj do
            Relax(u, v, w)
            if the call of Relax decreases v.d
                Decrease-Key(Q, v, v.d)
```

ASU®
Ira A. Fulton Schools of
Engineering
Arizona State University