CSE331 Introduction to Algorithms
Lecture 16: Dijkstra's Algorithm

Antoine Vigneron
antoine@unist.ac.kr

Ulsan National Institute of Science and Technology

July 11, 2017

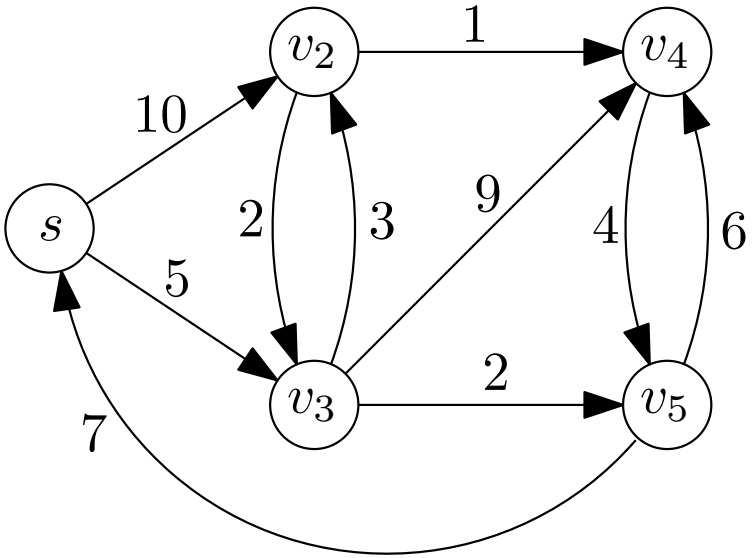
Introduction

- In Lecture 13, we saw that shortest paths in an *unweighted* graph can be computed by BFS in $O(n + m)$ time.
- This lecture presents Dijkstra's algorithm for computing shortest paths in a *weighted* graph.
- (Both algorithms work for directed and undirected graphs.)
- Dijkstra's algorithm can be seen as a greedy algorithm, or a dynamic programming approach.
- **Reference**: Section 24.3 of the textbook (p 658–)

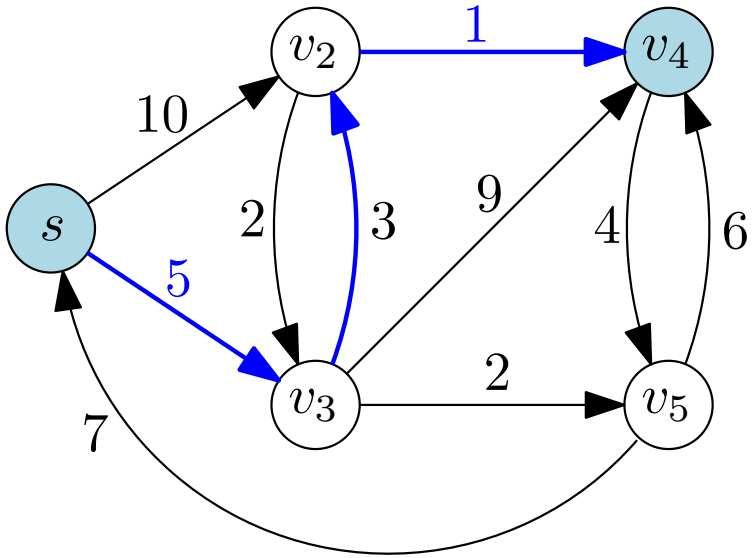
  Introduction to Algorithms by Cormen, Leiserson, Rivest and Stein.

- I will not be following this textbook closely in this lecture.

Introduction



Input graph



Shortest path from s to v_4

- The *distance* $d(s, v_4)$ between s and v_4 is the length of the shortest path from s to v_4. So $d(s, v_4) = 9$.

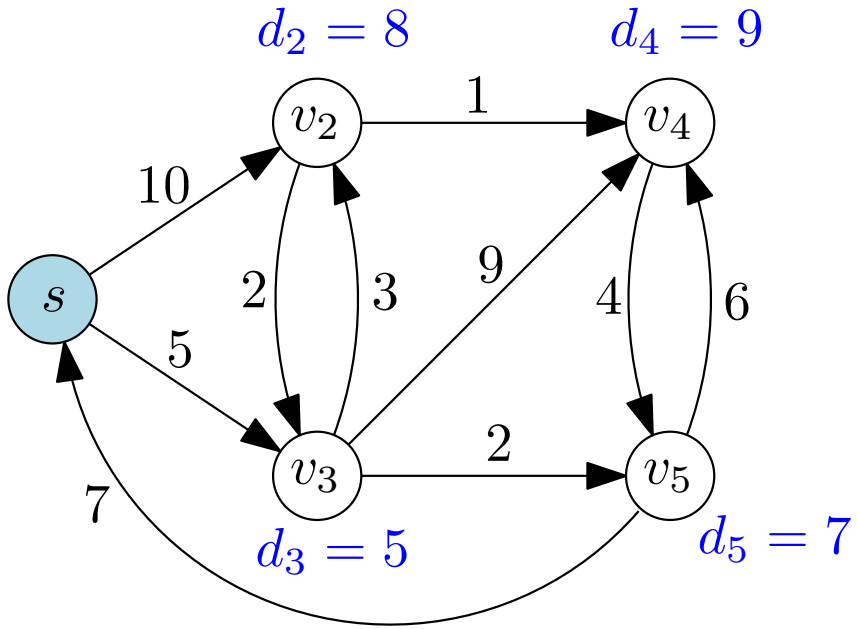
Problem Statement

Problem (Single-source shortest path)

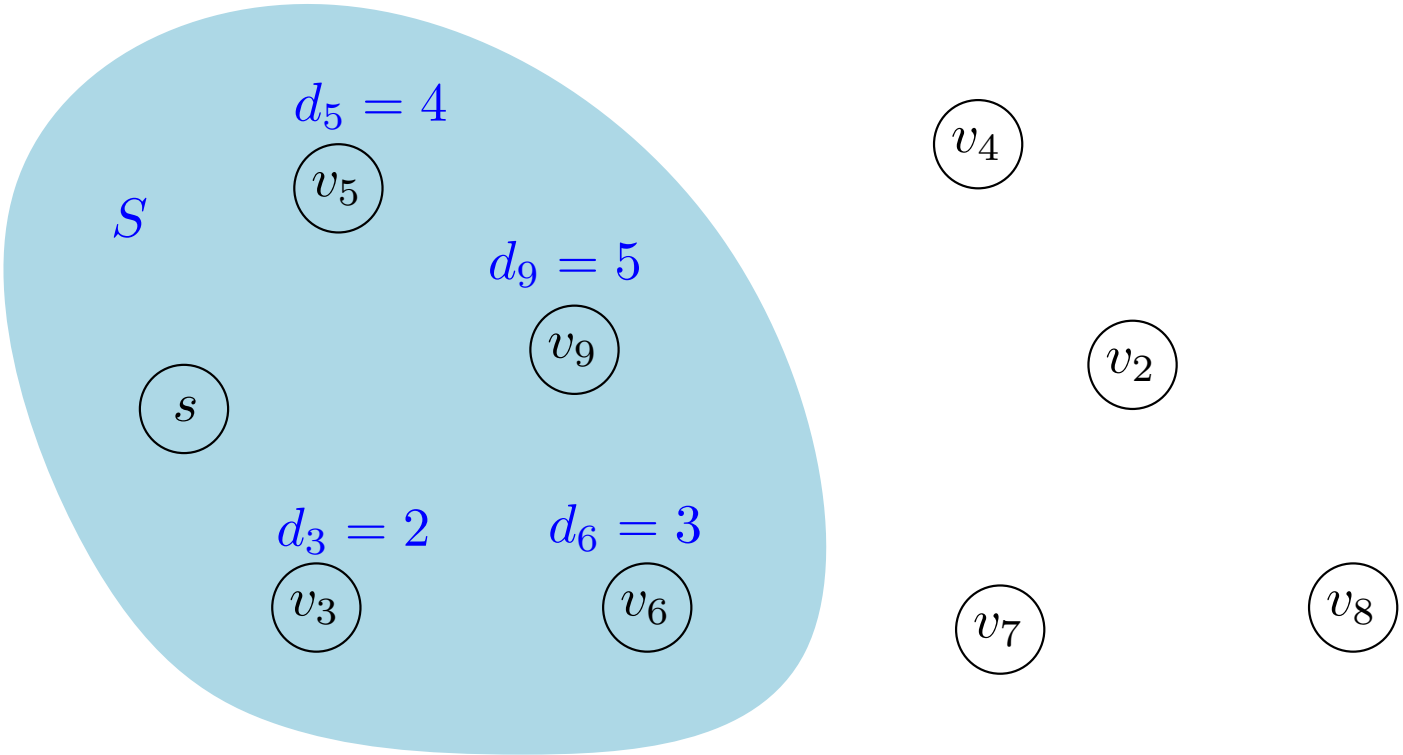
INPUT: A positively weighted, directed graph $G(V, E)$ with vertex set $V = \{v_1, v_2, \ldots, v_n\}$, where the *source* node is $s = v_1$. Each edge (v_i, v_j) has a weight $\ell(v_i, v_j) > 0$ called its *length*.

OUTPUT: The shortest path from s to all other nodes v_i, $i = 2, 3, \ldots, n$.

- We will show how to compute the distances $d_i = d(s, v_i)$ from s to all the other vertices.

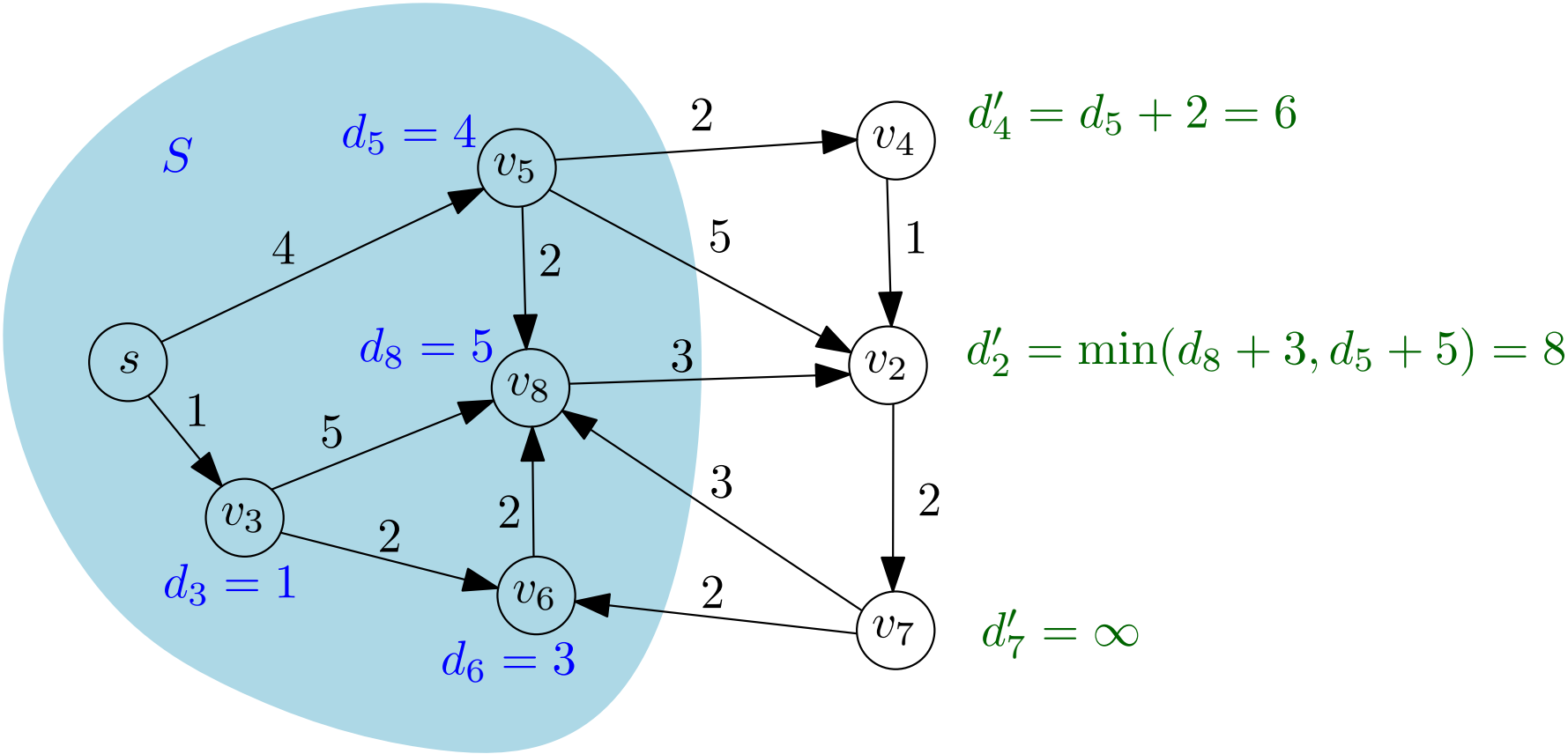


Dijkstra's Algorithm



- During the course of the algorithm, we expand a set S of vertices for which we know the distance from S, i.e. we know d_i for all $v_i \in S$.

Dijkstra's Algorithm



- For each $v_j \notin S$, we maintain the length d'_j of the shortest path from s such that the last edge starts in S. If there is no edge from S to v_j, d'_j is set to ∞.

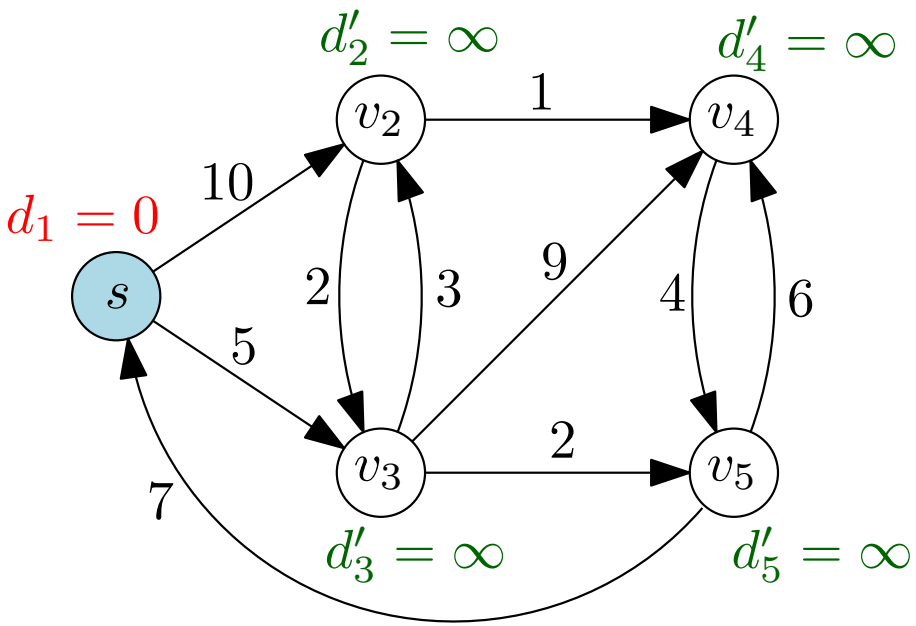
Dijkstra's Algorithm

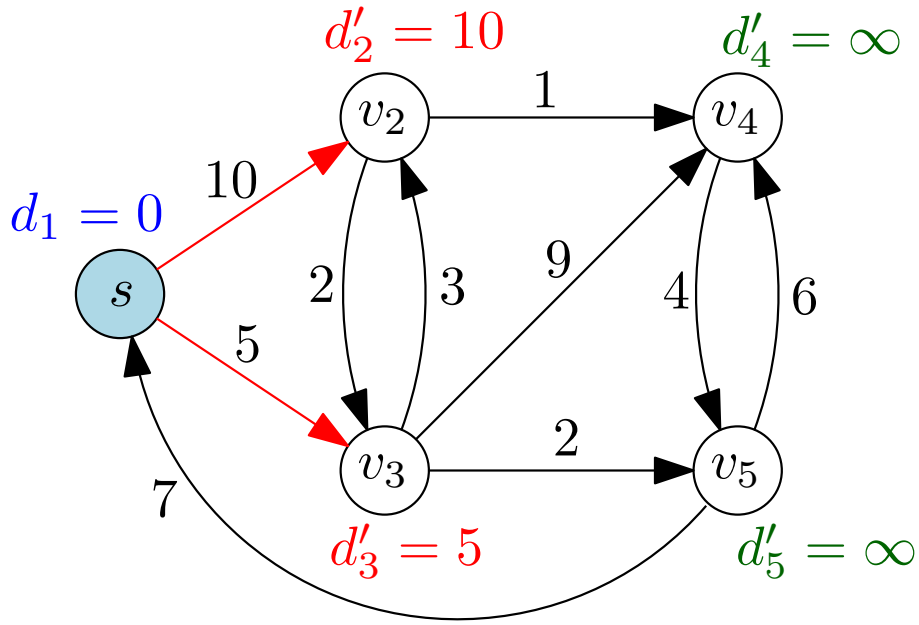
- The algorithm maintains a set S of *explored* vertices containing s.
- For each $v_i \in S$, we know the length of the shortest path $d_i = d(s, v_i)$ from the source.
- For each $v_j \notin S$ such that there is an edge from a vertex $v_i \in S$ to v_j, we know

$$d'_j = \min_{v_i \in S : (v_i, v_j) \in E} d_i + \ell(v_i, v_j),$$

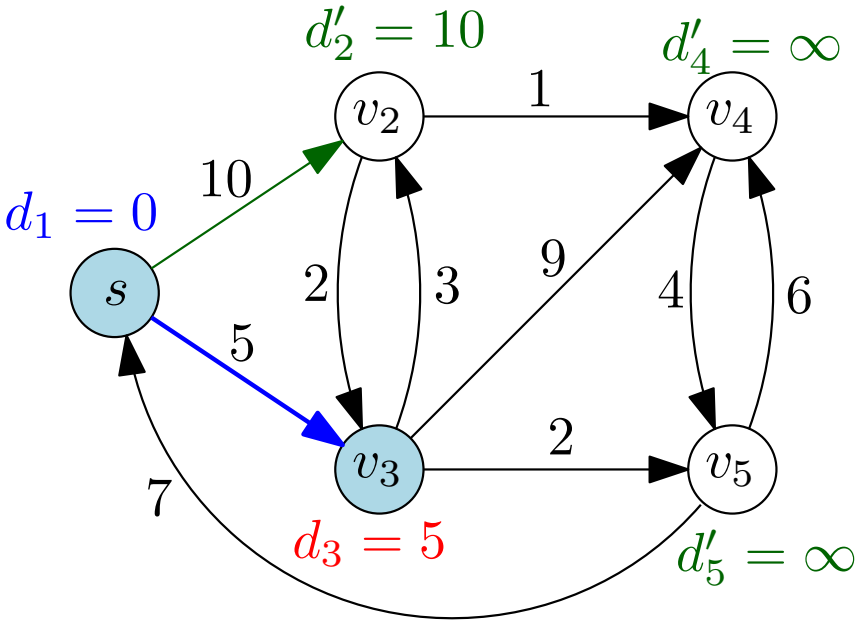
  and if there is no such edge, $d'_j = \infty$.
- At each iteration, we insert into S a node $v_i \notin S$ such that d'_i is minimum. We will see that in this case, we have $d_i = d'_i$. We then need to update other values d'_j.

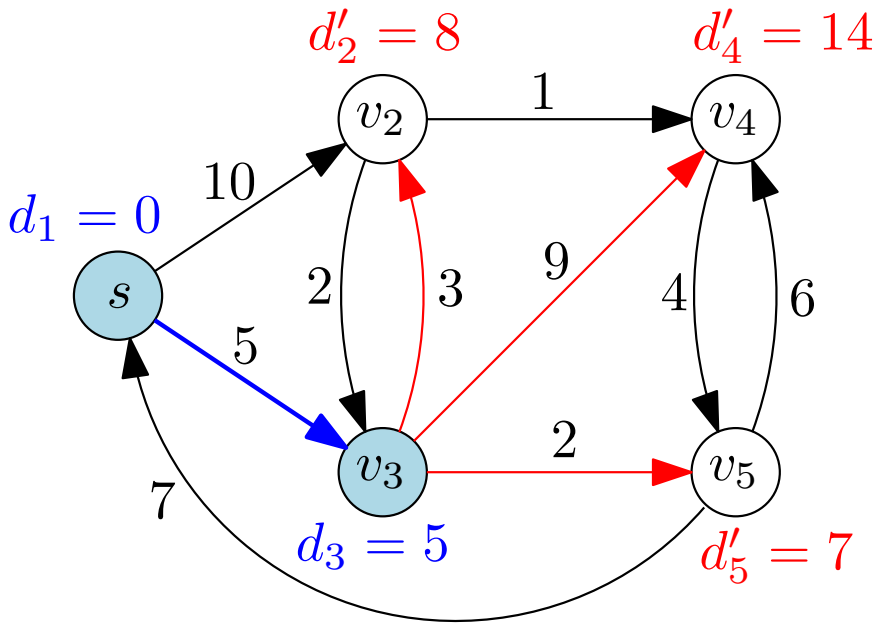
Example


$d'_2 = \infty$
$d'_4 = \infty$
v_2
v_4
1
10
$d_1 = 0$
s
2
3
9
4
6
5
v_3
v_5
2
7
$d'_3 = \infty$
$d'_5 = \infty$

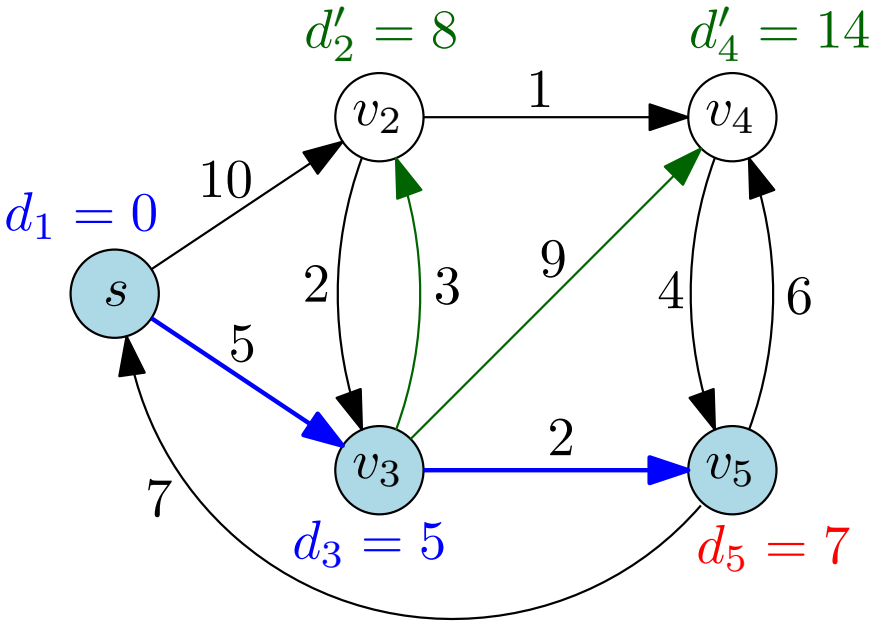

$d'_2 = 10$
$d'_4 = \infty$
v_2
v_4
1
10
$d_1 = 0$
s
2
3
9
4
6
5
v_3
v_5
2
7
$d'_3 = 5$
$d'_5 = \infty$

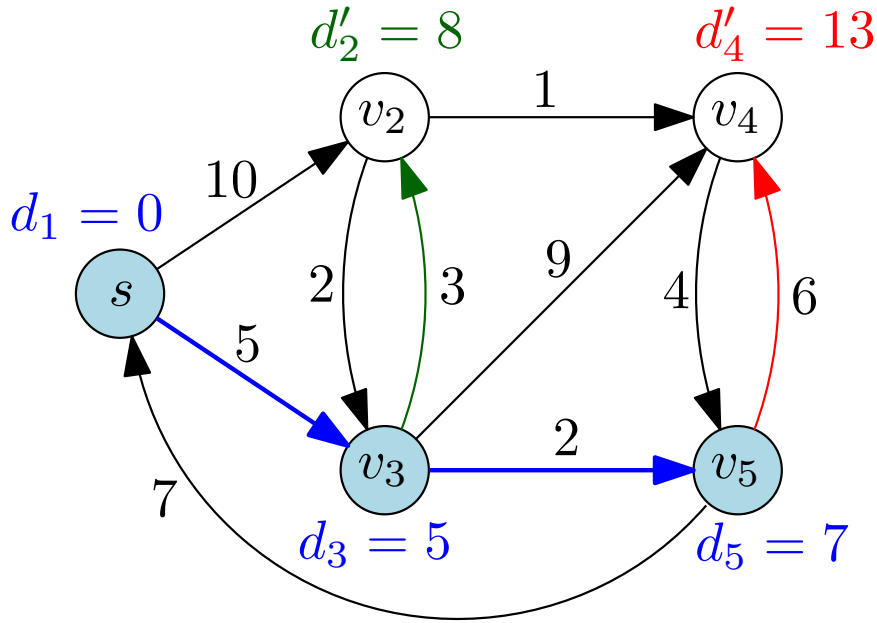
Example


$d'_2 = 10$
$d'_4 = \infty$
v_2
v_4
1
10
$d_1 = 0$
s
2
3
9
4
6
5
v_3
2
v_5
7
$d_3 = 5$
$d'_5 = \infty$

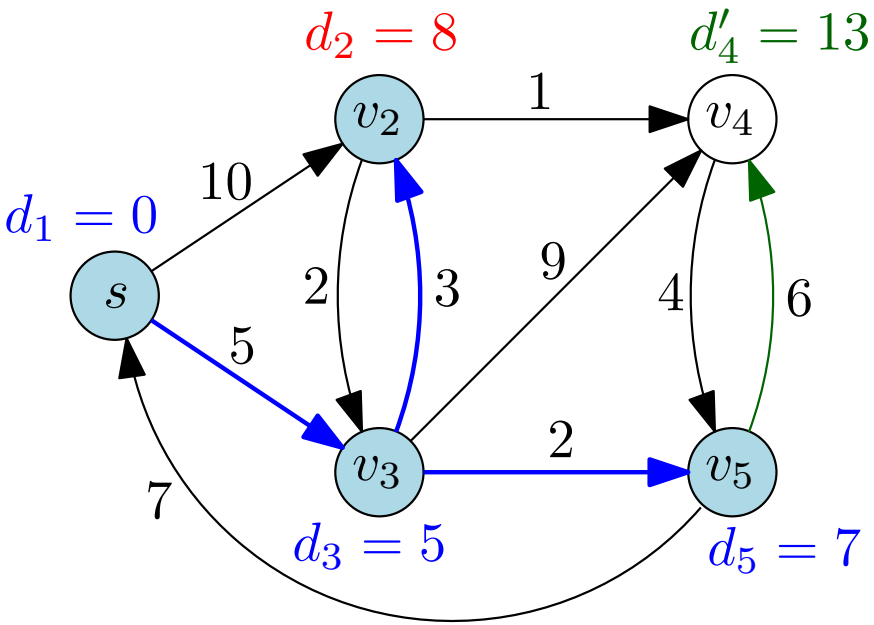

$d'_2 = 8$
$d'_4 = 14$
v_2
v_4
1
10
$d_1 = 0$
s
2
3
9
4
6
5
v_3
2
v_5
7
$d_3 = 5$
$d'_5 = 7$

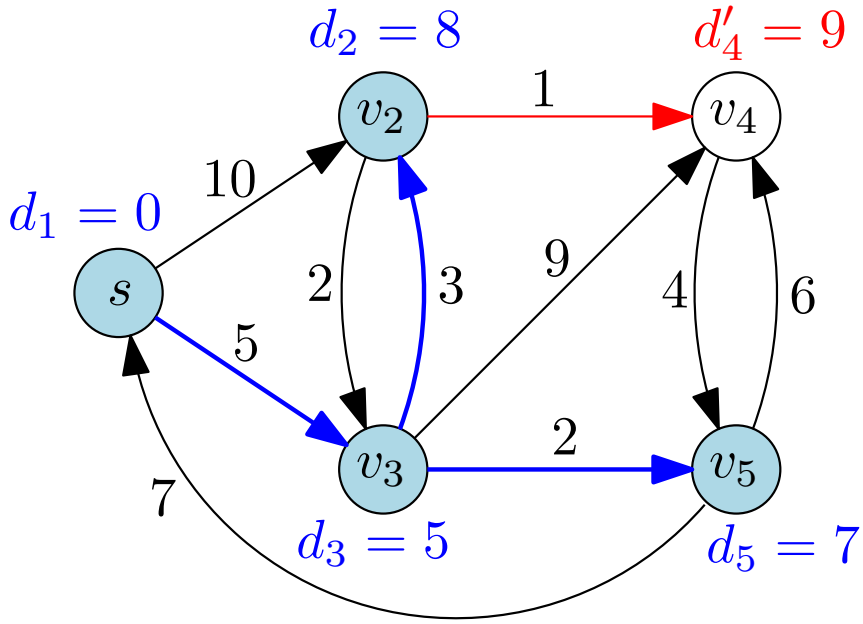
Example


$d'_2 = 8$
$d'_4 = 14$
v_2
v_4
1
10
$d_1 = 0$
s
2
3
9
4
6
5
v_3
v_5
2
7
$d_3 = 5$
$d_5 = 7$

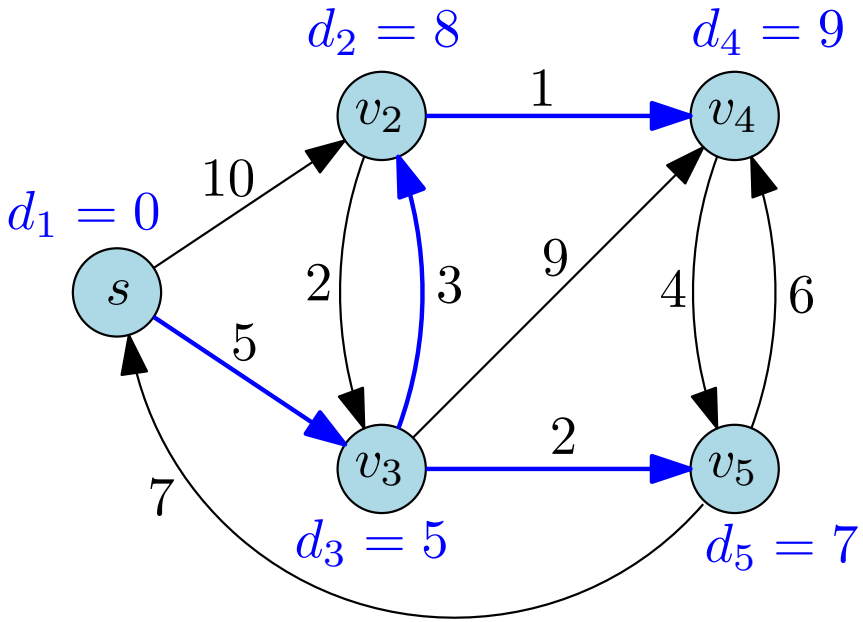

$d'_2 = 8$
$d'_4 = 13$
v_2
v_4
1
10
$d_1 = 0$
s
2
3
9
4
6
5
v_3
v_5
2
7
$d_3 = 5$
$d_5 = 7$

Example


$d_2 = 8$
$d'_4 = 13$
v_2
v_4
1
10
$d_1 = 0$
s
2
3
9
4
6
5
v_3
v_5
2
7
$d_3 = 5$
$d_5 = 7$


$d_2 = 8$
$d'_4 = 9$
v_2
v_4
1
10
$d_1 = 0$
s
2
3
9
4
6
5
v_3
v_5
2
7
$d_3 = 5$
$d_5 = 7$

Example



- The (unique) path from s to v_i in the blue tree is a shortest path.

Dijkstra's Algorithm

Pseudocode

1: **procedure** DIJKSTRA($G(V, E), \ell$)
2: $S \leftarrow \{v_1 = s\}$ ▷ set of explored nodes
3: $d_1 \leftarrow 0$
4: $d'_j \leftarrow \infty$ for all $j \geqslant 2$
5: **for** each edge (s, v_j) **do**
6: $d'_j \leftarrow \ell(s, v_j)$
7: **while** $S \neq V$ **do**
8: Let v_i be the node in $V \setminus S$ such that d'_i is minimum
9: $d_i \leftarrow d'_i$
10: $S \leftarrow S \cup \{v_i\}$
11: **for** each edge (v_i, v_j) such that $v_j \notin S$ **do**
12: $d'_j \leftarrow \min(d'_j, d_i + \ell(v_i, v_j))$

Proof of Correctness

- We prove that the algorithm is correct using the loop invariants below. Correctness of the algorithm then follows immediately from Invariant (a).

Lemma

At the beginning of each iteration of the **while** *loop, the following hold:*

(a) *For every $v_p \in S$, the value d_p is the length $d(s, v_p)$ of a shortest path from s to v_p.*

(b) *If there is an edge from a node of S to a node $v_q \notin S$, then d'_q is the length of a shortest path from s to v_q, under the constraint that the last edge starts from S.*

Proof of the Lemma

- Lines 2–6 ensure that the invariants are true at the beginning of the first iteration.
- Now suppose it is true at the beginning of the current iteration.
- Let v_i be the node chosen at Line 8.
- We first argue that $d'_i = d(s, v_i)$.
  - Consider a shortest path P from s to v_i, hence its length is $d(s, v_i)$.
  - Let v_b be the first node in P that is not in S, and let v_a be the previous vertex along P.
  - Then $v_a \in S$, $v_b \notin S$, and $(v_a, v_b) \in E$.
  - The length of P is at least $d_a + \ell(v_a, v_b)$.
  - Therefore, $d(s, v_i) \geqslant d_a + \ell(v_a, v_b)$.
  - By (b), we know that $d_a + \ell(v_a, v_b) \geqslant d'_b$.
  - Since $d'_b \geqslant d'_i$, it follows that $d(s, v_i) \geqslant d'_i$.
  - As d'_i is the length of a path from s to v_i, we also have $d'_i \geqslant d(s, v_i)$, and thus $d'_i = d(s, v_i)$.
- As $d'_i = d(s, v_i)$, Invariant (a) will be true at the end of this iteration.
- Then Lines 11–12 ensure that Invariant (b) is maintained, by handling the outgoing edges from the vertex v_i that was inserted into S.

Efficient Implementation and Analysis

- The while loop is iterated n times, when $n = |V|$.
- So Line 8 is executed n times.
- We store the values d'_j in a heap-based priority queue $\mathcal{Q}$, which allows to execute Line 8 in $O(\log n)$ time.
- Similarly, Line 12 takes $O(\log n)$ time using a ChangeKey operation.
- Line 12 is executed at most once for each edge (v_i, v_j), as v_i is inserted into S at most once, so in total Line 12 is executed $m = |E|$ times.

Theorem

Dijkstra's algorithm can be implemented to run in time $O((n + m) \log n)$ time, where n is the number of vertices and m is the number of edges.

Concluding Remarks

- Dijkstra's algorithm can be seen as a dynamic programming approach, as we compute each d_i using precomputed values $d_j \leqslant d_i$.
- It can also be seen as a greedy algorithm as at each step, we pick the closest unexplored vertex.
- We only saw how to compute d_i. A shortest path from s to v_i can be found by simple modifications of the algorithm, as usual with dynamic programming. Same as BFS, we can construct a spanning tree (blue in the figures) such that the unique path $s \rightsquigarrow v_i$ in this tree is the shortest path within the input graph.