Single-source shortest path with positive weights

Letícia Rodrigues Bueno

Federal University of ABC (UFABC)

Problem: Shortest path between two cities

- Given a map of cities and their distances between each other, what is the shortest path between any two cities A and B?

Problem: Shortest path between two cities

- Given a map of cities and their distances between each other, what is the shortest path between any two cities A and B?
- This problem can be modeled as a graph:
  - **City:** vertices;
  - **Roads between cities:** weighted edges representing the distance between cities.
  - **Weighted Edges:** can indicate time, cost, penalties, losses, etc;
- This problem is known as **Shortest Path Problem.**

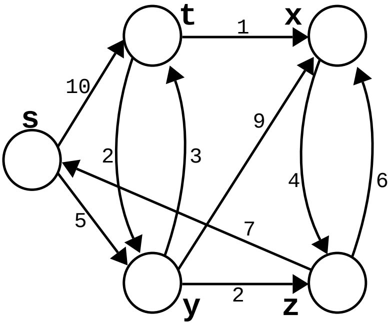
Approach

- Shortest path problem is similar to Erdös number problem;

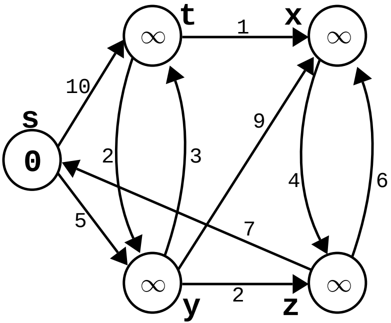
Approach

- Shortest path problem is similar to Erdös number problem;
- We can use a similar idea to the **Breadth-first search**;
- Important to consider:
  - the problem has **optimal substructure**: which means that a shortest path between two vertices can be constructed from the shortest paths of subproblems.

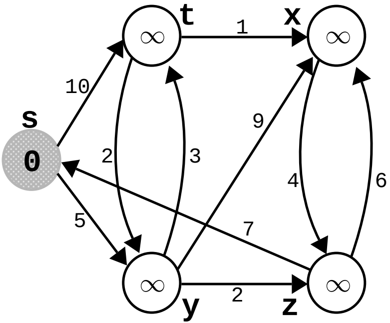
Single-source shortest path: Dijkstra's algorithm



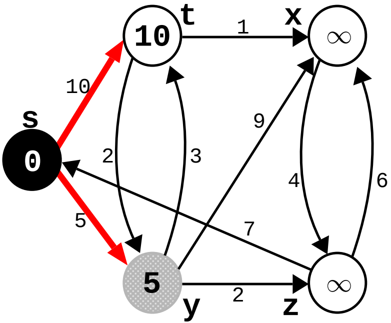
Single-source shortest path: Dijkstra's algorithm



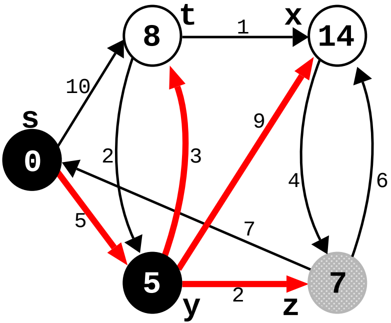
Single-source shortest path: Dijkstra's algorithm



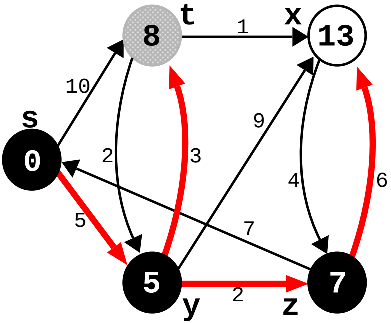
Single-source shortest path: Dijkstra’s algorithm



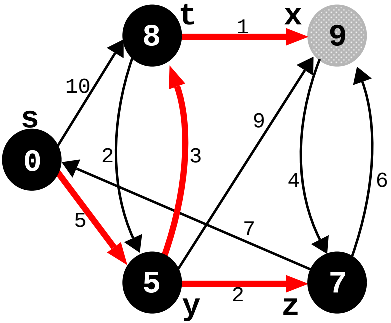
Single-source shortest path: Dijkstra's algorithm



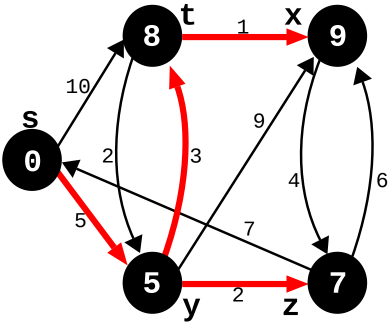
Single-source shortest path: Dijkstra's algorithm



Single-source shortest path: Dijkstra's algorithm

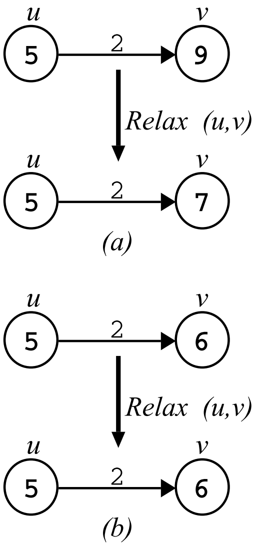


Single-source shortest path: Dijkstra's algorithm



Single-source shortest path: Dijkstra's algorithm

```
relax(u, v):
  if v.d > u.d + p(u, v) then
    v.d = u.d + p(u, v)
    v.p = u
```



Single-source shortest path: Dijkstra's algorithm

```
dijkstra(G, s):
   for u in V(G) do
       u.d = ∞
       u.p = None
   s.d = 0
   s.p = s
   A = Heap(V(G)) \\ using value of d
   S = [ ]
   while size(A) > 1 do
       u = remove_min(A)
       S = S + u
       for v in adj(u) do
           relax(u, v)
           redo_heap(A)
```

Single-source shortest path: Dijkstra's algorithm

```
dijkstra(G, s):
   for u in V(G) do
      u.d = ∞
      u.p = None
   s.d = 0
   s.p = s
   A = Heap(V(G)) \\ using value of d
   S = [ ]
   while size(A) > 1 do
      u = remove_min(A)
      S = S + u
      for v in adj(u) do
         relax(u, v)
         redo_heap(A)
```

Analysis of complexity:

Single-source shortest path: Dijkstra's algorithm

```
1   dijkstra(G, s):
2     for u in V(G) do
3         u.d = ∞
4         u.p = None
5     s.d = 0
6     s.p = s
7     A = Heap(V(G)) \\ using value of d
8     S = [ ]
9     while size(A) > 1 do
10        u = remove_min(A)
11        S = S + u
12        for v in adj(u) do
13            relax(u, v)
14            redo_heap(A)
```

Analysis of complexity:

- Construction of *heap* (line 6): $O(n)$;

Single-source shortest path: Dijkstra's algorithm

```
1   dijkstra(G, s):
2     for u in V(G) do
3       u.d = ∞
4       u.p = None
5     s.d = 0
6     s.p = s
7     A = Heap(V(G)) \\ using value of d
8     S = [ ]
9     while size(A) > 1 do
10      u = remove_min(A)
11      S = S + u
12      for v in adj(u) do
13        relax(u, v)
14        redo_heap(A)
```

Analysis of complexity:

- Construction of *heap* (line 6): $O(n)$;
- remove_min (line 9) has complexity $(\log n)$ and is executed n times: $O(n \log n)$;

Single-source shortest path: Dijkstra's algorithm

```
1   dijkstra(G, s):
2     for u in V(G) do
3       u.d = ∞
4       u.p = None
5     s.d = 0
6     s.p = s
7     A = Heap(V(G)) \\ using value of d
8     S = [ ]
9     while size(A) > 1 do
10      u = remove_min(A)
11      S = S + u
12      for v in adj(u) do
13        relax(u, v)
14        redo_heap(A)
```

Analysis of complexity:

- Construction of *heap* (line 6): $O(n)$;
- remove_min (line 9) has complexity $(\log n)$ and is executed n times: $O(n \log n)$;
- redo_heap (line 13) has complexity $(\log n)$ and is executed m times: $O(m \log n)$;

Single-source shortest path: Dijkstra’s algorithm

```
1   dijkstra(G, s):
2     for u in V(G) do
3       u.d = ∞
4       u.p = None
5     s.d = 0
6     s.p = s
7     A = Heap(V(G)) \\ using value of d
8     S = [ ]
9     while size(A) > 1 do
10      u = remove_min(A)
11      S = S + u
12      for v in adj(u) do
13        relax(u, v)
14        redo_heap(A)
```

Analysis of complexity:

- Construction of *heap* (line 6): $O(n)$;
- remove_min (line 9) has complexity $(\log n)$ and is executed n times: $O(n \log n)$;
- redo_heap (line 13) has complexity $(\log n)$ and is executed m times: $O(m \log n)$;
- Total complexity: $O((n + m) \log n)$.

Shortest Path Problems

- **Single-source shortest path:** Dijkstra's algorithm;

Shortest Path Problems

- **Single-source shortest path:** Dijkstra's algorithm;
- **Single-destination shortest path:** invert direction of edges and then apply Dijstra's algorithm;

Shortest Path Problems

- **Single-source shortest path:** Dijkstra’s algorithm;
- **Single-destination shortest path:** invert direction of edges and then apply Dijstra’s algorithm;
- **Shortest path between any vertices u and v:** Dijkstra’s algorithm;

Shortest Path Problems

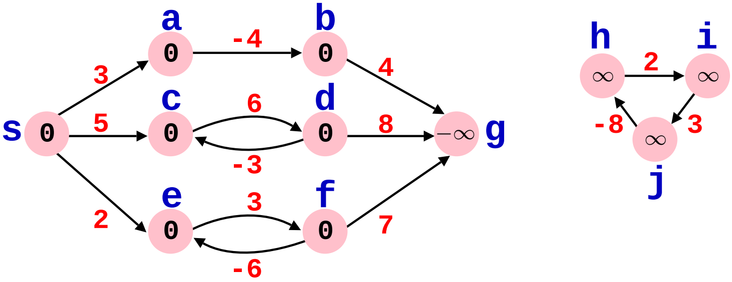
- **Single-source shortest path:** Dijkstra's algorithm;
- **Single-destination shortest path:** invert direction of edges and then apply Dijstra's algorithm;
- **Shortest path between any vertices u and v:** Dijkstra's algorithm;
- **Shortest path from all vertices to all vertices:** Floyd-Warshall's algorithm in time $O(n^3)$.

Limitations of Dijkstra's algorithm

- It does not work for graphs containing cycles with negative weights that are reachable from the source;

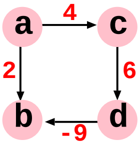
Limitations of Dijkstra's algorithm

- It does not work for graphs containing cycles with negative weights that are reachable from the source;
- **Option:** Bellman-Ford's algorithm with complexity $O(n \cdot m)$;



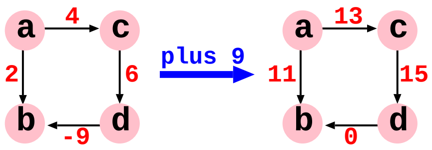
Shortest Path in Graphs with Negative Weights

- What if we add a constant to the weight of each edge in order to make all weights positive?



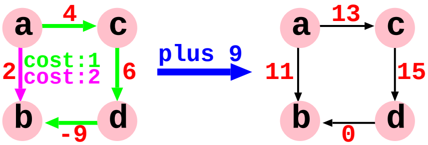
Shortest Path in Graphs with Negative Weights

- What if we add a constant to the weight of each edge in order to make all weights positive?



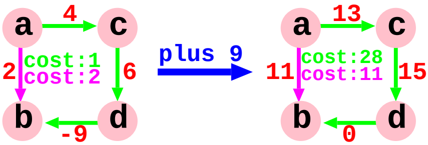
Shortest Path in Graphs with Negative Weights

- What if we add a constant to the weight of each edge in order to make all weights positive?



Shortest Path in Graphs with Negative Weights

- What if we add a constant to the weight of each edge in order to make all weights positive?



Bibliography

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. and STEIN, C. *Introduction to Algorithms*, 3rd edition, MIT Press, 2009.