Dijkstra's Algorithm

Single Source Shortest Path

Introduction

- Consider the problem of finding shortest paths between all pairs of vertices in a graph
- Problem might arise in making a table of distances between all pairs of cities for a road atlas
- given a weighted, directed graph $G = (V, E)$ with a weight function $w: E \rightarrow R$ that maps edges to real-valued weights

Introduction

- We wish to find, for every pair of vertices u, v ∈ V , a shortest (least–weight) path from u to v, where the weight of a path is the sum of the weights of its constituent edges
- We typically want the output in tabular form:
- the entry in u's row and v's column should be the weight of a shortest path from u to v

Introduction

- We can solve an all–pairs shortest–paths problem by running a single–source shortest–paths algorithm |V| times, once for each vertex as the source.
- If all edge weights are nonnegative, we can use Dijkstra's algorithm

Introduction

- If we use the linear–array implementation of the min–priority queue, the running time is $O(V^3 + VE) = O(V^3)$.
- The binary min–heap implementation of the min–priority queue yields a running time of $O(VE \lg V)$, which is an improvement if the graph is sparse.
- Alternatively, we can implement the min–priority queue with a Fibonacci heap, yielding a running time of $O(V^2 \lg V + VE)$

Introduction

- If the graph has negative–weight edges, we cannot use Dijkstra's algorithm
- Instead, we must run the slower Bellman–Ford algorithm once from each vertex
- The resulting running time is $O(V^2E)$, which on a dense graph is $O(V^4)$

Introduction

- In this chapter we shall see how to do better.
- We also investigate the relation of the all–pairs shortest–paths problem to matrix multiplication and study its algebraic structure.
- Unlike the single–source algorithms, which assume an adjacency–list represen–tation of the graph, most of the algorithms in this chapter use an adjacency–matrix representation.

Introduction

- (Johnson's algorithm for sparse graphs, in Section 25.3, uses adjacency lists.)
- For convenience, we assume that the vertices are numbered 1, 2,…,|V|, so that the input is an n x n matrix W representing the edge weights of an n–vertex directed graph G =(V, E).
- That is, $W = (w_{ij})$, where

Introduction

- We allow negative–weight edges, but we assume for the time being that the input graph contains no negative–weight cycles.

Dijkstra's algorithm

```
DIJKSTRA(G, w, s)
  INITIALIZE-SINGLE-SOURCE(G, s)
  S = Ø
  Q = G.V
  while Q ≠ Ø
      u = EXTRACT-MIN(Q)
      S = S ∪ {u}
      for each vertex v ∈ G.Adj[u]
          RELAX(u, v, w)
```

Dijkstra's algorithm

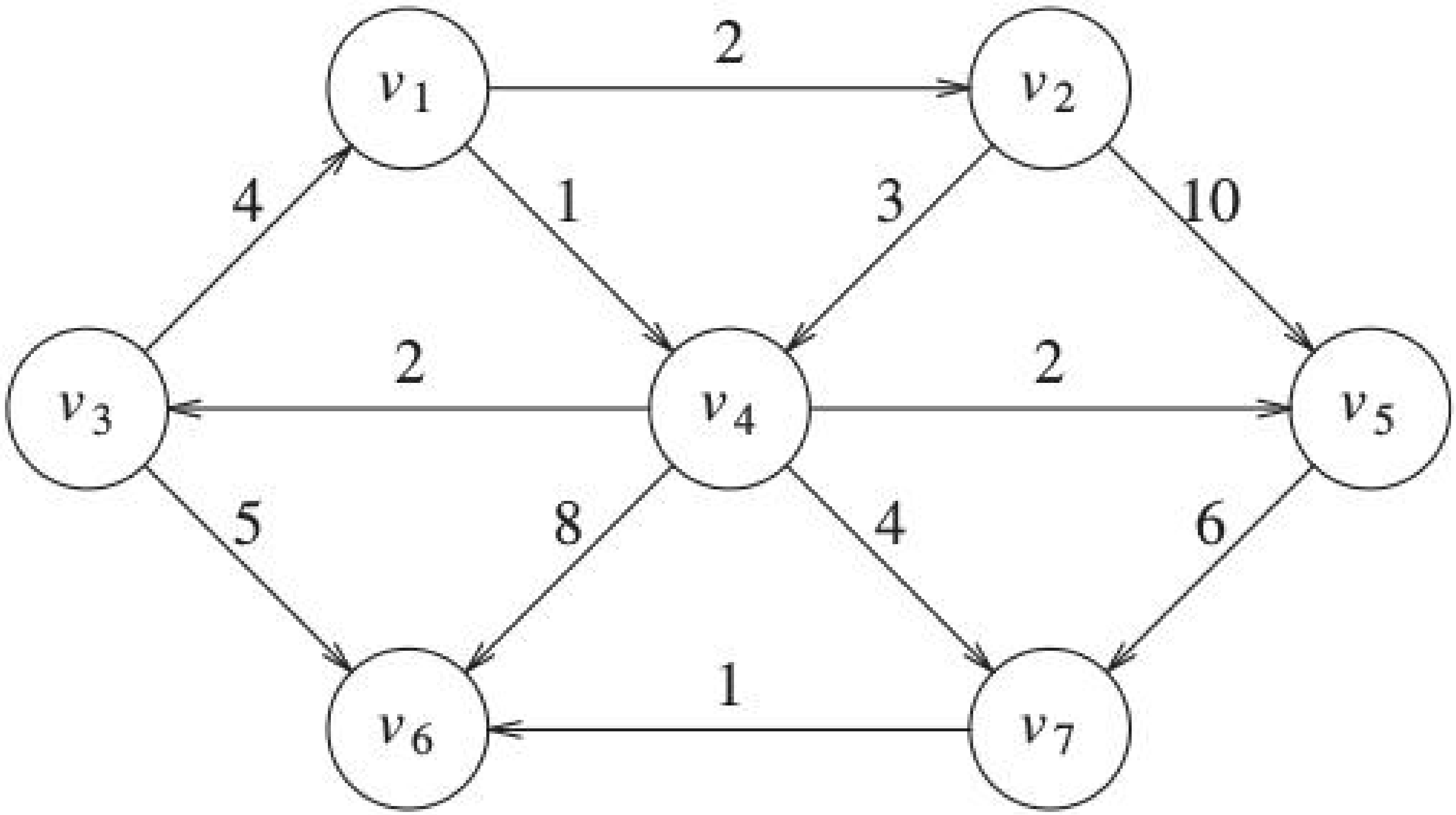


Figure 9.20 The directed graph G (again)

Dijkstra's algorithm

v	*known*	d_v	p_v
v_1	F	0	0
v_2	F	∞	0
v_3	F	∞	0
v_4	F	∞	0
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

Figure 9.21 Initial configuration of table used in Dijkstra's algorithm

Dijkstra's algorithm

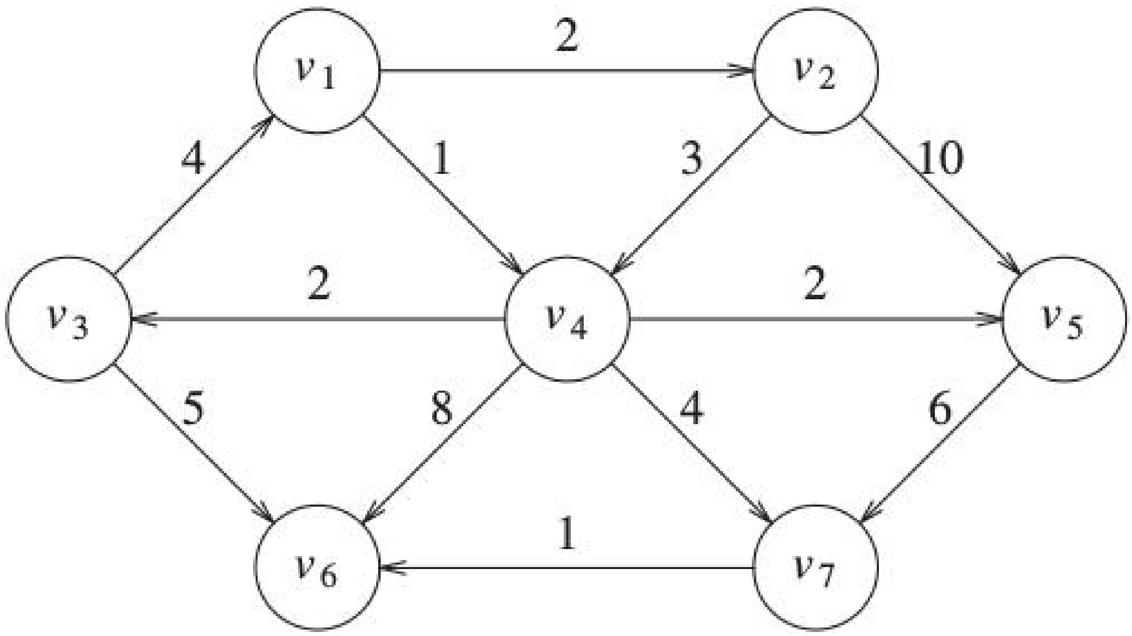


Figure 9.20 The directed graph G (again)

v	*known*	d_v	p_v
v_1	T	0	0
v_2	F	2	v_1
v_3	F	∞	0
v_4	F	1	v_1
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0

Figure 9.22 After v_1 is declared *known*

Dijkstra's algorithm

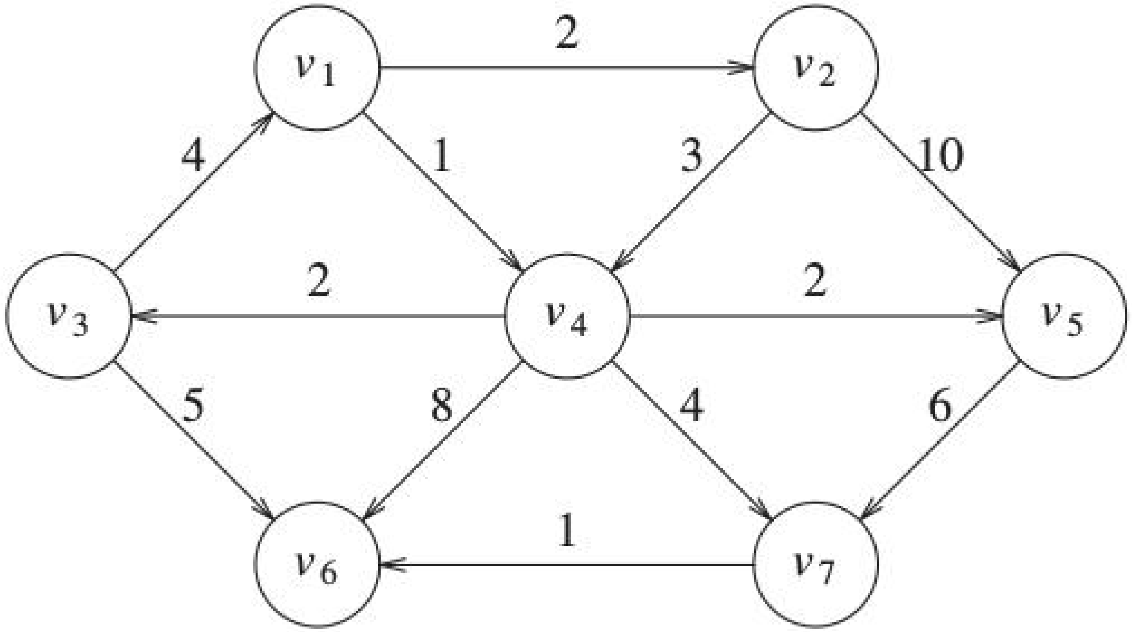


Figure 9.20 The directed graph G (again)

v	*known*	d_v	p_v
v_1	T	0	0
v_2	F	2	v_1
v_3	F	3	v_4
v_4	T	1	v_1
v_5	F	3	v_4
v_6	F	9	v_4
v_7	F	5	v_4

Figure 9.23 After v_4 is declared *known*

Dijkstra's algorithm

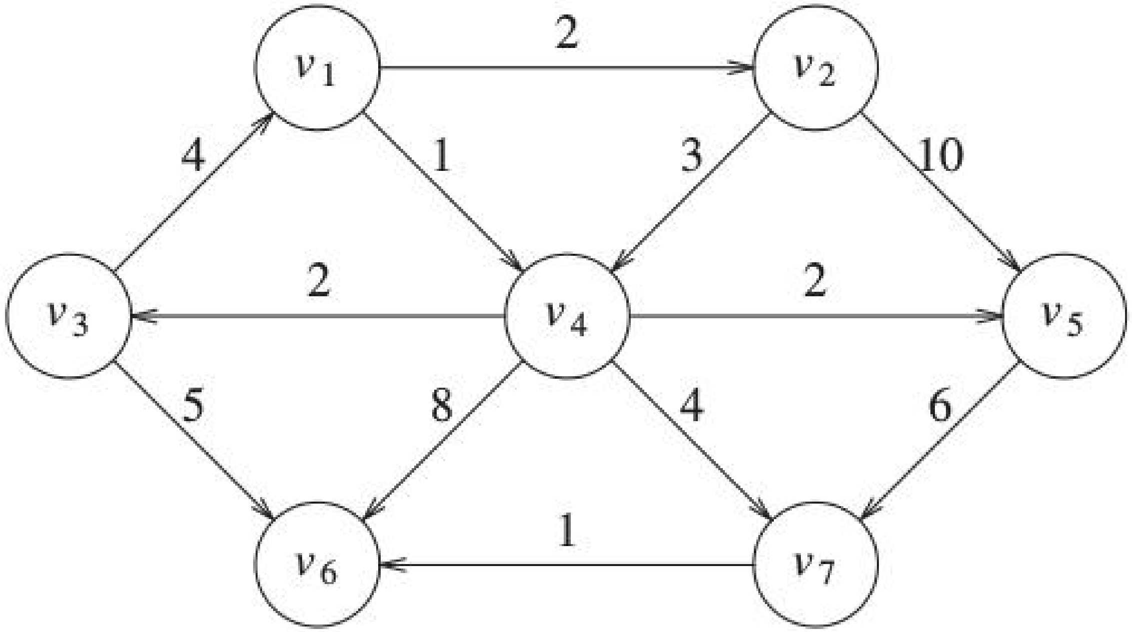


Figure 9.20 The directed graph G (again)

v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	F	3	v_4
v_4	T	1	v_1
v_5	F	3	v_4
v_6	F	9	v_4
v_7	F	5	v_4

Figure 9.24 After v_2 is declared *known*

Dijkstra's algorithm

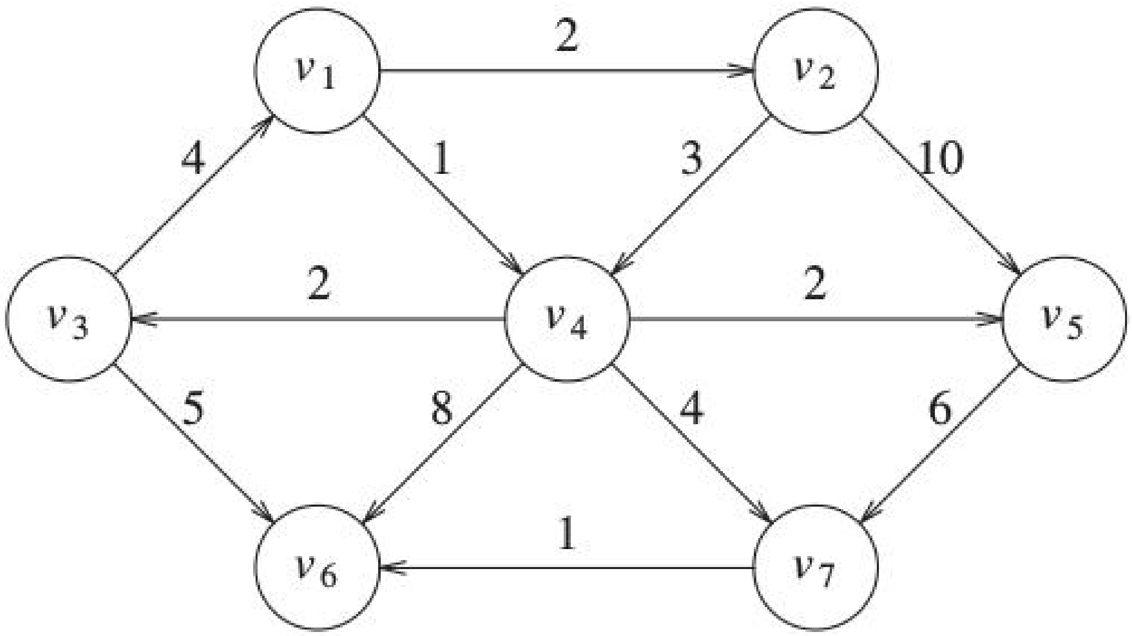


Figure 9.20 The directed graph G (again)

v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	3	v_4
v_4	T	1	v_1
v_5	T	3	v_4
v_6	F	8	v_3
v_7	F	5	v_4

Figure 9.25 After v_5 and then v_3 are declared *known*

Dijkstra's algorithm

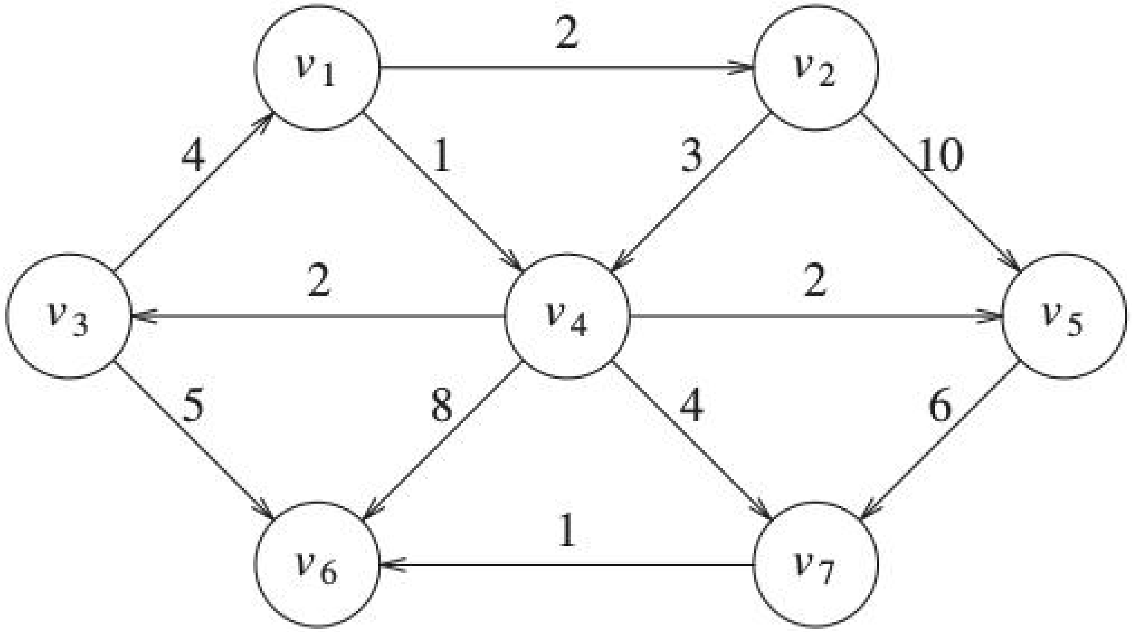


Figure 9.20 The directed graph G (again)

v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	3	v_4
v_4	T	1	v_1
v_5	T	3	v_4
v_6	F	6	v_7
v_7	T	5	v_4

Figure 9.26 After v_7 is declared *known*

Dijkstra's algorithm

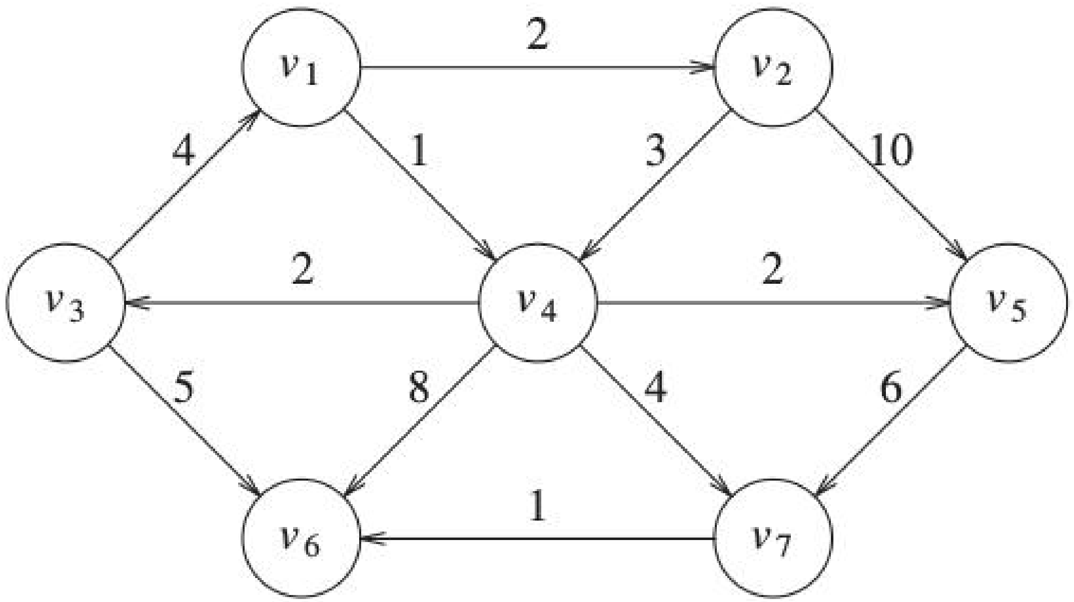
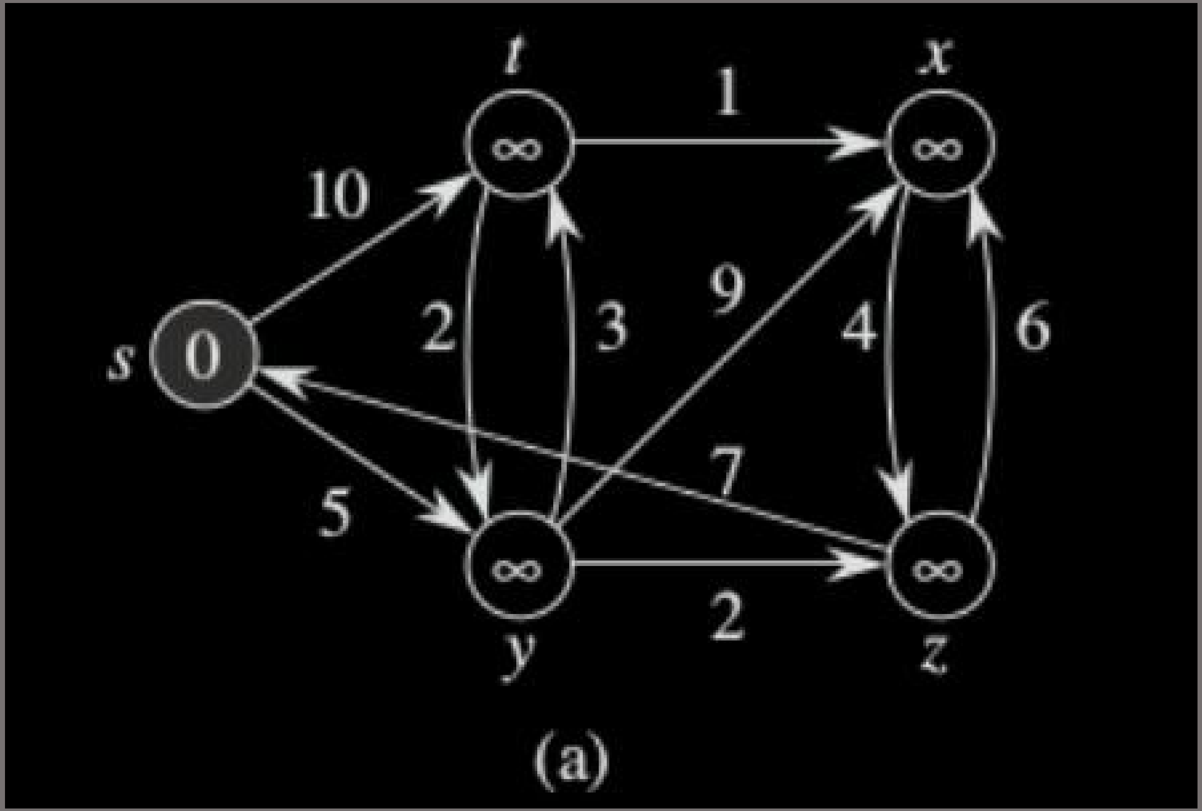


Figure 9.20 The directed graph G (again)

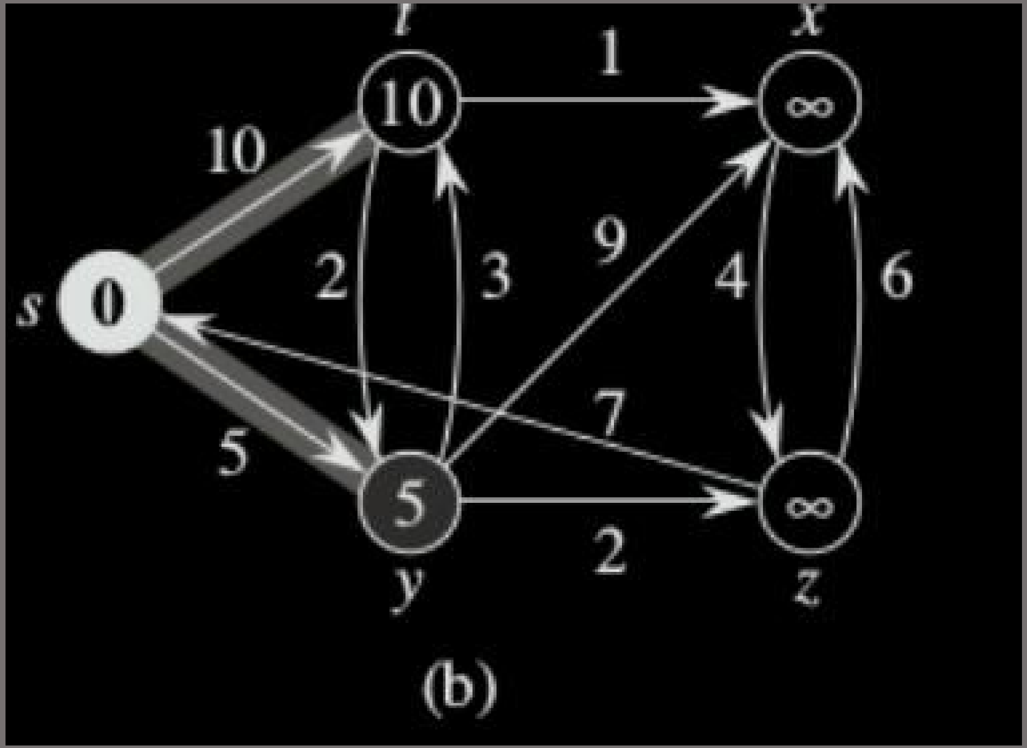
v	*known*	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	3	v_4
v_4	T	1	v_1
v_5	T	3	v_4
v_6	T	6	v_7
v_7	T	5	v_4

Figure 9.27 After v_6 is declared *known* and algorithm terminates

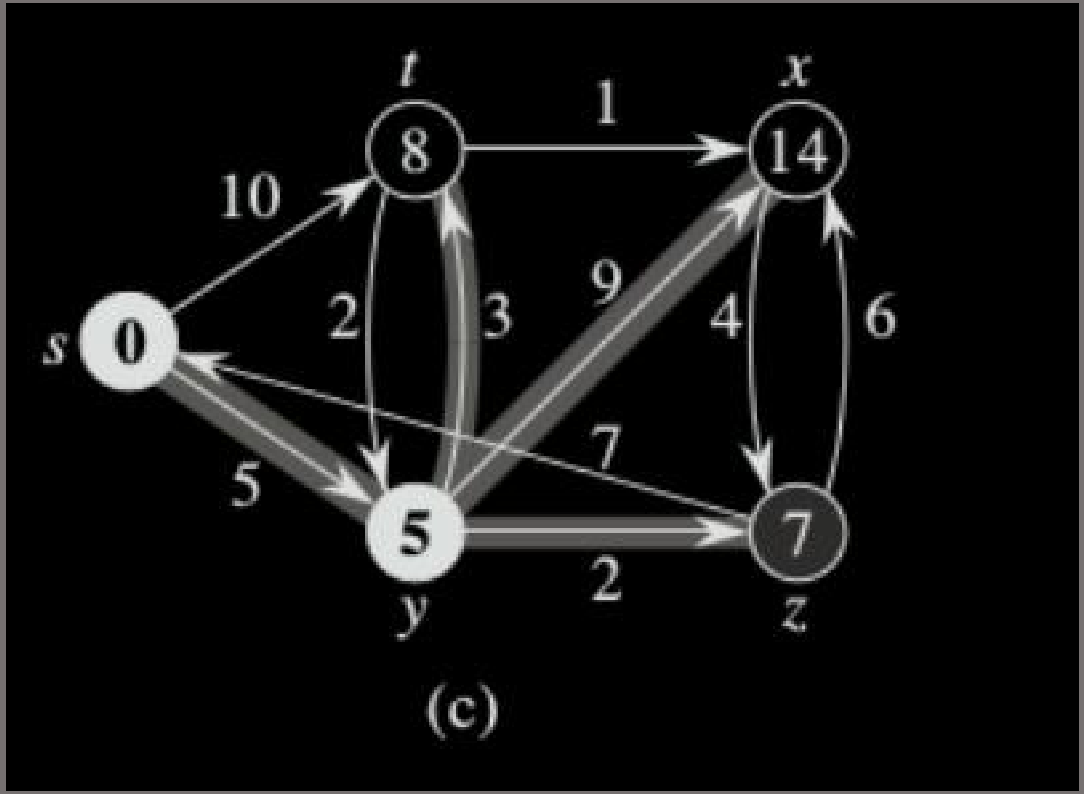
Dijkstra’s algorithm



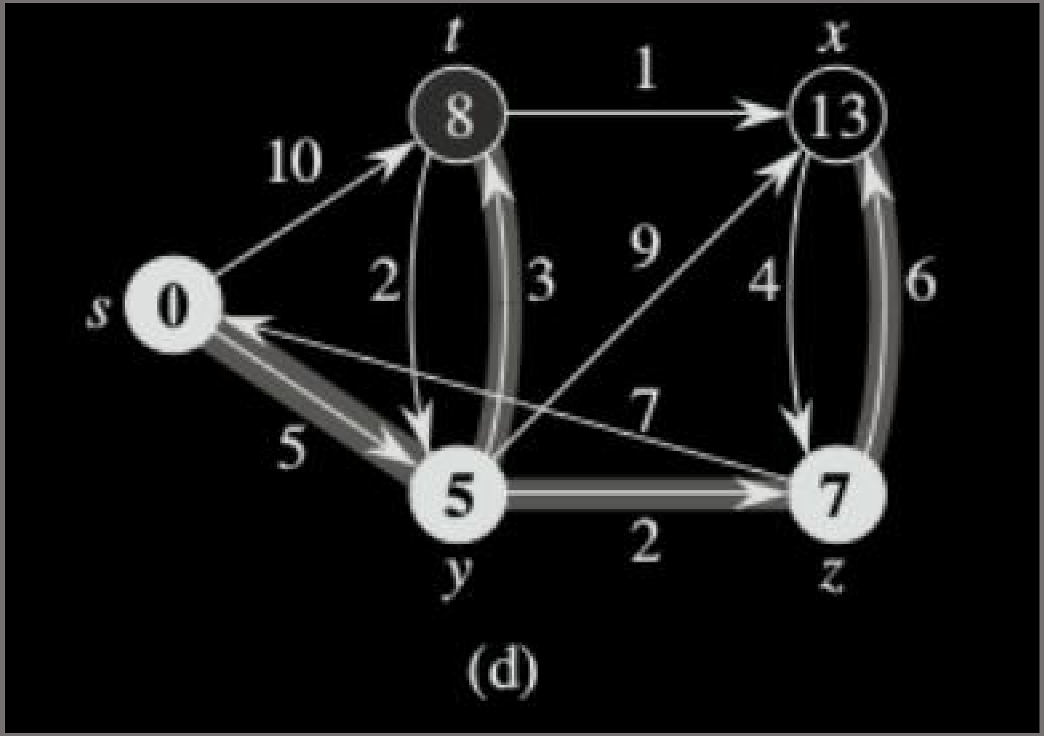
Dijkstra's algorithm



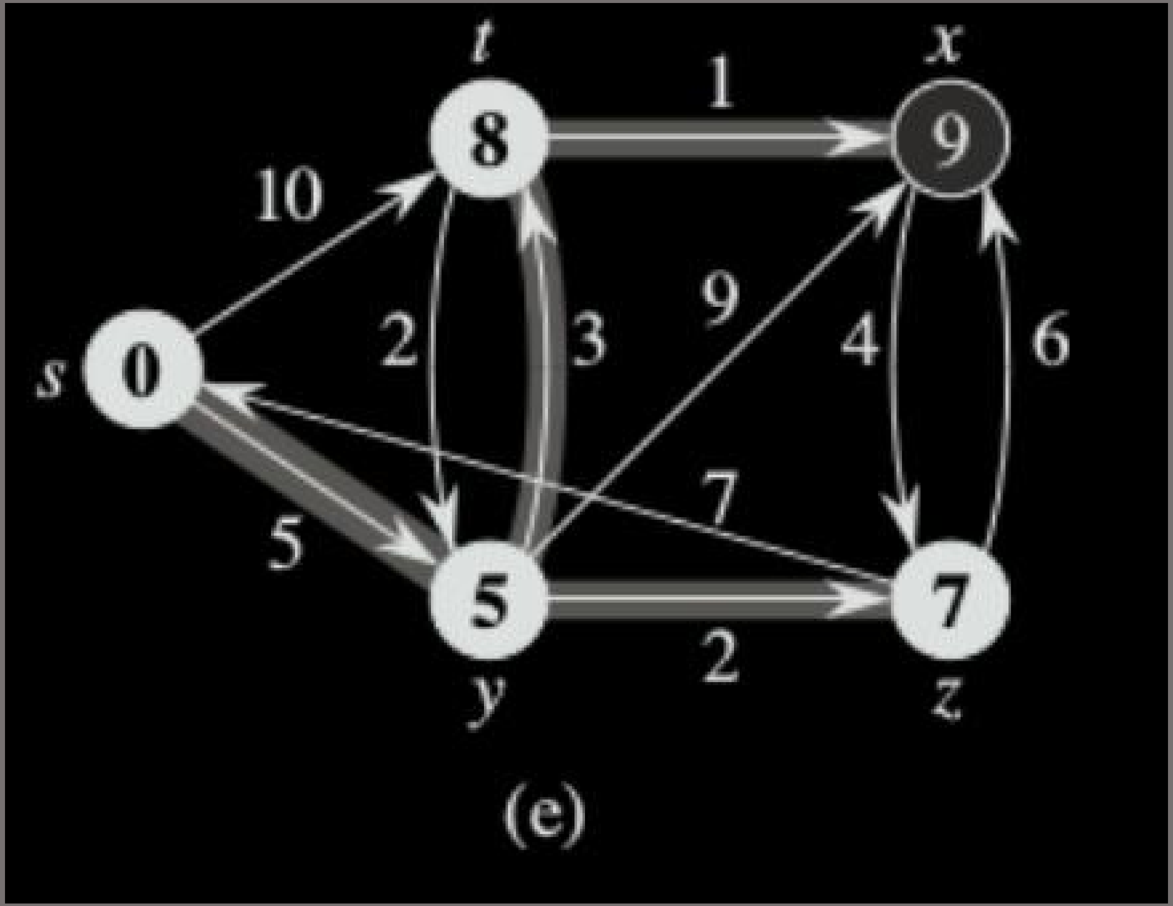
Dijkstra’s algorithm



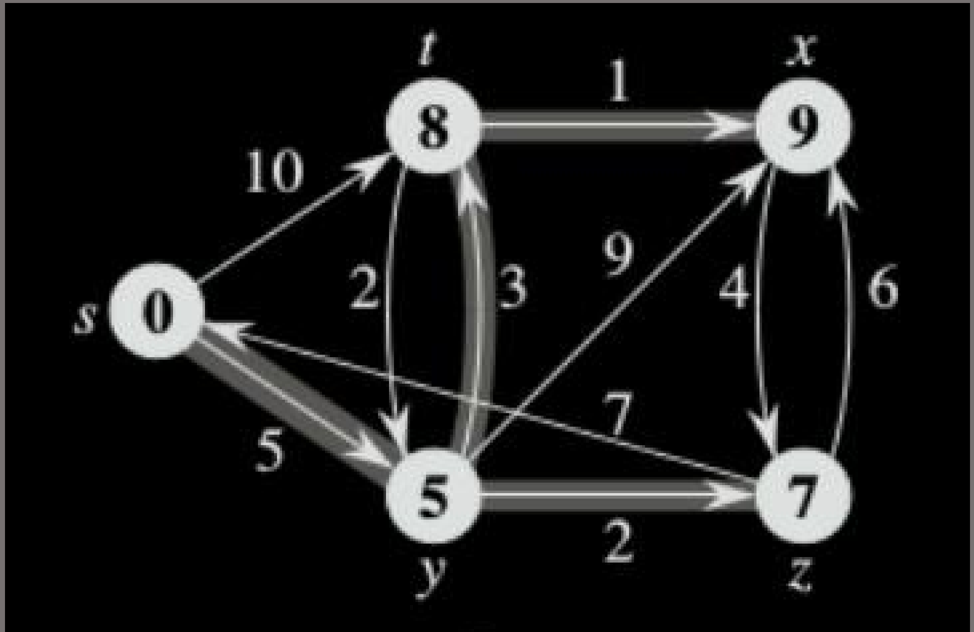
Dijkstra's algorithm



Dijkstra's algorithm



Dijkstra’s algorithm



(f)

Dijkstra’s algorithm

- We allow negative–weight edges, but we assume for the time being that the input graph contains no negative–weight cycles.
- Time Complexity of Dijkstra's Algorithm is $O(V^2)$ but with min–priority queue it drops down to $O(V + E\log V)$