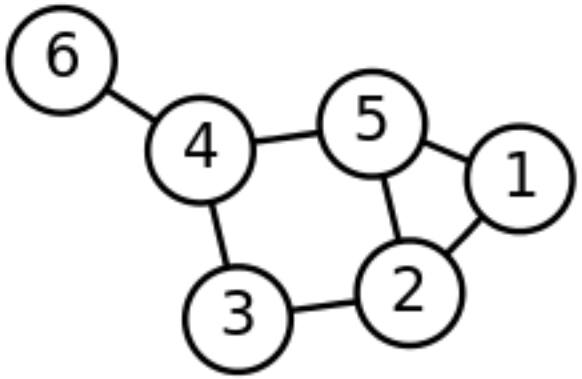
Northern Illinois University

Graphs

Dr. Maoyuan Sun – smaoyuan@niu.edu

Definitions (1)

- **graph** – a *graph* $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ consists of a set of *vertices* $\boldsymbol{V}$ and a set of *edges* $\boldsymbol{E}$
- **vertices** – a vertex (plural vertices) or node is the fundamental unit of which graphs are formed*
- **edges** (arcs) – each edge is a pair (v, w) where $v, w \in V$

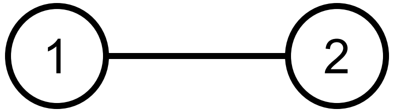


Graph G with 6 vertices (1, 2, 3, 4, 5, 6) and 7 edges ((1, 2), (1,5), (2, 3), (2,5), (3, 4), (4, 5), (4, 6))

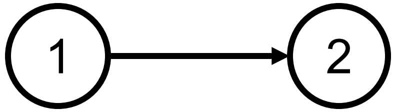
*en.wikipedia.org/wiki/Vertex_(graph_theory)

Definitions (2)

- **directed** (digraphs) – if the pair is ordered then the graph is *directed*



Vertices: 1, 2 Edge: $(1, 2) \equiv (2, 1)$



Vertices: 1, 2 Edge: $(1, 2) \neq (2, 1)$

- **adjacent** – vertices v and w are *adjacent* if they are they are endpoints of the same edge, that is vertex w is adjacent to v if and only if $(v, w) \in E$

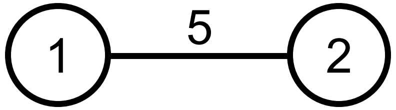
Vertices: 1, 2 are adjacent

Vertices: 1, 2 are **not** adjacent

Definitions (3)

- **weight** (cost) – optional third component to an edge, numerical value assigned as a label to the edge

Vertices: 1, 2 Edge: (1, 2) no weight



Vertices: 1, 2 Edge: (1, 2) weight of 5

- **path** – in a graph is a sequence of vertices $w_1, w_2, w_3, \dots, w_N$ such that $(w_i, w_{i+1}) \in E$ for $1 \le i \le N$

Vertices: w_1, w_2, w_3, w_4 Edges: (w_1, w_2), (w_2, w_3) with a path from w_1 to w_3, where $N = 3$

Definitions (4)

- **length** – is the number of edges on a path, it is equal to $N - 1$

Vertices: w_1, w_2, w_3, w_4 Edges: (w_1, w_2), (w_2, w_3)
with a path from w_1 to w_3, where $N = 3$ and the length is 2
If a path contains no edges, then its path length is 0

- **loop** – if there is an edge (v, v) from a vertex to itself then this path is known as a *loop*, we will consider graphs in general will be loopless
- **simple path** – is a *path* that all vertices are distinct, except the first and last could be the same

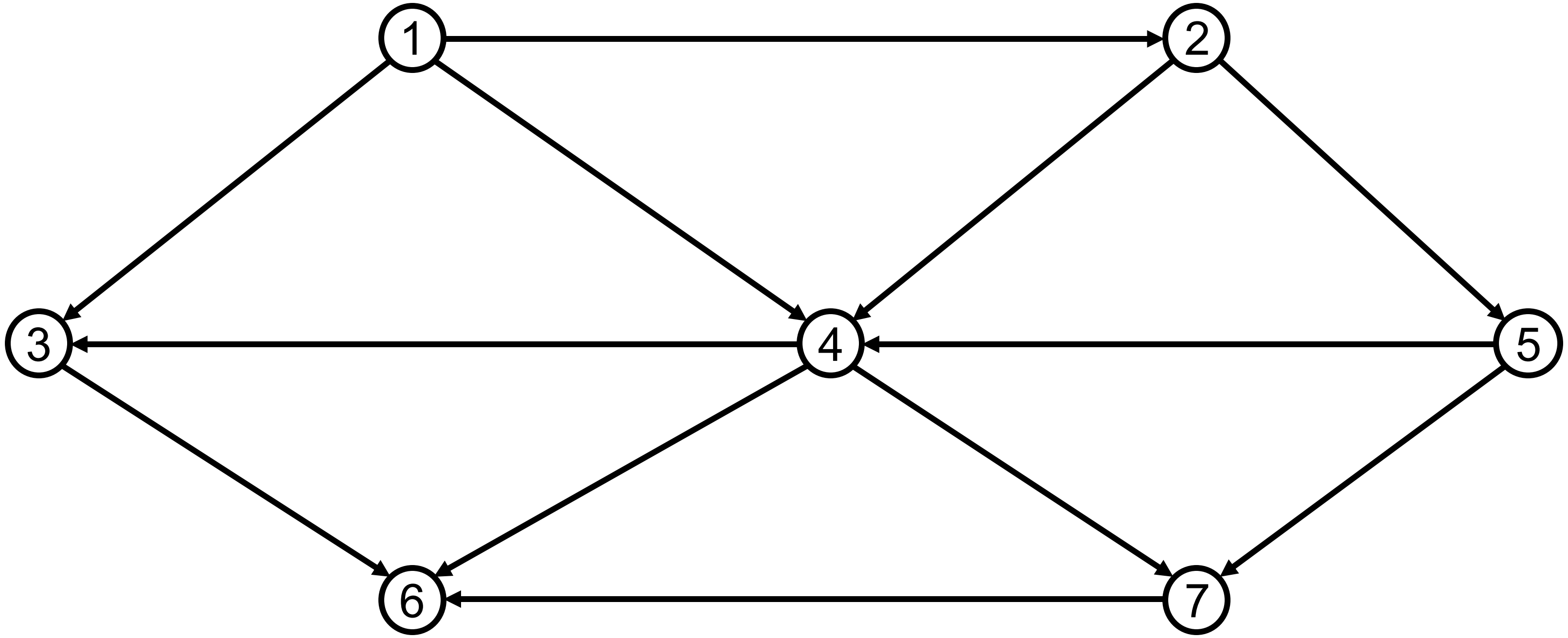
Definitions (5)

- **cycle** – in a directed graph is a path with a length of at least 1, such that $w_1 = w_N$, the *cycle* is simple if the path is simple; in an undirected graph the edges must be distinct
- **acyclic** (DAG) – is a directed graph with no cycles
- **connected** – in an undirected graph, the graph is connected if there is a path from every vertex to every other vertex
- **strongly connected** – a **connected** directed graph is known as a *strongly connected* graph
- **weakly connected** – a graph is *weakly connected* if the directed graph is **connected** when direction of the edges is ignored
- **complete** – is a graph where there is an edge between every pair of vertices
- **Indegree** – the number of incoming edges in directed graph
- **Outdegree** – the number of outgoing edges in directed graph

Topological Sort

- Formally, we use our definition of **indegree** of a vertex v as the number of edges (u, v). Compute the indegree of all vertices in the graph and keep in adjacency list to generate a topological order

1	0
2	1
3	2
4	3
5	1
6	3
7	2



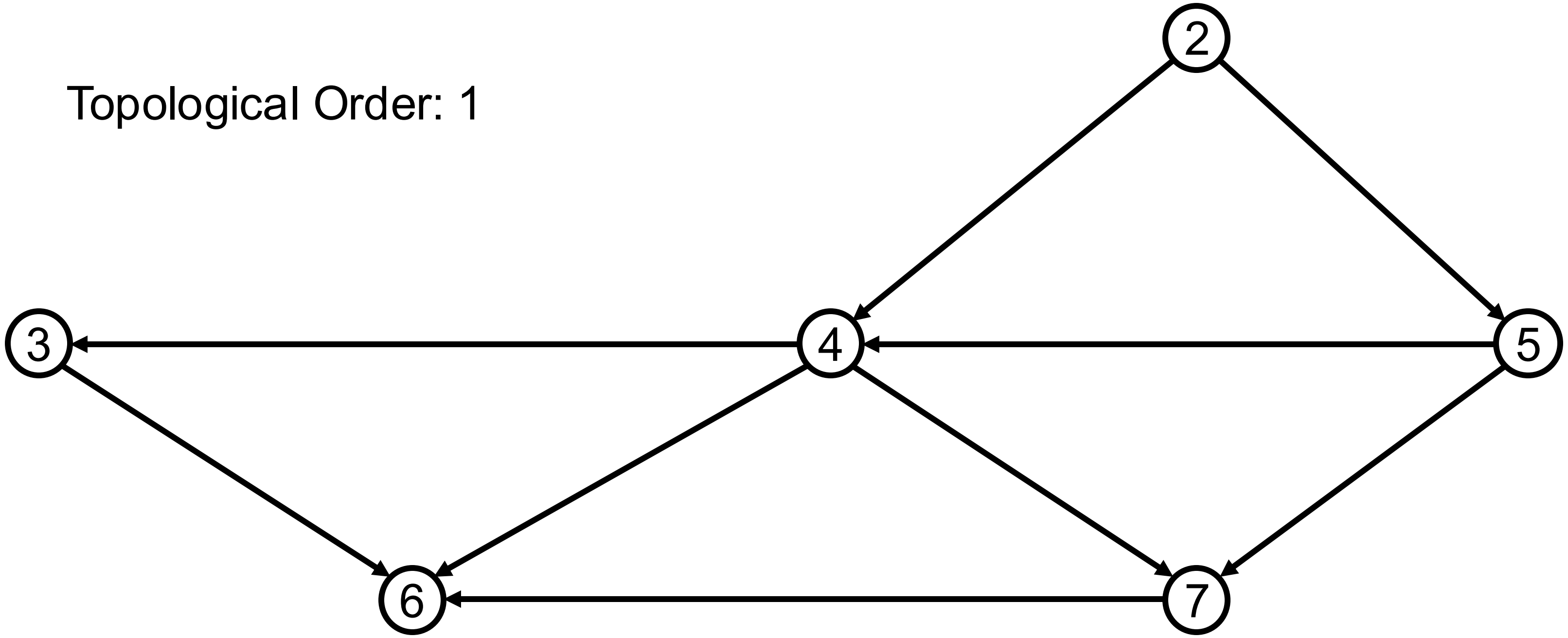
Pick one with Zero (0)

Topological Sort

- Formally, we use our definition of **indegree** of a vertex v as the number of edges (u, v). Compute the indegree of all vertices in the graph and keep in adjacency list to generate a topological order

1	0	X
2	1	0
3	2	1
4	3	2
5	1	1
6	3	3
7	2	2

Topological Order: 1



Pick one with Zero (0)

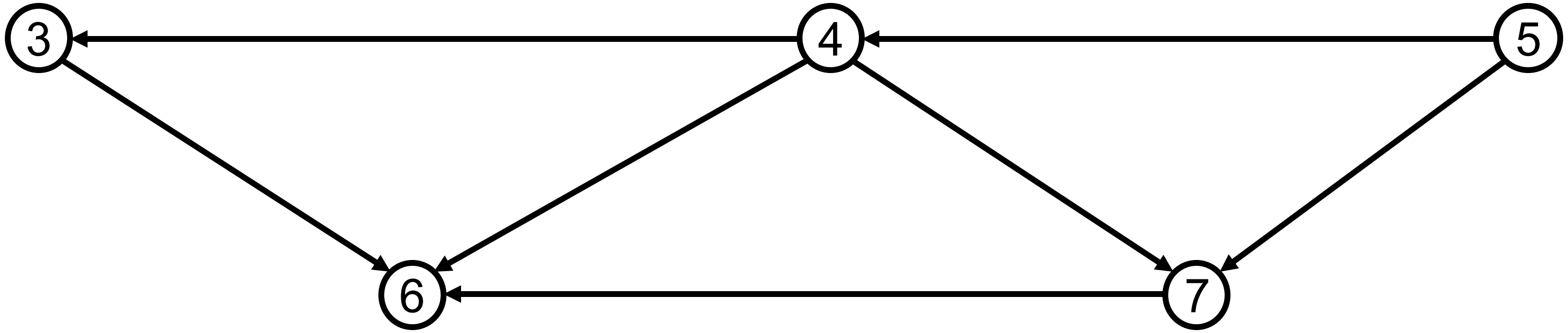
Topological Sort

- Formally, we use our definition of **indegree** of a vertex v as the number of edges (u, v). Compute the indegree of all vertices in the graph and keep in adjacency list to generate a topological order

1	0	X	
2	1	0	X
3	2	1	1
4	3	2	1
5	1	1	0
6	3	3	3
7	2	2	2

Pick one with Zero (0)

Topological Order: 1, 2



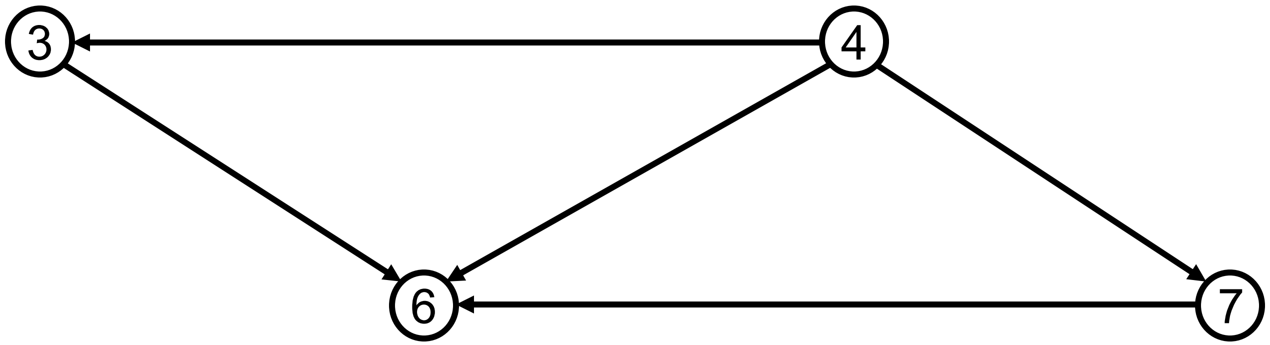
Topological Sort

- Formally, we use our definition of **indegree** of a vertex v as the number of edges (u, v). Compute the indegree of all vertices in the graph and keep in adjacency list to generate a topological order

1	0	X		
2	1	0	X	
3	2	1	1	1
4	3	2	1	0
5	1	1	0	X
6	3	3	3	3
7	2	2	2	1

Pick one with Zero (0)

Topological Order: 1, 2, 5

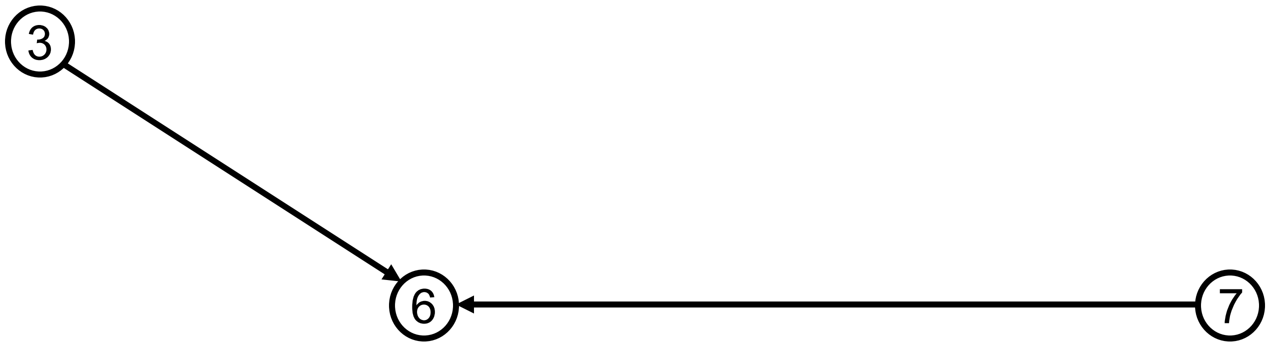


Topological Sort

- Formally, we use our definition of **indegree** of a vertex v as the number of edges (u, v). Compute the indegree of all vertices in the graph and keep in adjacency list to generate a topological order

1	0	X			
2	1	0	X		
3	2	1	1	1	0
4	3	2	1	0	X
5	1	1	0	X	
6	3	3	3	3	2
7	2	2	2	1	0

Topological Order: 1, 2, 5, 4



Pick one with Zero (0)

Topological Sort

- Formally, we use our definition of **indegree** of a vertex v as the number of edges (u, v). Compute the indegree of all vertices in the graph and keep in adjacency list to generate a topological order

1	0	X				
2	1	0	X			
3	2	1	1	1	0	X
4	3	2	1	0	X	
5	1	1	0	X		
6	3	3	3	3	2	0
7	2	2	2	1	0	X

Topological Order: 1, 2, 5, 4, {3, 7}

6

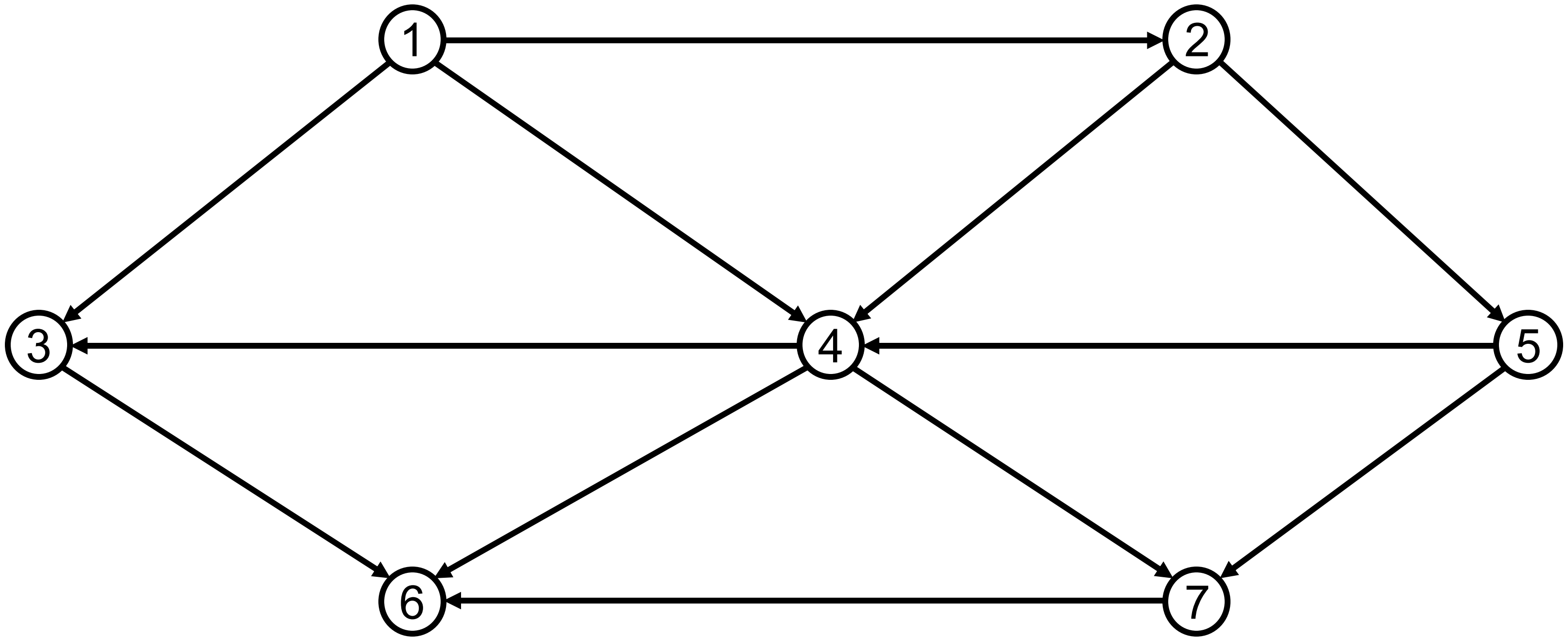
Pick one with Zero (0)

Topological Sort

- Formally, we use our definition of **indegree** of a vertex v as the number of edges (u, v). Compute the indegree of all vertices in the graph and keep in adjacency list to generate a topological order

1	0	X					
2	1	0	X				
3	2	1	1	1	0	X	
4	3	2	1	0	X		
5	1	1	0	X			
6	3	3	3	3	2	0	X
7	2	2	2	1	0	X	

Pick one with Zero (0)



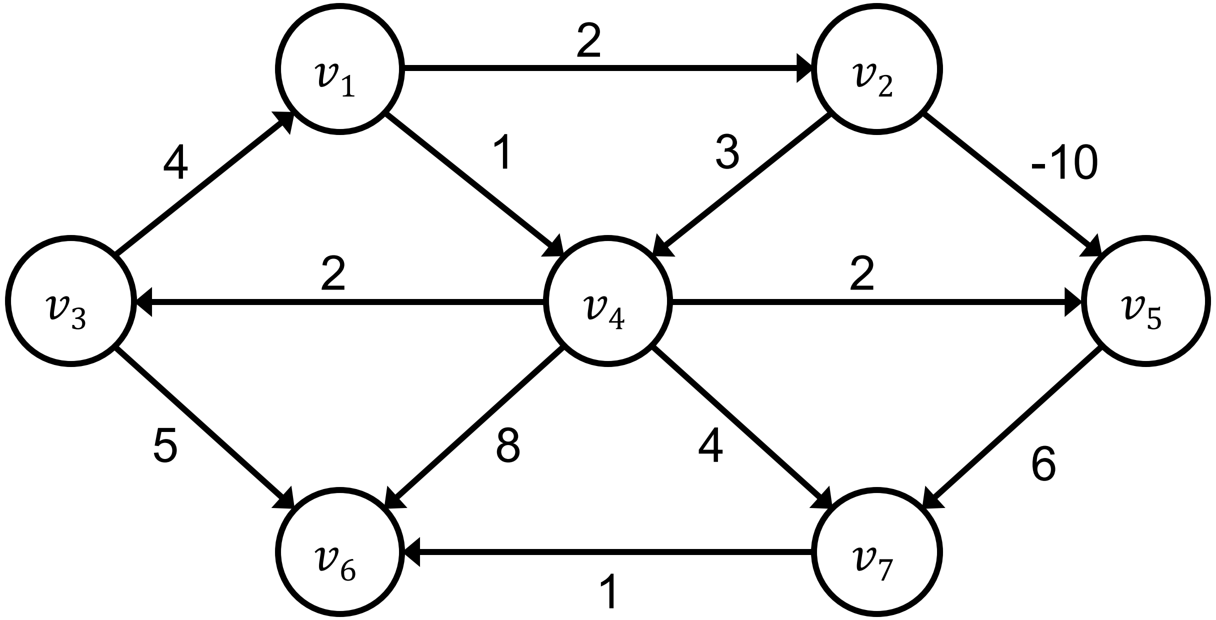
Topological Order: 1, 2, 5, 4, {3, 7}, 6

Shortest Path Algorithms

- The input is a weighted graph
- Associated with each edge (v_i, v_j) is a cost $c_{i,j}$ to traverse the edge
- The cost of a path $v_1, v_2, v_3, \ldots v_N$ is $\sum_{i=1}^{N-1} c_{i,i+1}$, this is called the **weighted path length**, the **unweighted path length** is the number of edges on the path, $N-1$

Single-Source Shortest Path Problem

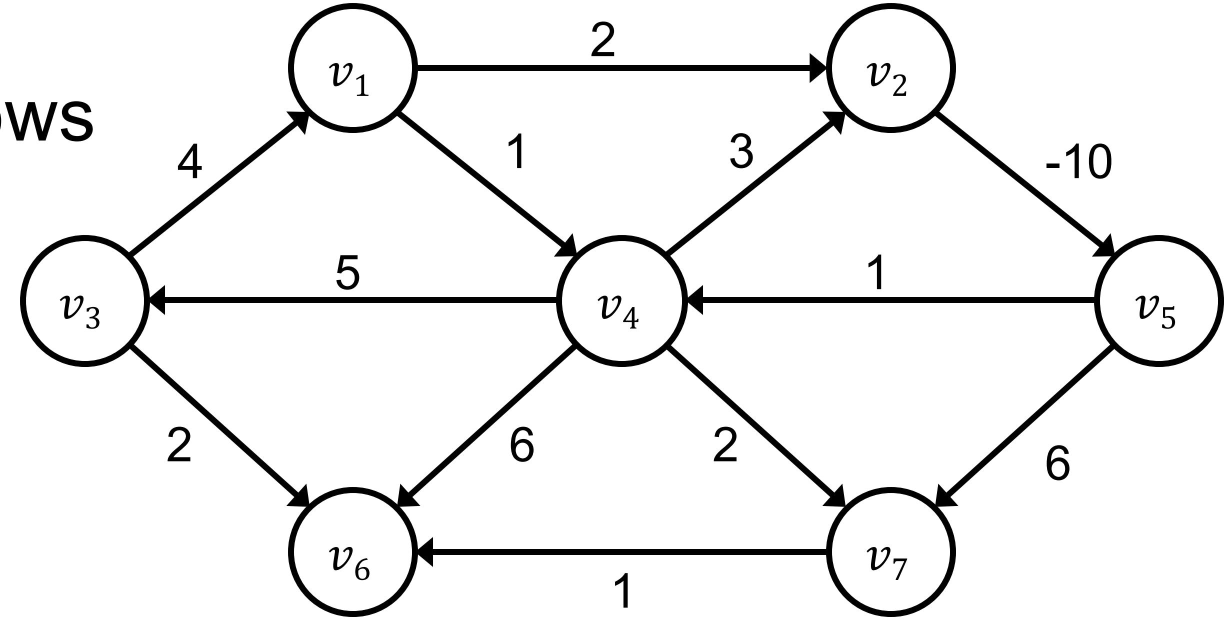
- Given a weighted graph, $G = (V, E)$ and a distinguished vertex, s, find the shortest weighted path from s to every other vertex in G



- What is shortest weighted path from v_1 to v_6? v_1 to v_4 to v_7 to v_6 with a cost of 6

Single-Source Shortest Path Problem

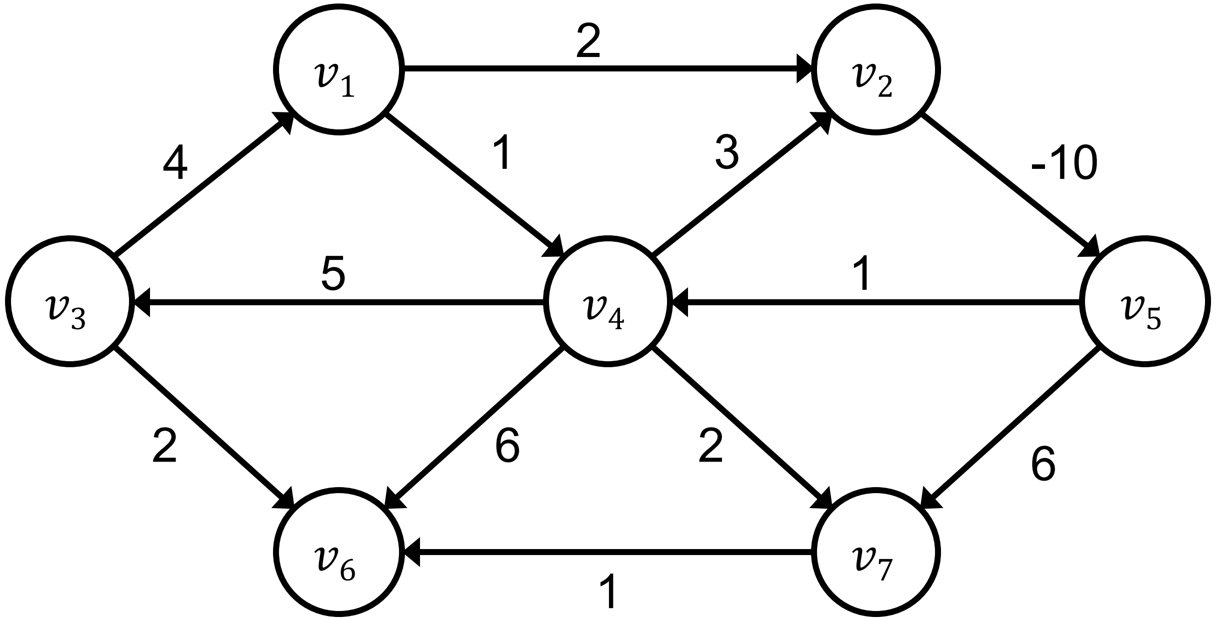
- Updating the weights and arrows



- The path v_5 to v_4 cost 1, but a cheaper path exists by following the loop v_5 to v_4 to v_2 to v_5 to v_4 which costs -5, it could get even cheaper by staying in the loop forever.

Single-Source Shortest Path Problem

- Where are other loops?



- The path v_3 to v_4 where we can go v_3 to v_1 to v_4 with cost of 5 or v_3 to v_1 to v_2 to v_5 to v_4 for cost of -3. Similar for v_1 to v_6
- These are **negative-cost cycles,** and the shortest path is **undefined**

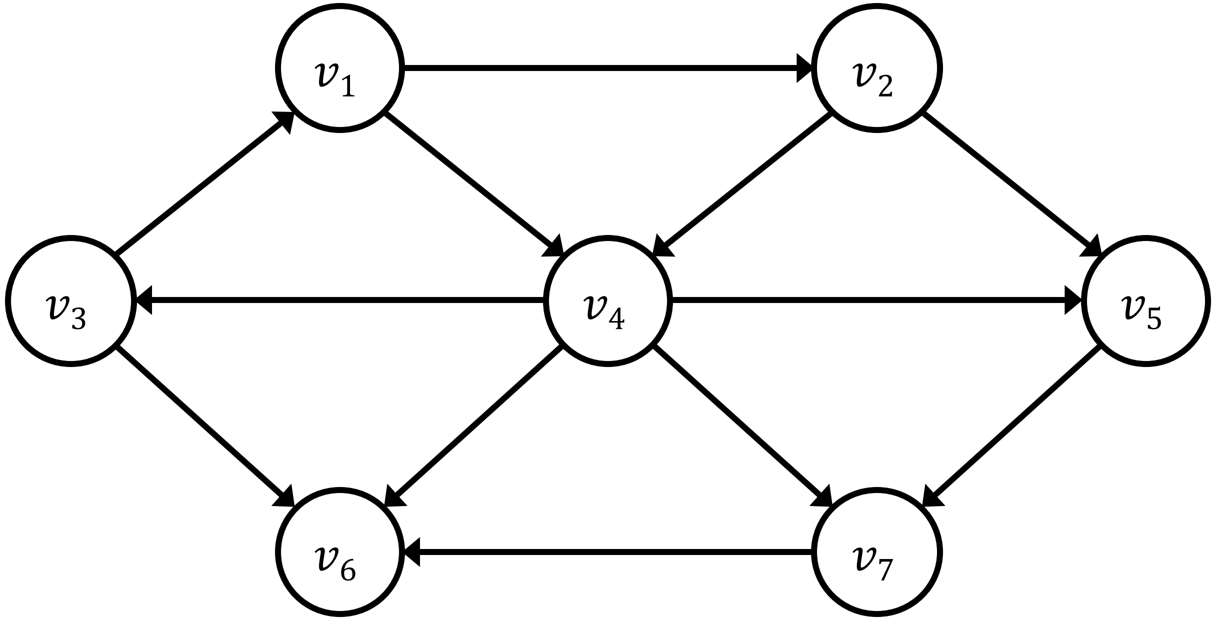
***NOTE:** Negative edges are not bad, they just make shortest path problem harder!

Shortest Path Problem

- Examples:
  - Airline travel from city to city
  - Transfer file from one computer to other
  - ……

Unweighted Shortest Path Problem

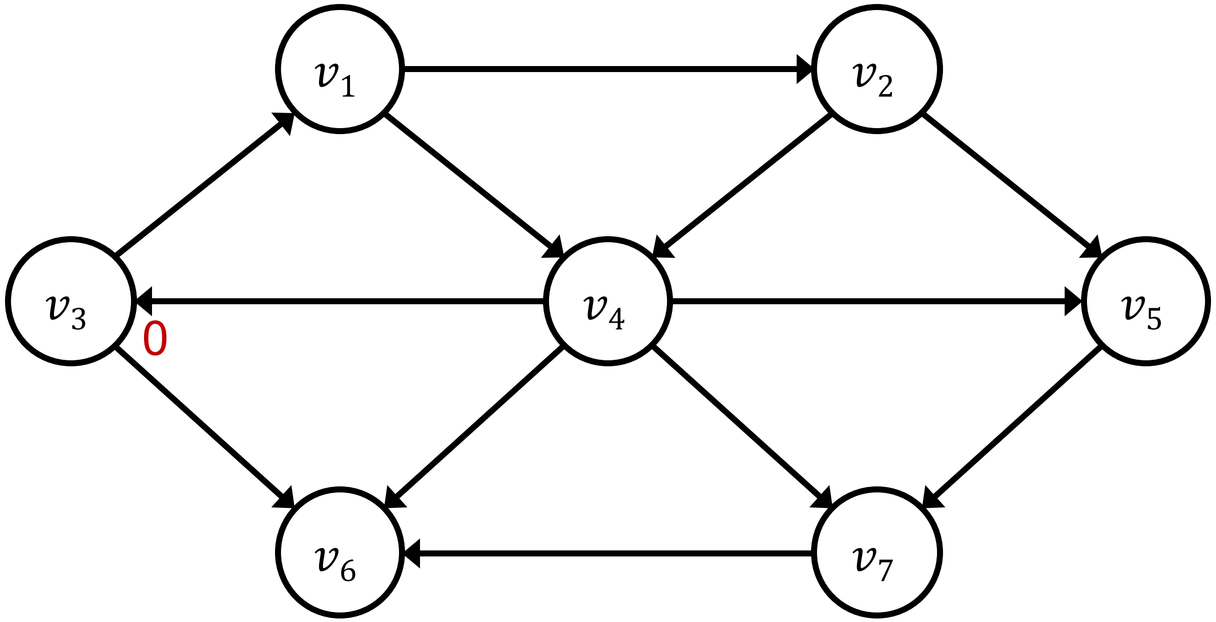
- Given an unweighted graph G, given some vertex s as an input, find the shortest path from s to all other vertices. Given its an unweighted graph we are only interested in the number of edges in the path



- Think of this as a special case of the weighted problem where all the weights are 1

Unweighted Shortest Path Problem

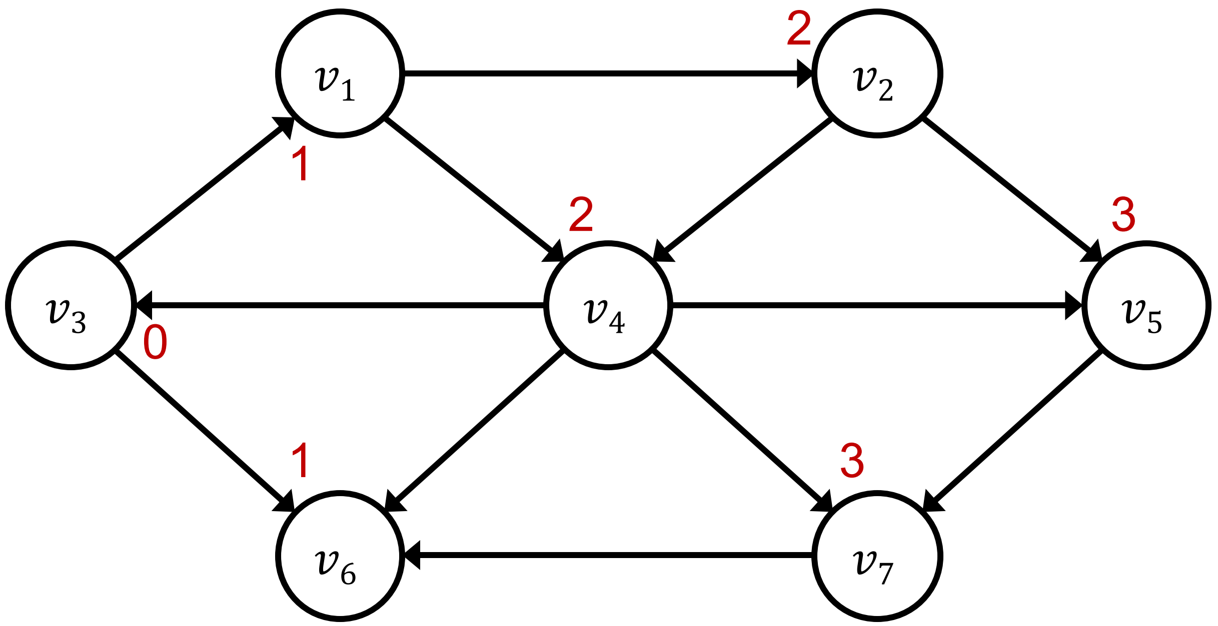
- Only think about the length not the path itself
- If s is v_3, then short path is from v_3 to itself and its length is 0



- Now look for all the vertices that are distance 1 away from s, that is vertices that are adjacent to s (v_3) – What vertices are adjacent to v_3?

Unweighted Shortest Path Problem

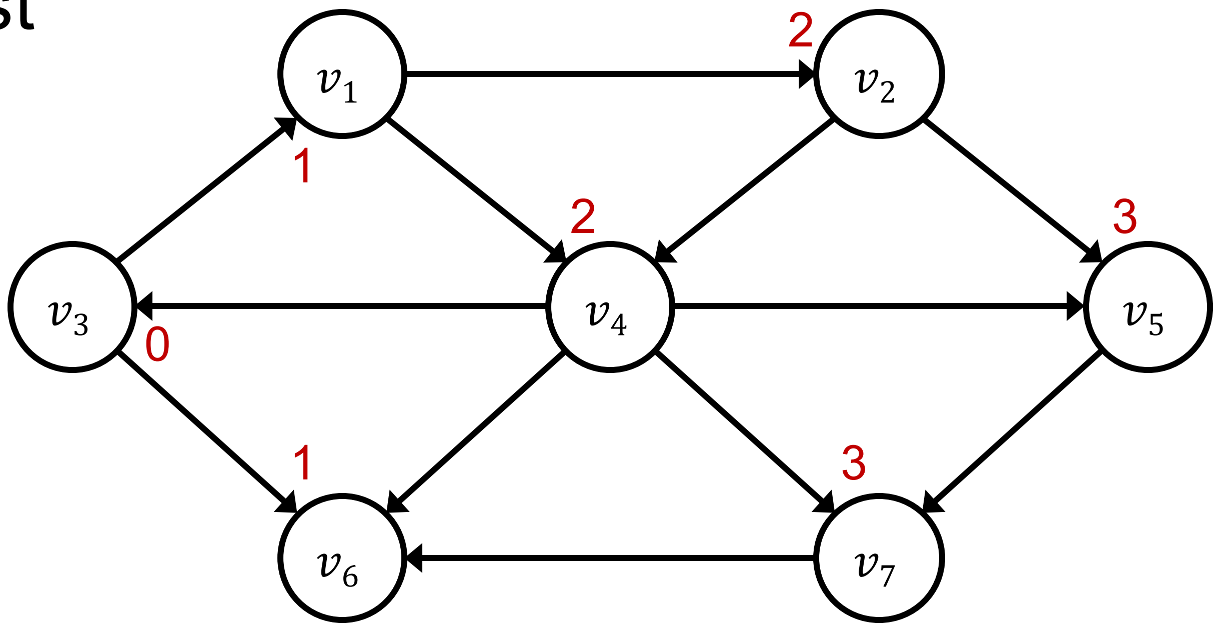
- What vertices are adjacent to v_3? v_1 and v_6 and we can add 1



- Continue until all vertices have been reached.

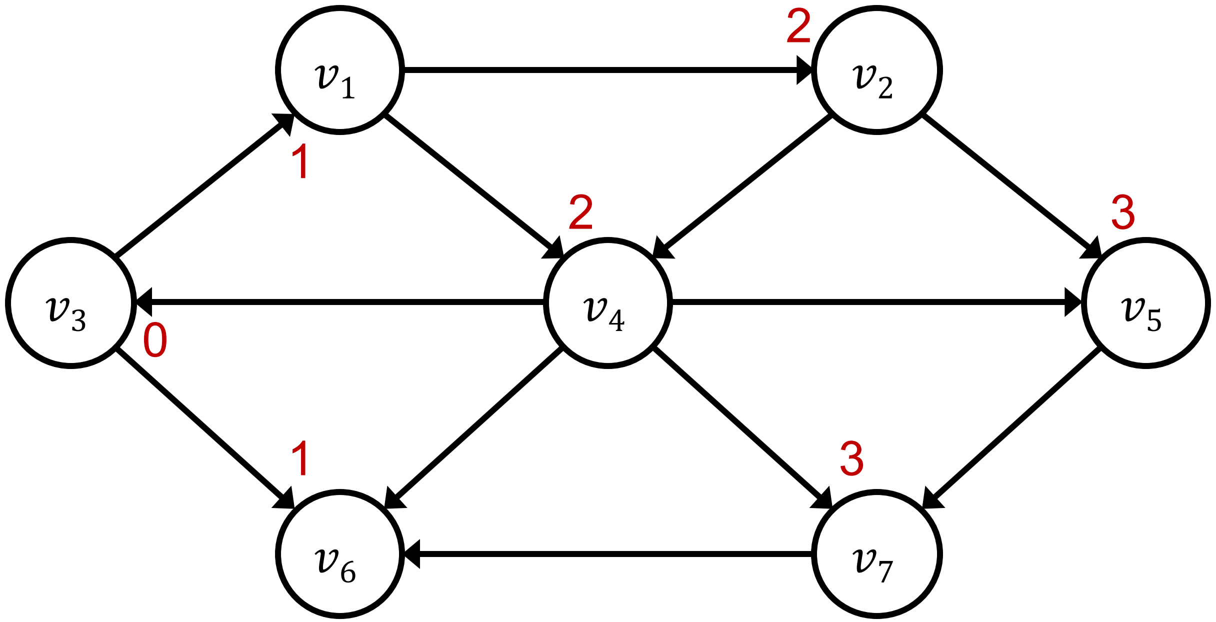
Unweighted Shortest Path Problem

- This should look familiar it is a **breadth-first search.** It operates by processing all the vertices in layers
- The vertices closest to the start are evaluated first, then the next layer and so on until the most distance vertices are evaluated last



Unweighted Shortest Path Problem

- How would you translate this into code?

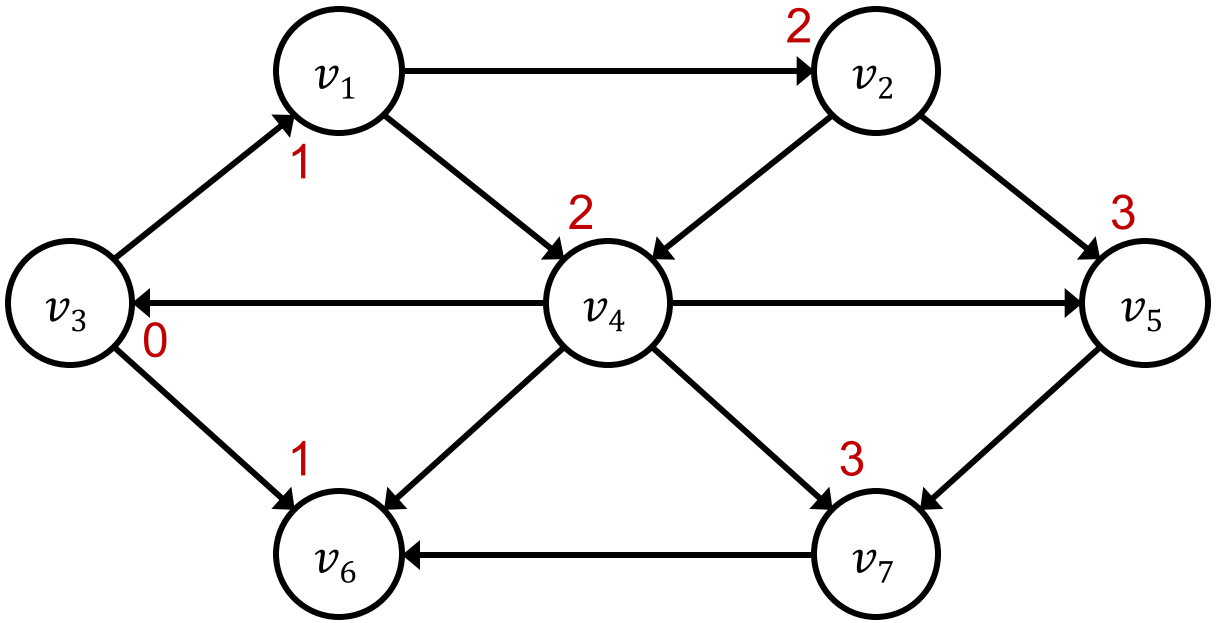


Unweighted Shortest Path Problem

- Construct a table, where d_v is the distance from s (initially unknown and set to ∞, except s itself set to 0)

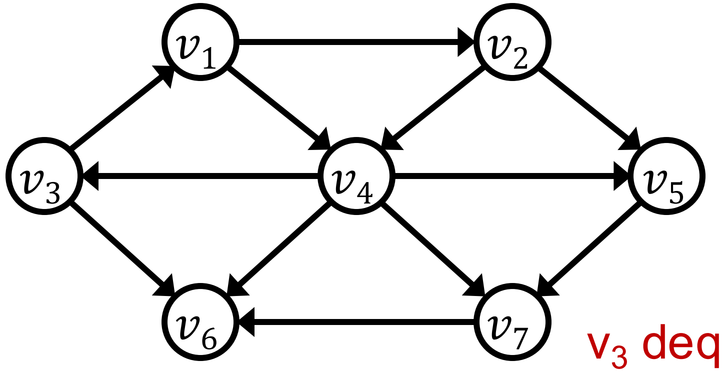
Q:
v_3

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



known is initially set to F, after being visited it will be set to T, $\mathbf{d_v}$ is distance and $\mathbf{p_v}$ is a bookkeeping variable

Unweighted Shortest Path Problem



- Construct a table, where d_v is the distance from s (initially unknown and set to ∞, except s itself set to 0)

Initial State

Q: v_3

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

v_3 dequeued

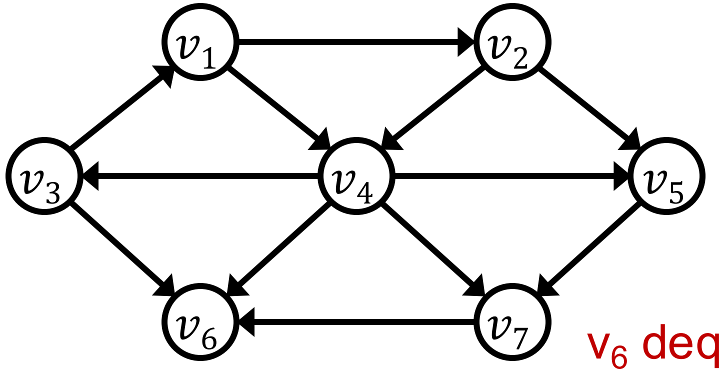
Q: v_1 v_6

V	known	d_v	p_v
v_1	F	1	v_3
v_2	F	∞	0
v_3	T	0	0
v_4	F	∞	0
v_5	F	∞	0
v_6	F	1	v_3
v_7	F	∞	0

known is initially set to F, after being visited it will be set to T, **d_v** is distance and **p_v** is a bookkeeping variable

Unweighted Shortest Path Problem

- Construct a table, where d_v is the distance from s (initially unknown and set to ∞, except s itself set to 0)



v_1 dequeued

Q: v_6, v_2, v_4

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

v_6 dequeued

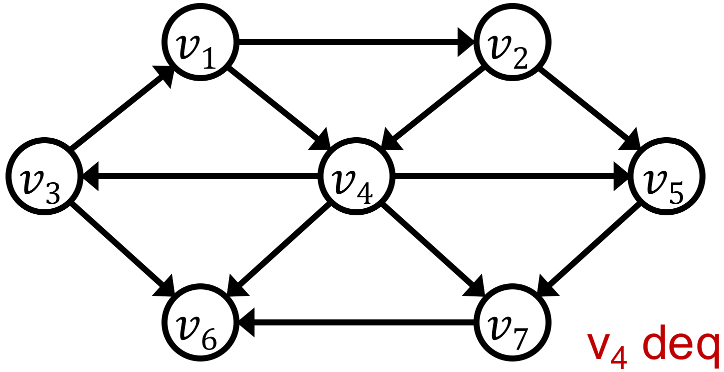
Q: v_2, v_4

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

known is initially set to F, after being visited it will be set to T, $\mathbf{d_v}$ is distance and $\mathbf{p_v}$ is a bookkeeping variable

Unweighted Shortest Path Problem

- Construct a table, where d_v is the distance from s (initially unknown and set to ∞, except s itself set to 0)



v_2 dequeued

Q: v_4, v_5

V	known	d_v	p_v
v_1	T	1	v_3
v_2	T	2	v_1
v_3	T	0	0
v_4	F	2	v_1
v_5	F	3	v_2
v_6	T	1	v_3
v_7	F	∞	0

v_4 dequeued

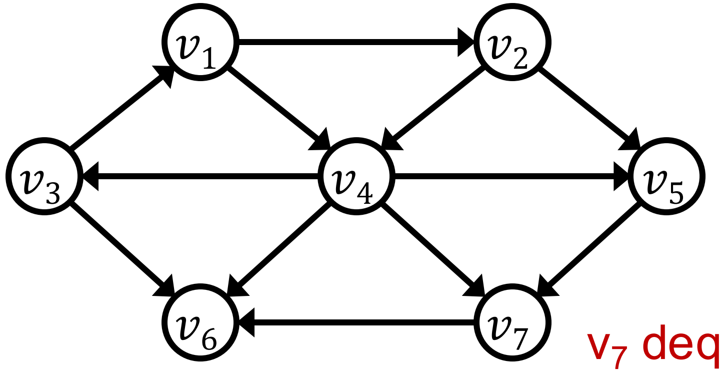
Q: v_5, v_7

V	known	d_v	p_v
v_1	T	1	v_3
v_2	T	2	v_1
v_3	T	0	0
v_4	T	2	v_1
v_5	F	3	v_2
v_6	T	1	v_3
v_7	F	3	v_4

known is initially set to F, after being visited it will be set to T, $\mathbf{d_v}$ is distance and $\mathbf{p_v}$ is a bookkeeping variable

Unweighted Shortest Path Problem

- Construct a table, where d_v is the distance from s (initially unknown and set to ∞, except s itself set to 0)



v_5 dequeued

Q: v_7

V	known	d_v	p_v
v_1	T	1	v_3
v_2	T	2	v_1
v_3	T	0	0
v_4	T	2	v_1
v_5	T	3	v_2
v_6	T	1	v_3
v_7	F	3	v_4

v_7 dequeued

Q:

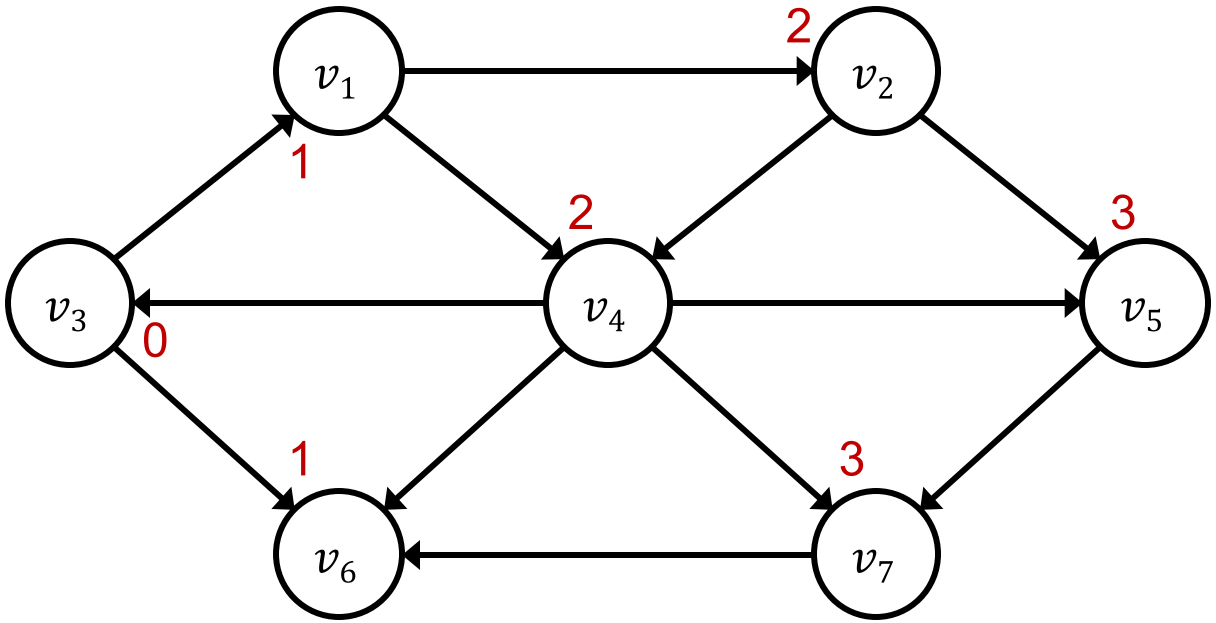
V	known	d_v	p_v
v_1	T	1	v_3
v_2	T	2	v_1
v_3	T	0	0
v_4	T	2	v_1
v_5	T	3	v_2
v_6	T	1	v_3
v_7	T	3	v_4

known is initially set to F, after being visited it will be set to T, $\mathbf{d_v}$ is distance and $\mathbf{p_v}$ is a bookkeeping variable

Unweighted Shortest Path Problem

- Construct a table, where d_v is the distance from s (initially unknown and set to ∞, except s itself set to 0)   **Breadth-First Search**

V	known	d_v	p_v
v_1	T	1	v_3
v_2	T	2	v_1
v_3	T	0	0
v_4	T	2	v_1
v_5	T	3	v_2
v_6	T	1	v_3
v_7	T	3	v_4



known is initially set to F, after being visited it will be set to T, $\mathbf{d_v}$ is distance and $\mathbf{p_v}$ is a bookkeeping variable

Unweighted Shortest Path Problem

```
void
unweighted(Vertex s)
{
    for each Vertex v
    {
        v.dist = INFINITY;
        v.known = false;
    }
    s.dist = 0;
    for(int currDist = 0;
            currDist < NUM_VERTICES;
            currDist++)
    {
        for each Vertex v
        {
            if(!v.known && v.dist == currDist)
            {
                v.known = true
                for each Vertex w adjacent to v
                {
                    if(w.dist == INFINITY)
                    {
                        w.dist = currDist + 1
                        w.path = v;
                    }// End of IF
                }// End of FOR
            }// End of IF
        }// End of FOR
    }// End of FOR
}// End of unweighted()
```

Unweighted Shortest Path Problem

```
void
unweighted(Vertex s)
{
    for each Vertex v
    {
        v.dist = INFINITY;
        v.known = false;
    }
    s.dist = 0;
    for(int currDist = 0;
            currDist < NUM_VERTICES;
            currDist++)
    {
        for each Vertex v
        {
            if(!v.known && v.dist == currDist)
            {
                v.known = true
                for each Vertex w adjacent to v
                {
                    if(w.dist == INFINITY)
                    {
                        w.dist = currDist + 1
                        w.path = v;
                    }// End of IF
                }// End of FOR
            }// End of IF
        }// End of FOR
    }// End of FOR
}// End of unweighted()
```

Initialize table

$O(V^2)$ algorithm

Weighted Shortest Path Problem

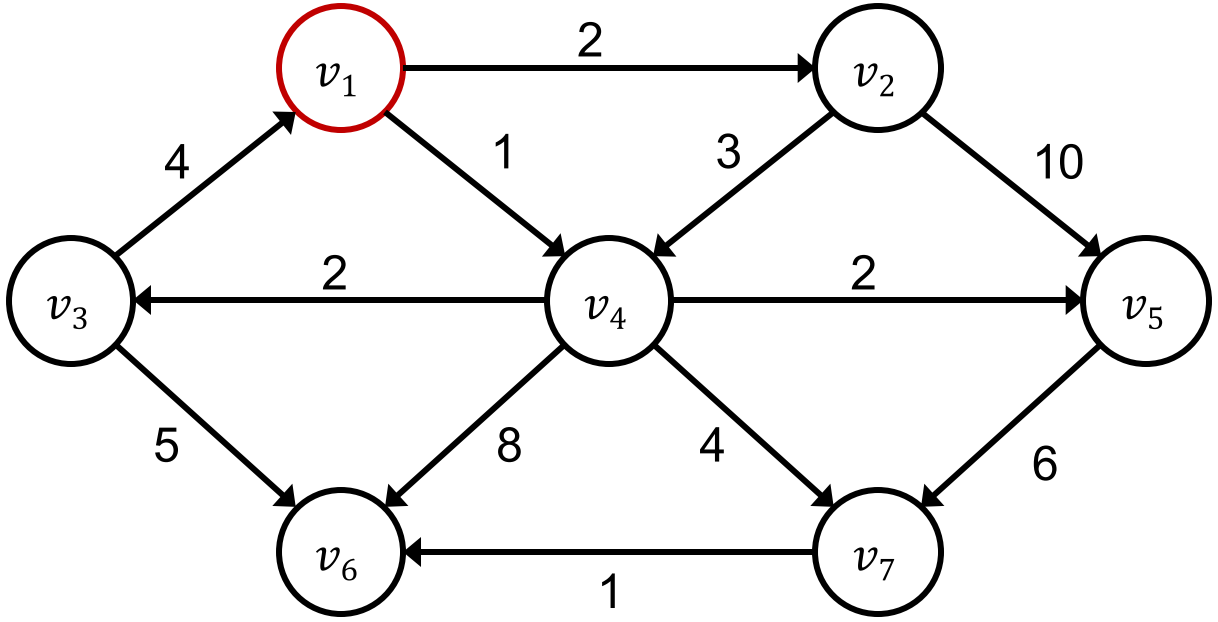
- Using what we know from unweighted, we can now tackle weighted shorted path – p_v is the last vertex to cause a change to d_v, this solution is known as ***Dijkstra's Algorithm*** and it's an example of a **Greedy Algorithm**
- **Greedy algorithms** solve a problem in stages doing what appears to be the best thing at each stage
- Greedy Example: making change at checkout counter – first the checkout person gives quarters, then dimes, then nickels and then pennies
  - $0.79, 3 quarters and 4 pennies
  - $0.87, 3 quarters, 1 dime and two pennies
  - Minimizing the number of coins given …

Dijkstra's Algorithm

- We choose a v that has the smallest d_v from all unknown vertices and is adjacent to s
- This path is declared the shortest path from s to v and marked known
- The remaining step is updating d_w (we didn't track d_w before, as we just were thinking $d_w = dv + 1$ if $d_w = \infty$) and $d_w = dv + c_{v,w}$ if this new value for d_w would be an improvement
- The algorithm decides if it's a good idea or not to use v on path to w given known cost and new cost

Dijkstra's Algorithm

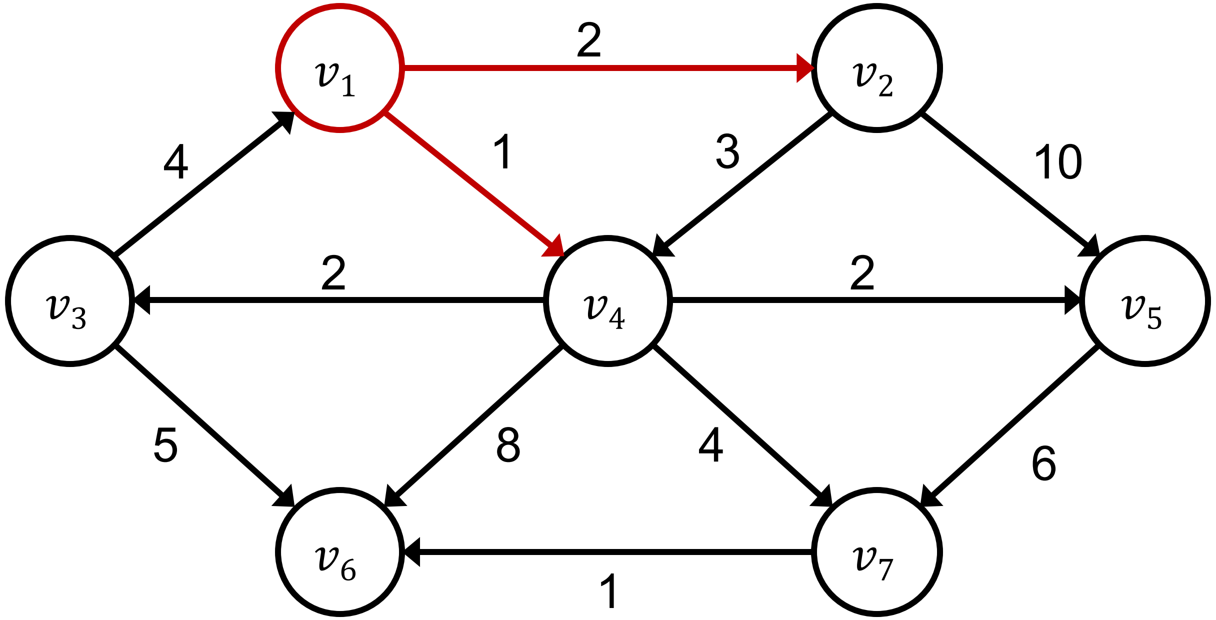
V	known	d_v	p_v
v_1	F	0	0
v_2	F	∞	0
v_3	F	∞	0
v_4	F	∞	0
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0



Pick s to be v_1, the path to v_1 is 0

Dijkstra's Algorithm

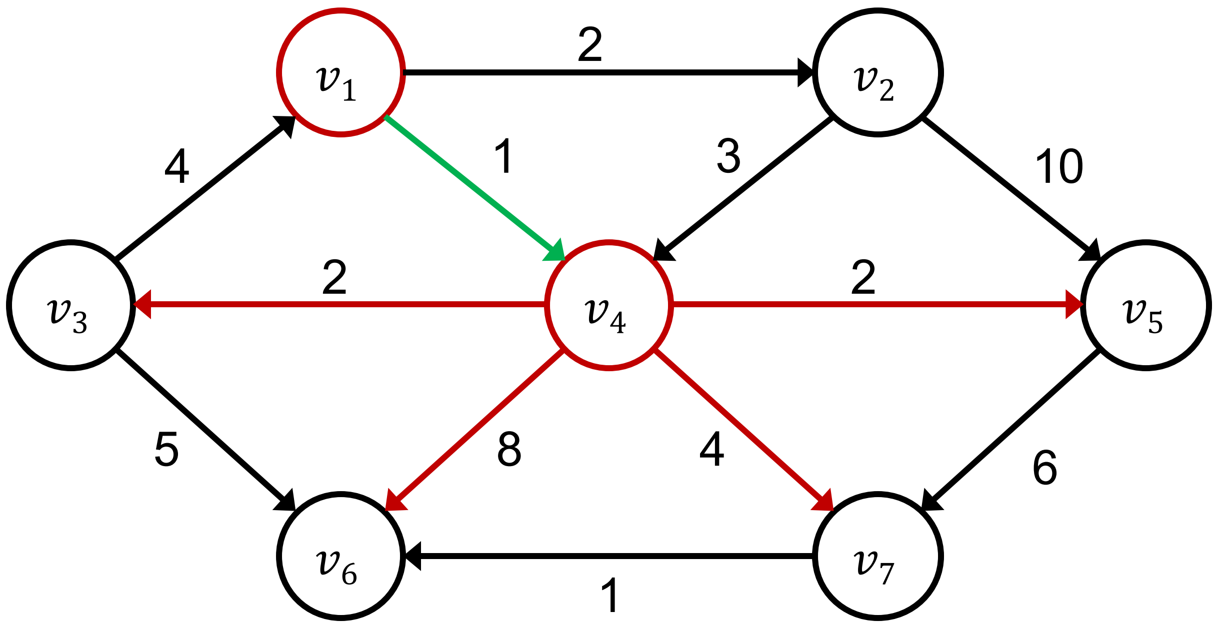
V	known	d_v	p_v
v_1	T	0	0
v_2	F	2	v_1
v_3	F	∞	0
v_4	F	1	v_1
v_5	F	∞	0
v_6	F	∞	0
v_7	F	∞	0



From v_1 we have path to v_2 and v_4, we choose v_4 (why?)

Dijkstra's Algorithm

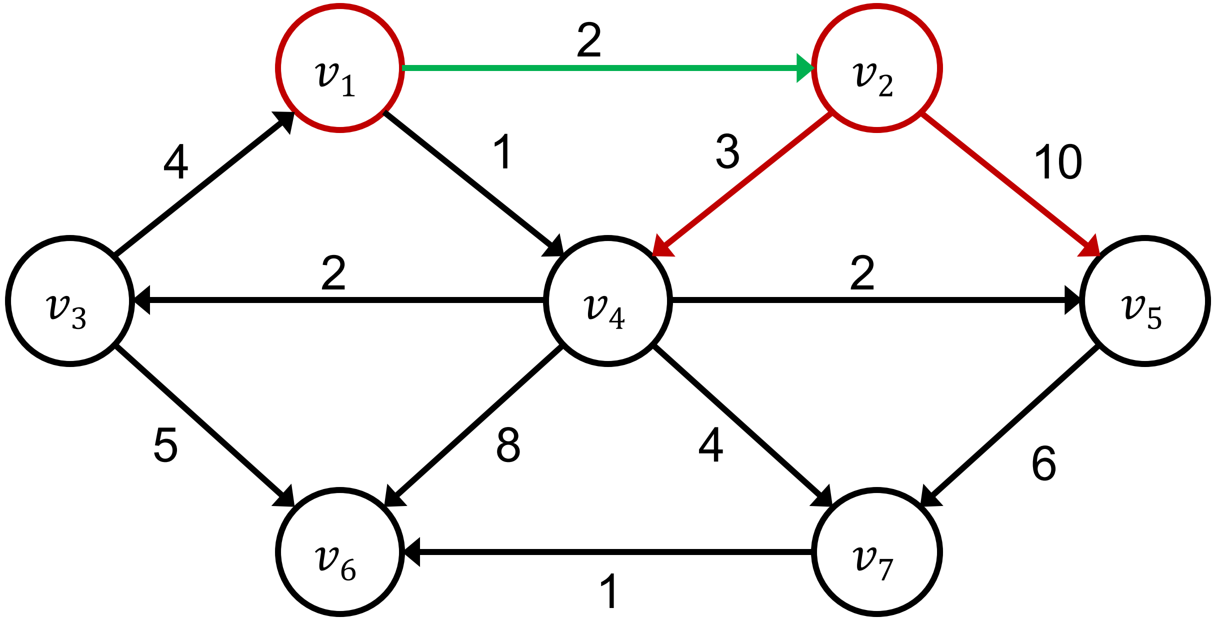
V	known	d_v	p_v
v_1	T	0	0
v_2	F	2	v_1
v_3	F	$3\ (1+2)$	v_4
v_4	T	1	v_1
v_5	F	$3\ (1+2)$	v_4
v_6	F	$9\ (1+8)$	v_4
v_7	F	$5\ (1+4)$	v_4



From v_4 we have path to v_3 , v_5, v_6 , v_7, we choose v_2 (new cheapest)

Dijkstra's Algorithm

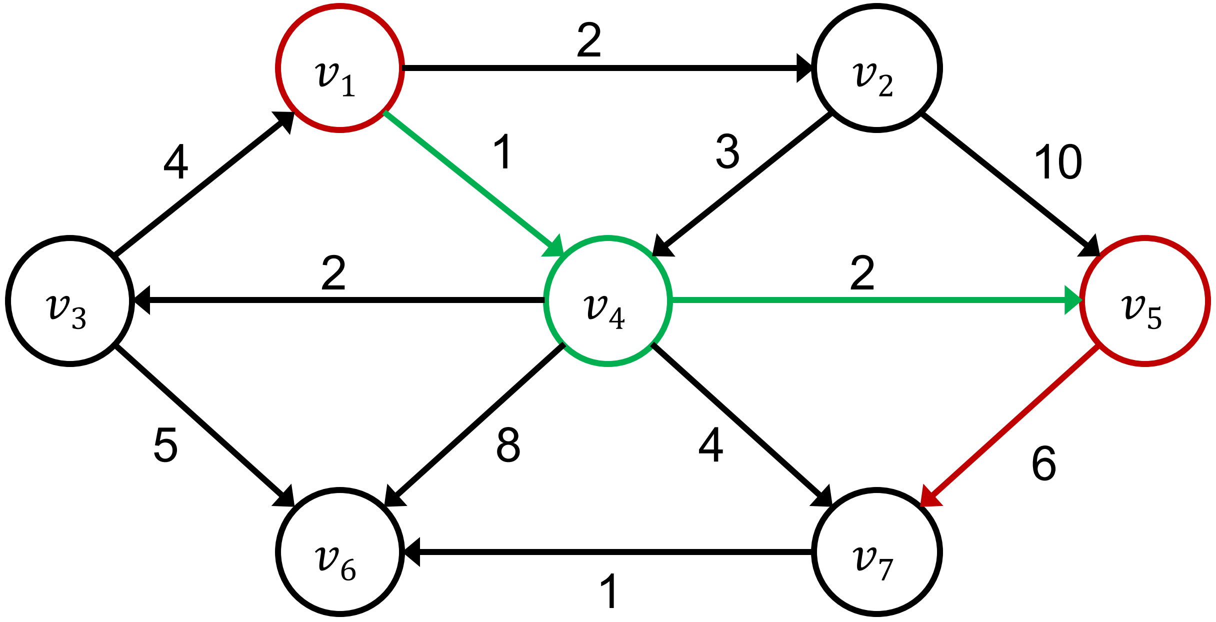
V	known	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	F	$3\ (1+2)$	v_4
v_4	T	1	v_1
v_5	F	$3\ (1+2)$	v_4
v_6	F	$9\ (1+8)$	v_4
v_7	F	$5\ (1+4)$	v_4



From v_2 we have path to v_4 , v_5 we look at v_5 (since v_4 is already known) none of the paths are better, v_1 to v_2 to v_5 costs 2 + 10 = 12 > 3

Dijkstra's Algorithm

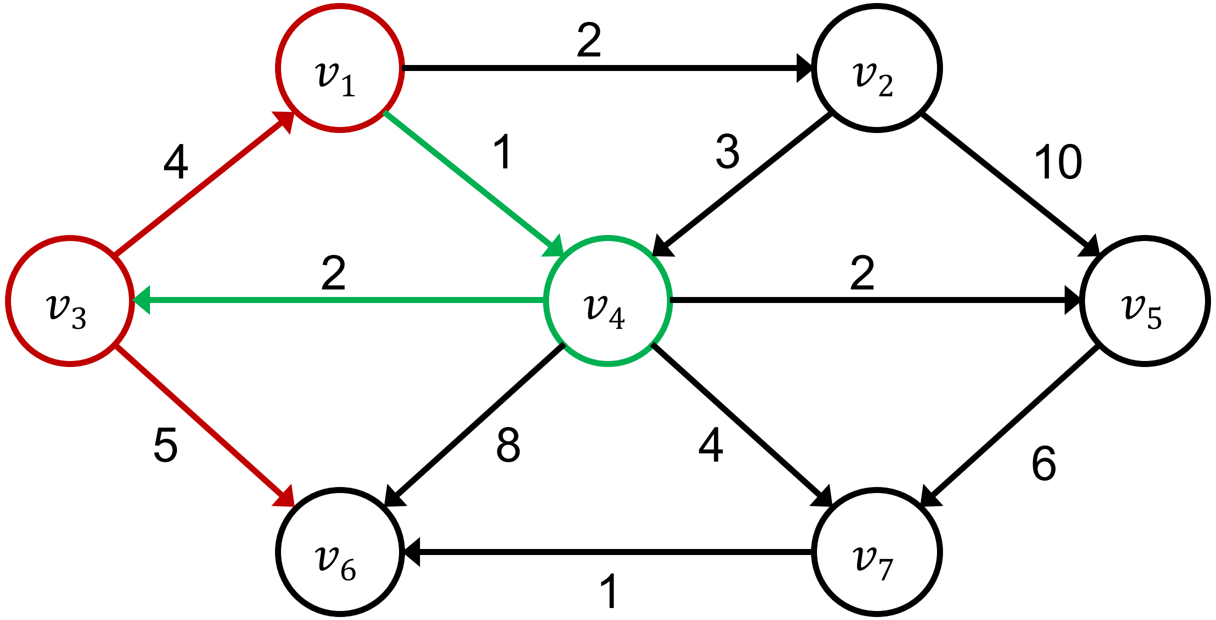
V	known	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	F	$3\ (1+2)$	v_4
v_4	T	1	v_1
v_5	T	$3\ (1+2)$	v_4
v_6	F	$9\ (1+8)$	v_4
v_7	F	$5\ (1+4)$	v_4



From v_5 we have path to v_7 none of the paths are better v_1 to v_4 to v_5, 1 + 2 + 6 = 9 > 5 back to selecting the smallest unvisited node which is v_3

Dijkstra’s Algorithm

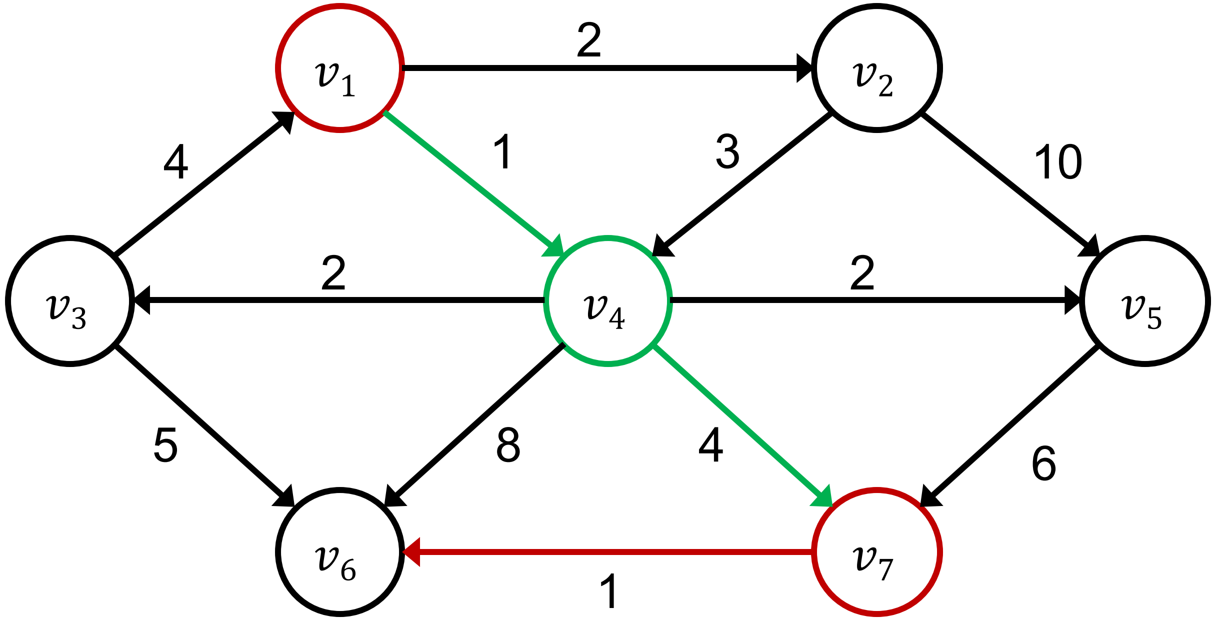
V	known	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	$3\ (1+2)$	v_4
v_4	T	1	v_1
v_5	T	$3\ (1+2)$	v_4
v_6	F	$8\ (1+2+5)$	v_3
v_7	F	$5\ (1+4)$	v_4



From v_3 we have path to v_1 , v_6 with v_1 cost is 1 + 2 + 4 = 7 > 0 but v_6 is 1 + 2 + 5 = 8 < 9, so we update, v_7 is now selected as the smallest

Dijkstra's Algorithm

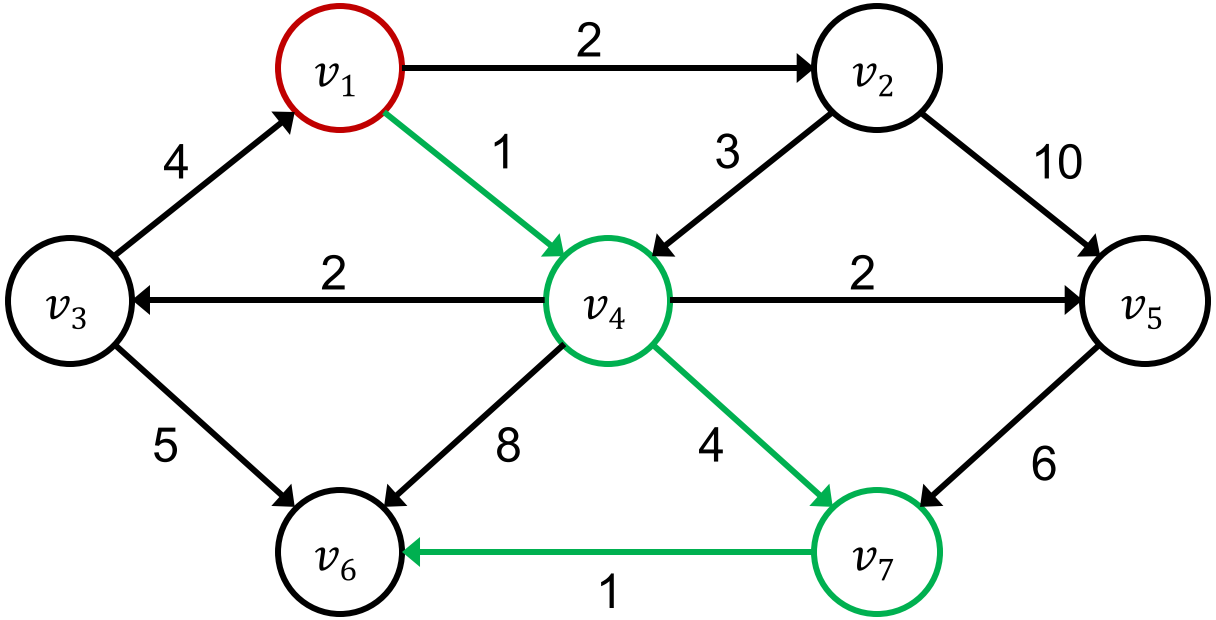
V	known	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	$3\ (1+2)$	v_4
v_4	T	1	v_1
v_5	T	$3\ (1+2)$	v_4
v_6	F	$6\ (1+4+1)$	v_7
v_7	T	$5\ (1+4)$	v_4



From v_7 we have path to v_6 with cost is 1 + 4+ 1 = 6 < 8, so we update and v_6 is the last one for us to visit

Dijkstra's Algorithm

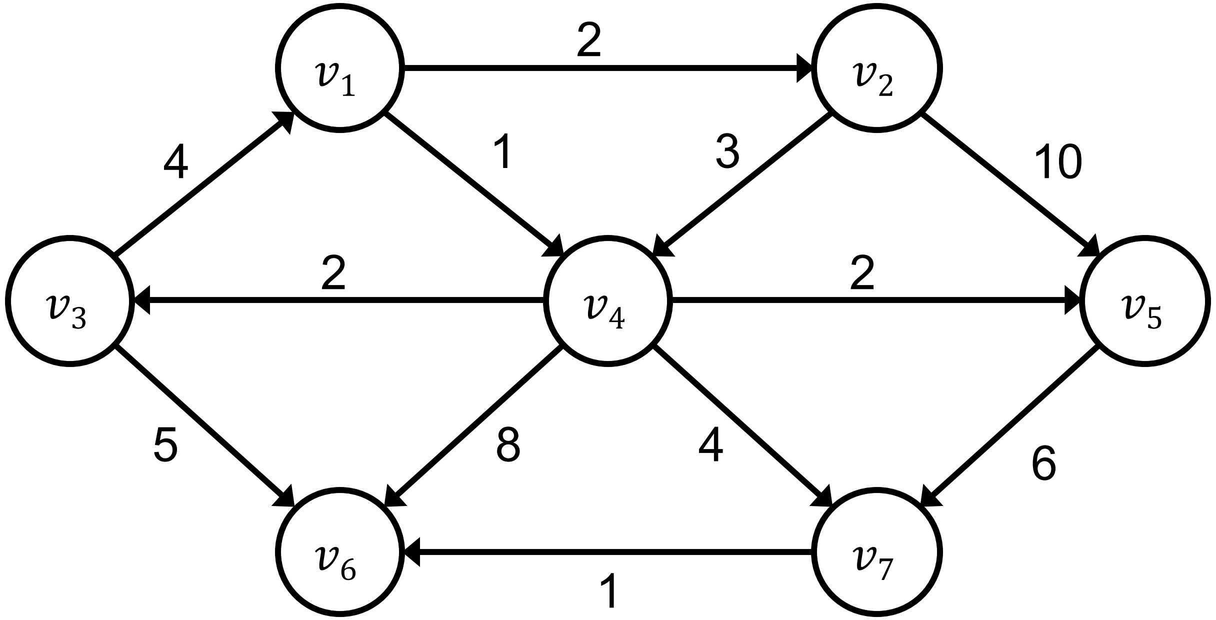
V	known	d_v	p_v
v_1	T	0	0
v_2	T	2	v_1
v_3	T	$3\ (1+2)$	v_4
v_4	T	1	v_1
v_5	T	$3\ (1+2)$	v_4
v_6	T	$6\ (1+4+1)$	v_7
v_7	T	$5\ (1+4)$	v_4



v_6 no where to visit

Dijkstra's Algorithm

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



What is the shortest path from:

v_1 to $v_1 \rightarrow v_1$

v_1 to $v_2 \rightarrow v_1 - v_2$

v_1 to $v_3 \rightarrow v_1 - v_4 - v_3$

v_1 to $v_4 \rightarrow v_1 - v_4$

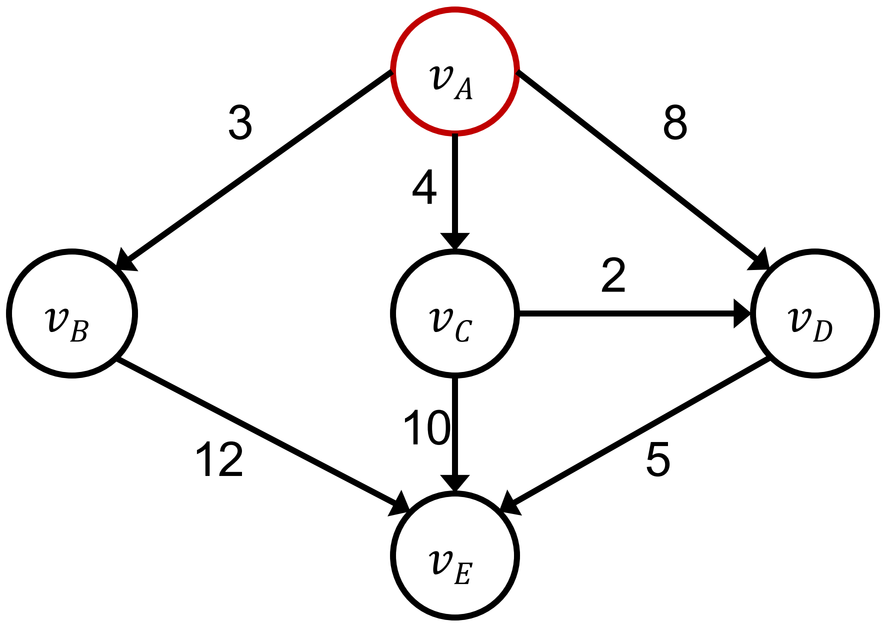
v_1 to $v_5 \rightarrow v_1 - v_4 - v_5$

v_1 to $v_6 \rightarrow v_1 - v_4 - v_7 - v_6$

v_1 to $v_7 \rightarrow v_1 - v_4 - v_7$

Dijkstra's Algorithm (Example 2)

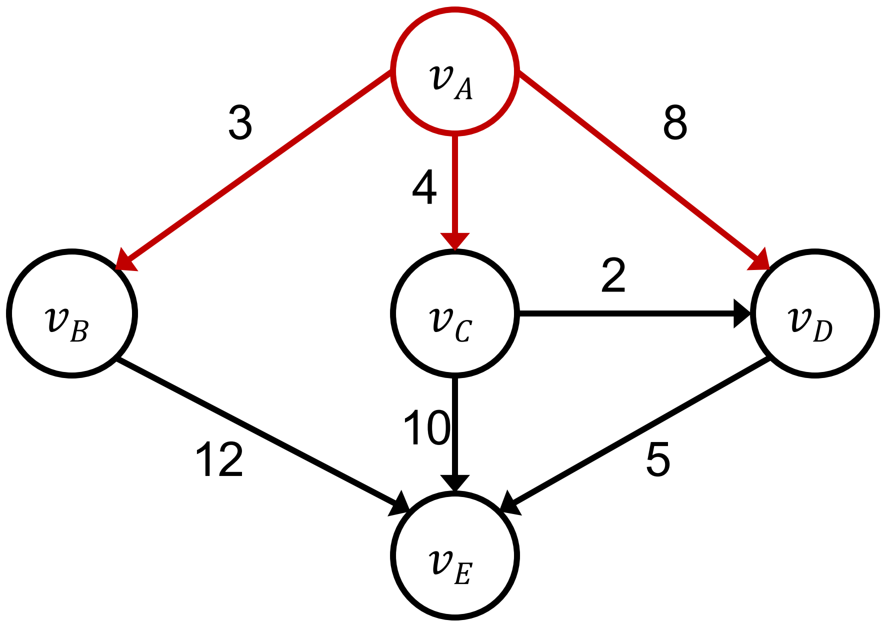
V	known	d_v	p_v
v_A	F	0	0
v_B	F	∞	0
v_C	F	∞	0
v_D	F	∞	0
v_E	F	∞	0



V_A is our starting point (s)

Dijkstra’s Algorithm (Example 2)

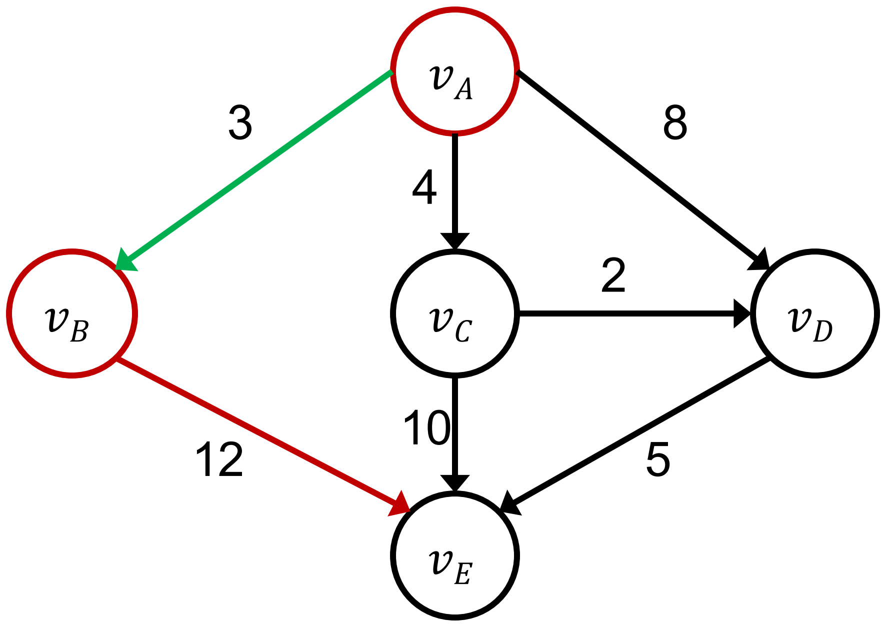
V	known	d_v	p_v
v_A	T	0	v_A
v_B	F	3	v_A
v_C	F	4	v_A
v_D	F	8	v_A
v_E	F	∞	0



V_A is our starting point (s), move to V_B

Dijkstra's Algorithm (Example 2)

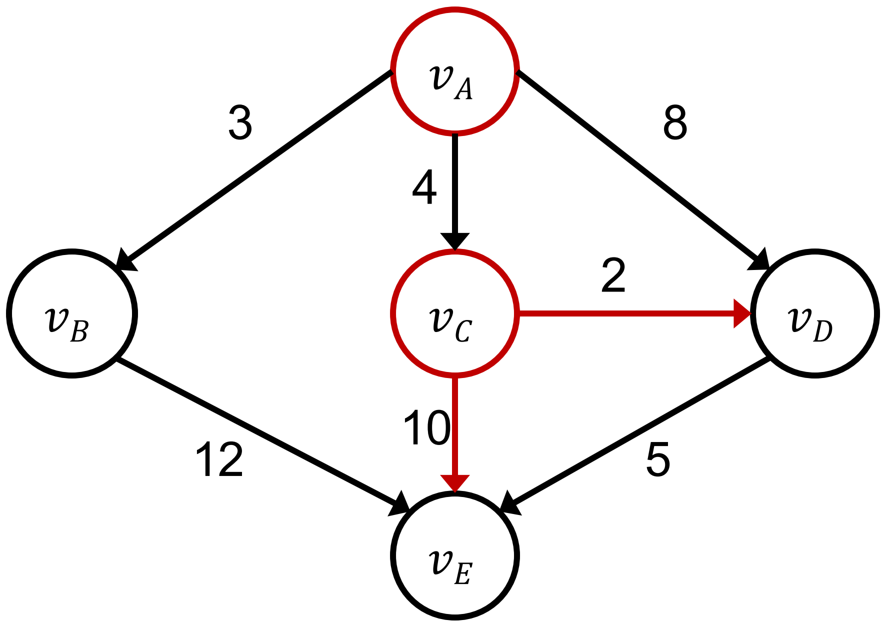
V	known	d_v	p_v
v_A	T	0	v_A
v_B	T	3	v_A
v_C	F	4	v_A
v_D	F	8	v_A
v_E	F	15	V_B



V_A is our starting point (s), move to V_B then on to V_C

Dijkstra's Algorithm (Example 2)

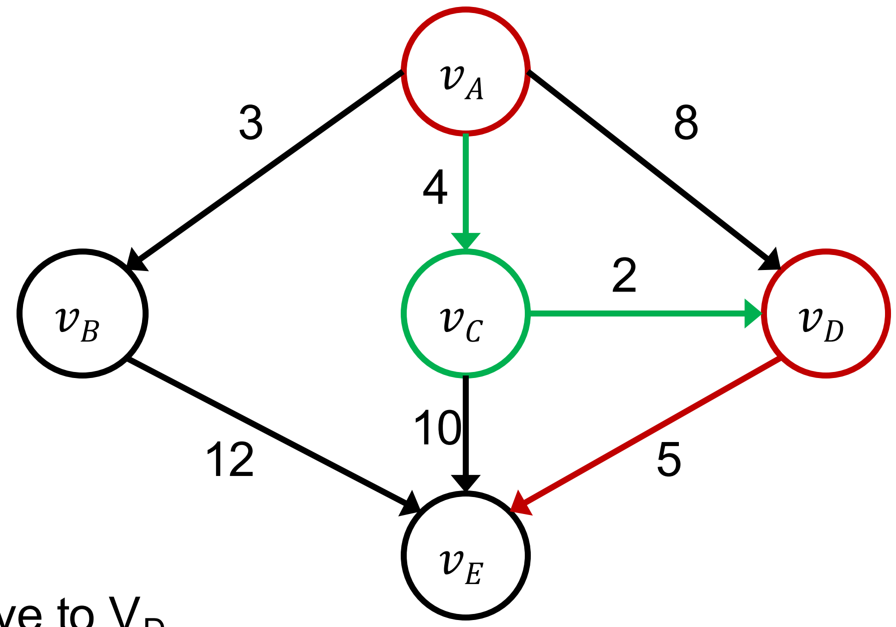
V	known	d_v	p_v
v_A	T	0	v_A
v_B	T	3	v_A
v_C	T	4	v_A
v_D	F	6	v_C
v_E	F	14	v_C



V_A is our starting point (s), move to V_B then on to V_C
Update V_D since it is a shorter path 6 < 8
Update V_E since it is a shorter path 4 + 10 = 14 < 15, we now move to V_D

Dijkstra's Algorithm (Example 2)

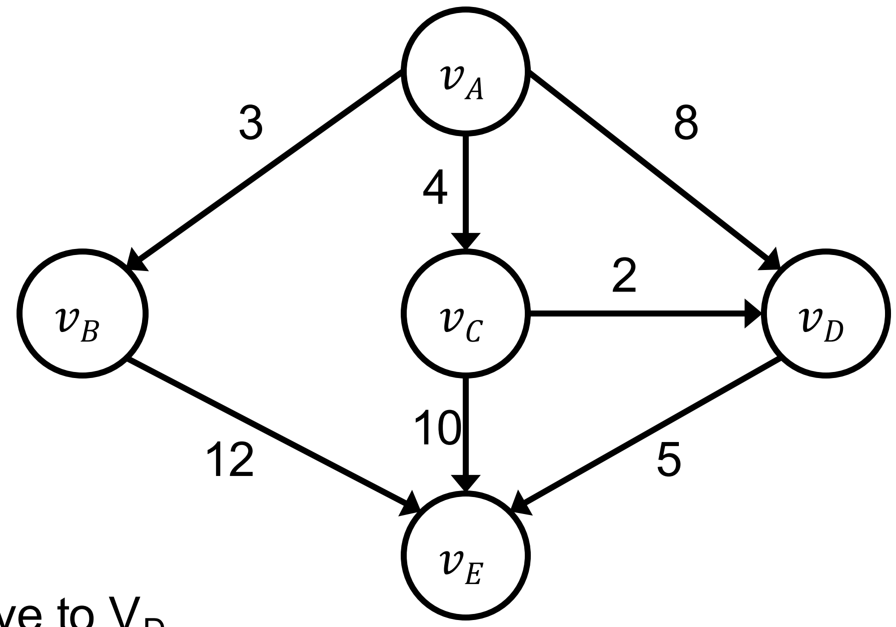
V	known	d_v	p_v
v_A	T	0	v_A
v_B	T	3	v_A
v_C	T	4	v_A
v_D	T	6	v_C
v_E	F	11	v_D



V_A is our starting point (s), move to V_B then on to V_C
Update V_D since it is a shorter path 6 < 8, we now move to V_D
Update path to V_E given 4 + 2 + 5 = 11 < 14, we move on to V_E

Dijkstra's Algorithm (Example 2)

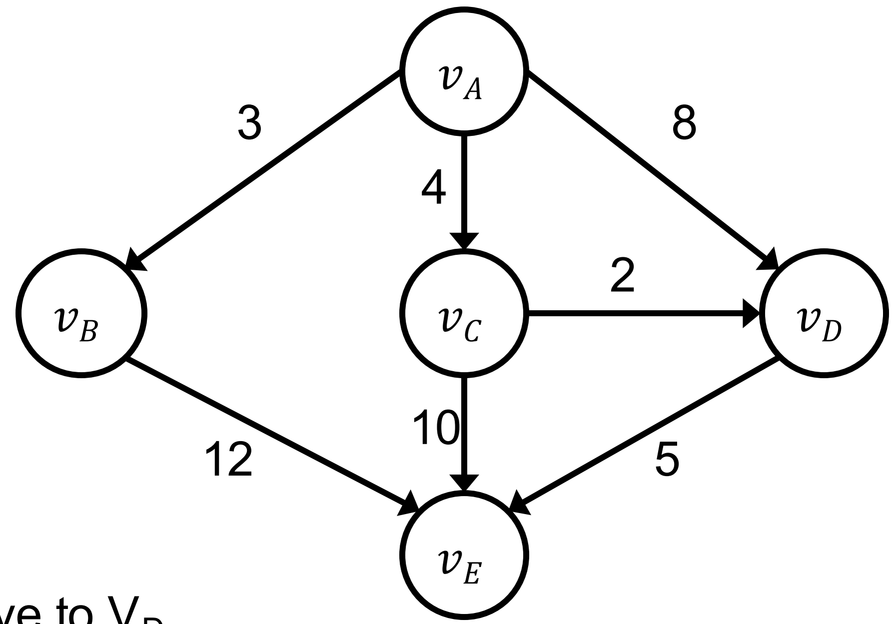
V	known	d_v	p_v
v_A	T	0	v_A
v_B	T	3	v_A
v_C	T	4	v_A
v_D	T	6	v_C
v_E	T	11	V_D



V_A is our starting point (s), move to V_B then on to V_C
Update V_D since it is a shorter path 6 < 8, we now move to V_D
Update path to V_E given 4 + 2 + 5 = 11 < 12, we move on to V_E
V_E can't get anywhere so we are done!

Dijkstra's Algorithm (Example 2)

V	known	d_v	p_v
v_A	T	0	v_A
v_B	T	3	v_A
v_C	T	4	v_A
v_D	T	6	v_C
v_E	T	11	V_D



V_A is our starting point (s), move to V_B then on to V_C
Update V_D since it is a shorter path 6 < 8, we now move to V_D
Update path to V_E given 4 + 2 + 5 = 11 < 12, we move on to V_E
V_E can't get anywhere so we are done!

What is the shortest path from:
V_A to $V_A \rightarrow V_A$
V_A to $V_B \rightarrow V_A - V_B$
V_A to $V_C \rightarrow V_A - V_C$
V_A to $V_D \rightarrow V_A - V_C - V_D$
V_A to $V_E \rightarrow V_A - V_C - V_D - V_E$

Acknowledgement

These slides have been adapted and borrowed from books on the right as well as the CS340 notes of NIU CS department (Professors: Alhoori, Hou, Lehuta, and Winans) and many google searches.