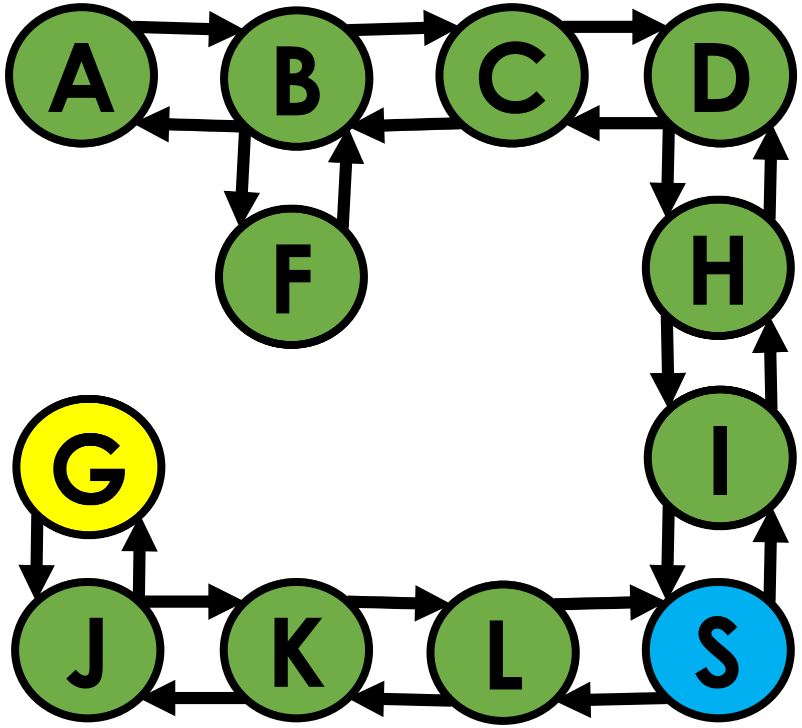
Graph Search

Dijkstra's Algorithm

By the end of this video you will be able to…

- Apply Dijkstra's Algorithm to a weighted graph
- Write the code for Dijkstra
- Describe how ADT Priority Queue works
- Describe how Priority Queues are used in Dijkstra's Algorithm

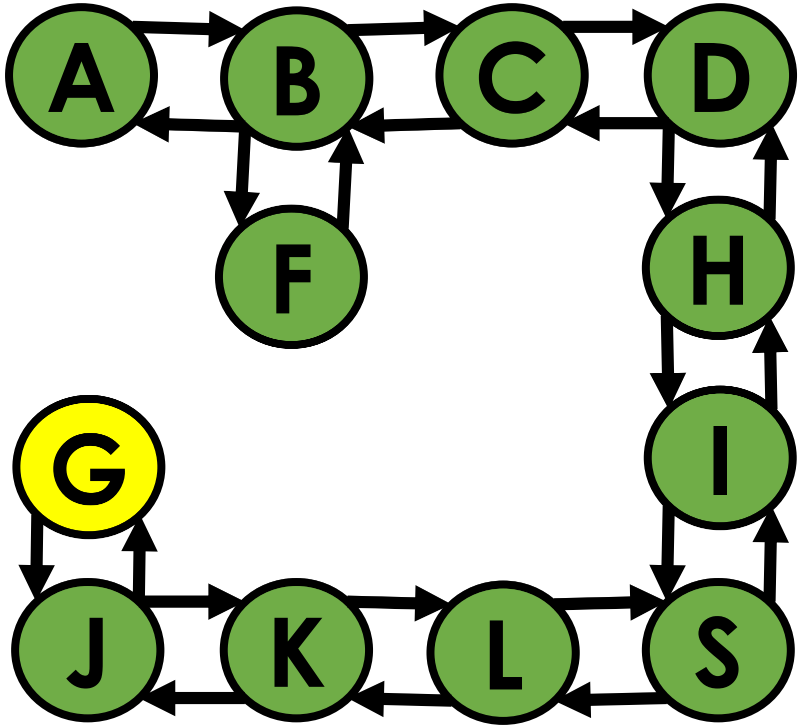
Breadth-first Search (BFS)



How to keep track of where to search next?

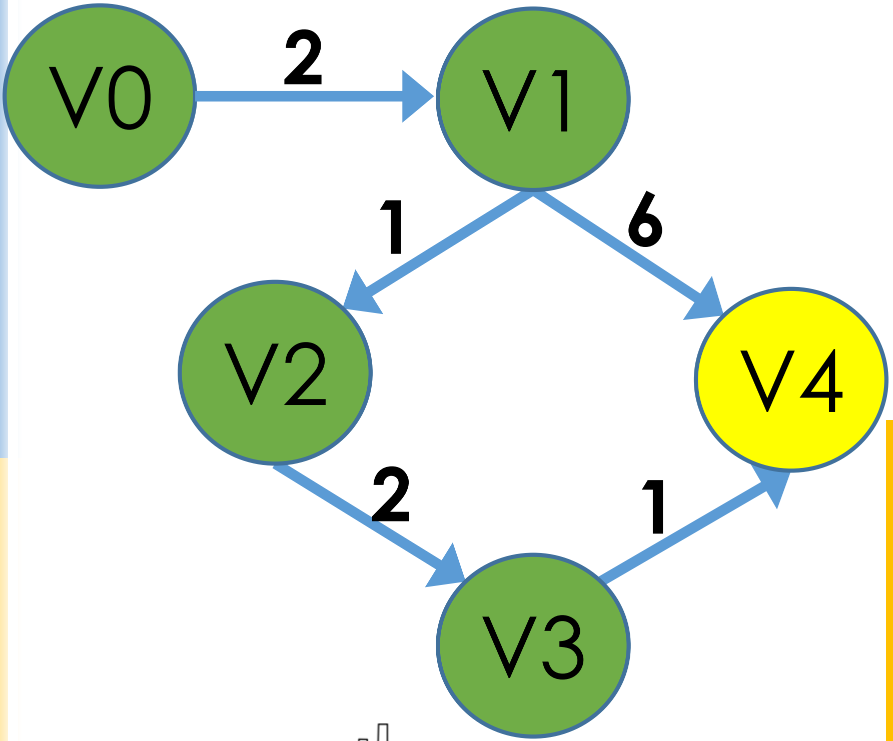
Queue: List where you add elements to one end and remove them from the other
enqueue → add an element
dequeue → remove an element

BFS: Algorithm



```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue
    while queue is not empty:
        dequeue node curr from front of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue:
curr:
visited:



Dijkstra's Algorithm

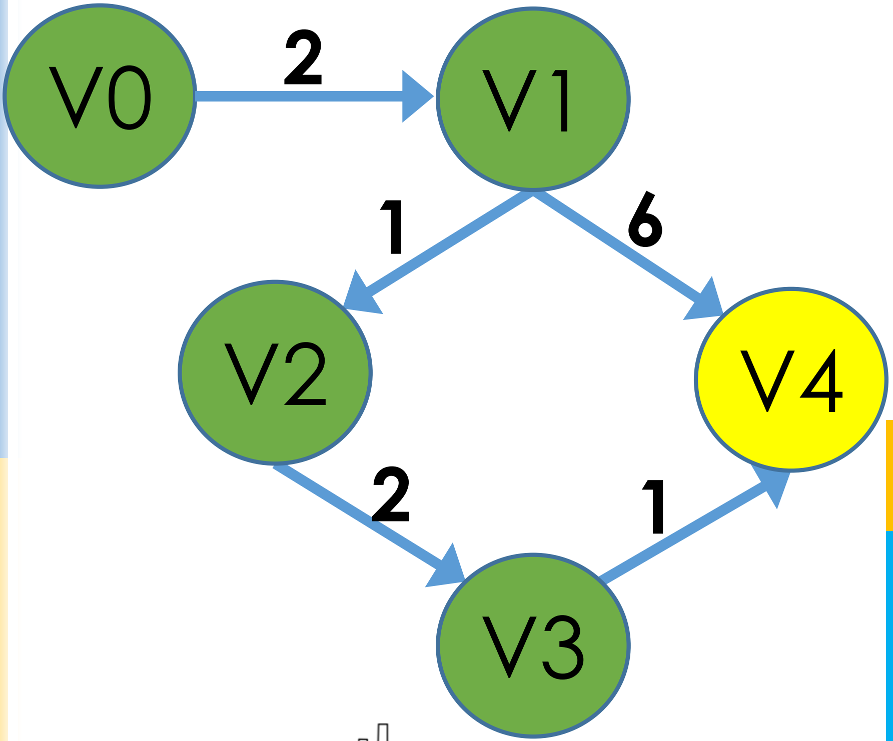
How to keep track of where to search next?

Priority Queue: List where you add an {element, priority} to one end and remove highest priority item from the other

enqueue → add an {element, priority}

dequeue → remove the highest priority element

Dijkstra's Algorithm

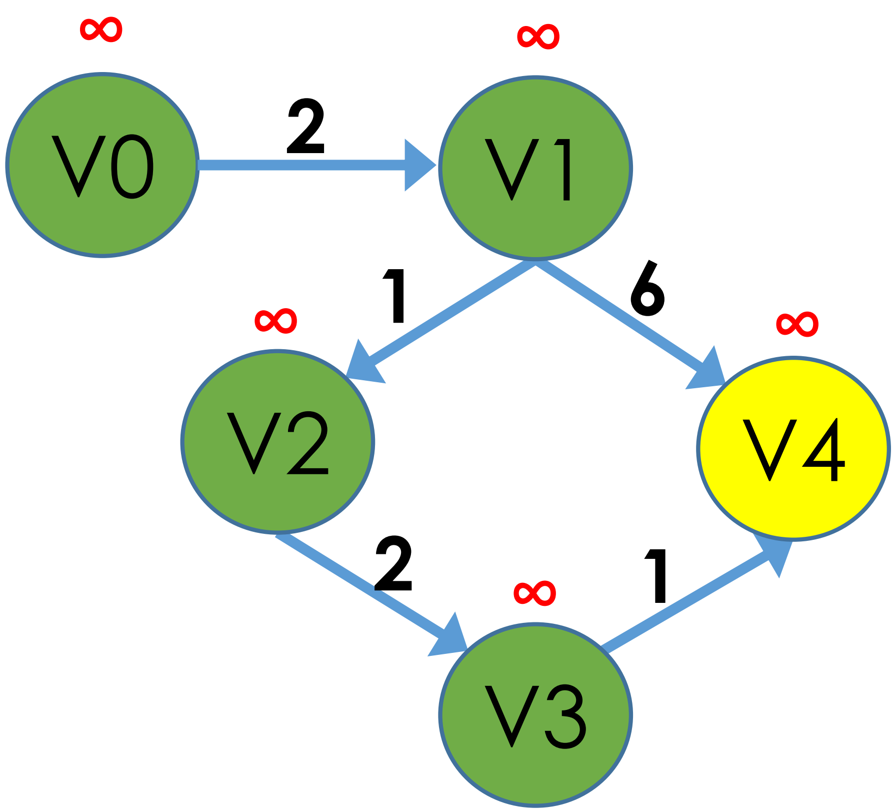


How to keep track of where to search next?

Priority Queue: List where you add an {element,

Priority Queues are often implemented using "Heaps" and can prioritize low values (Min-Heap) or large values (Max-Heap).

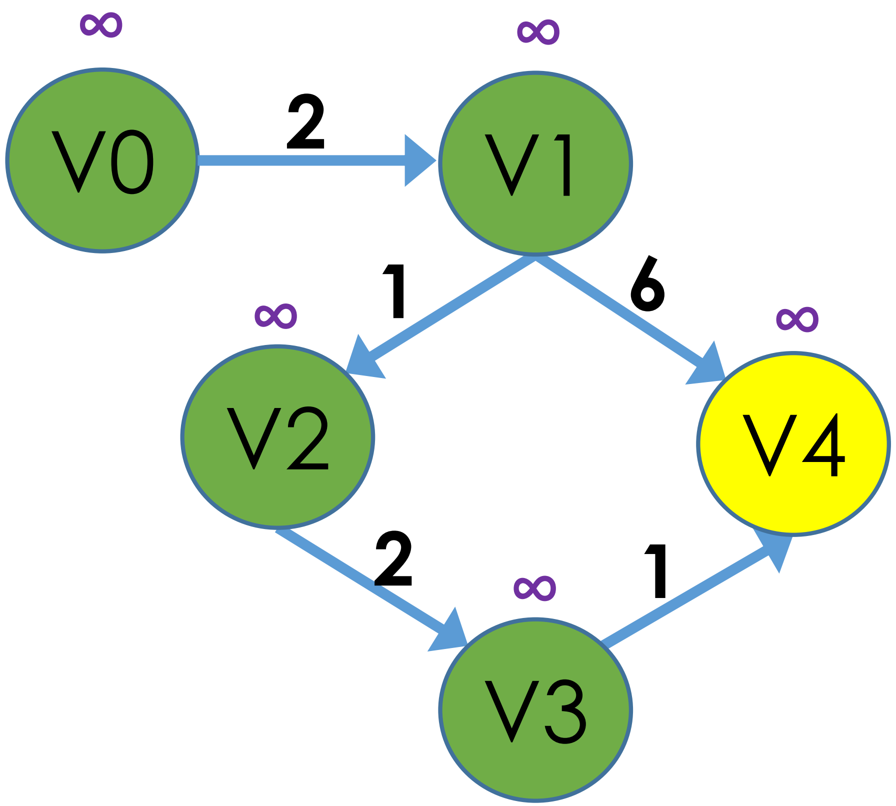
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                ~~add n to visited set~~
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ:
curr:
visited:

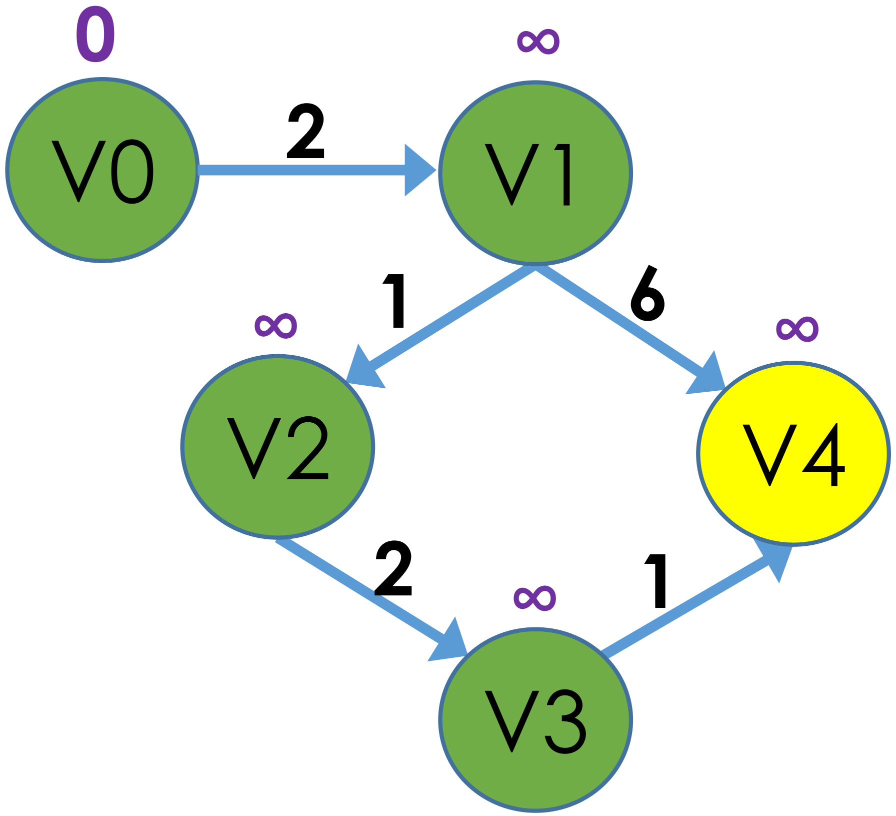
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ:
curr:
visited:

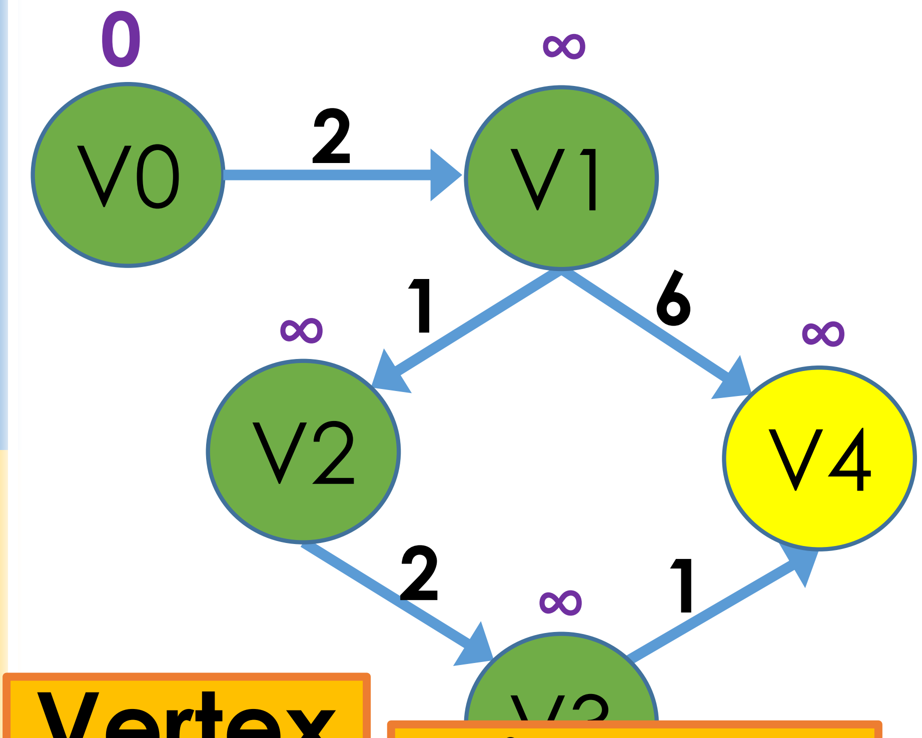
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ: {V0, 0}
curr:
visited:

Dijkstra: Algorithm



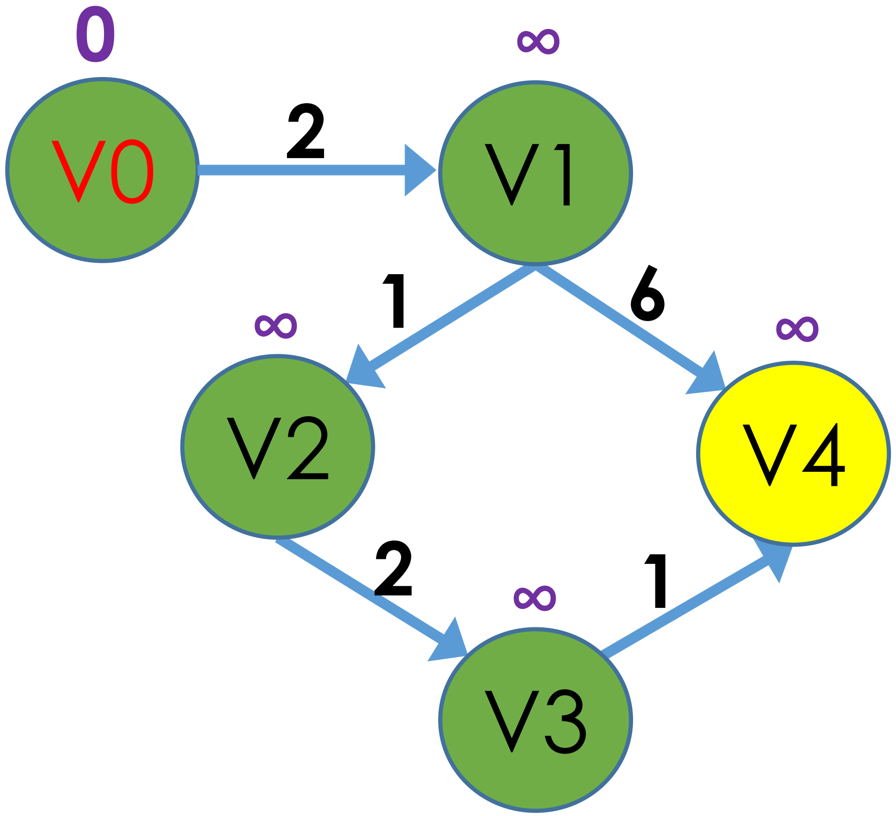
```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

Vertex

Distance

PQ: {V0, 0}
curr:
visited:

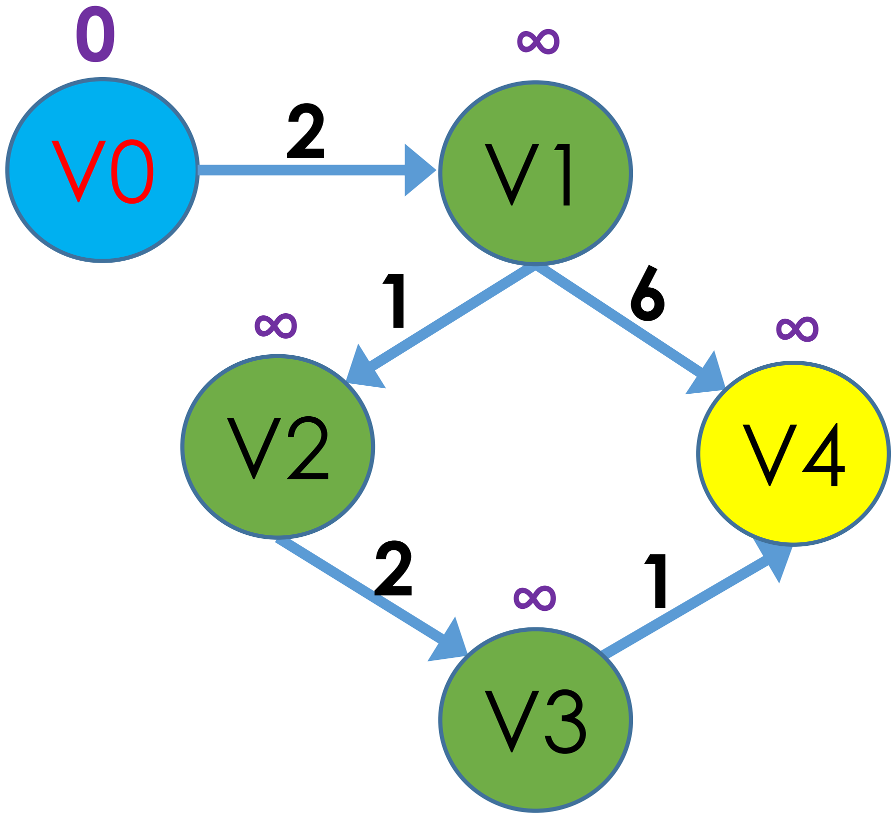
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ: ~~{V0, 0}~~
curr: V0
visited:

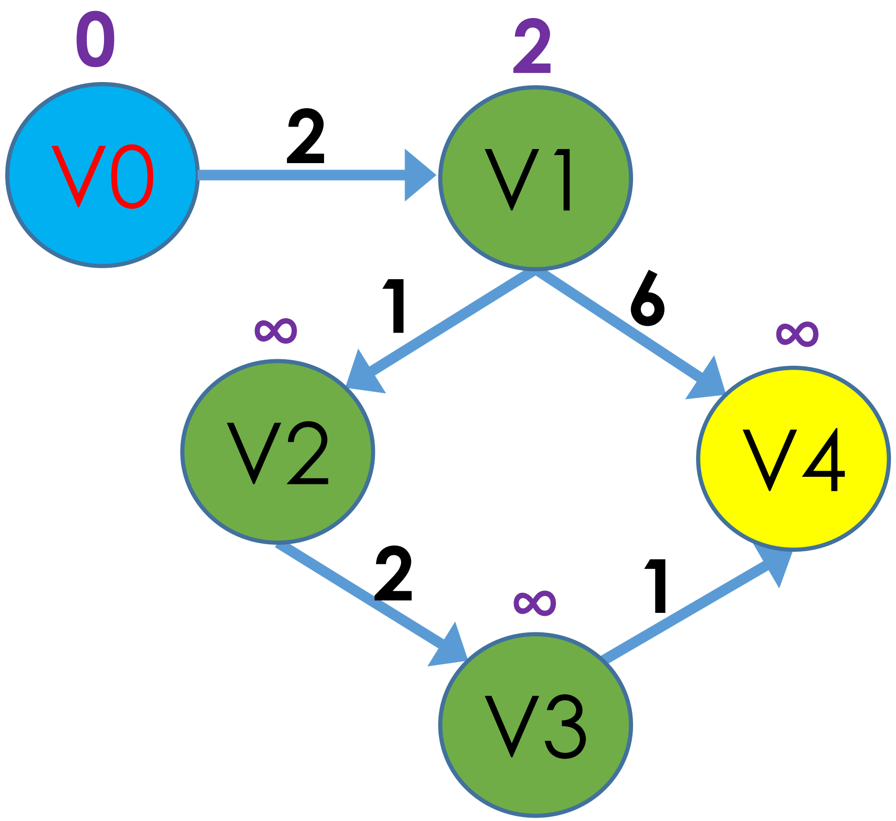
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ:
curr: V0
visited: V0

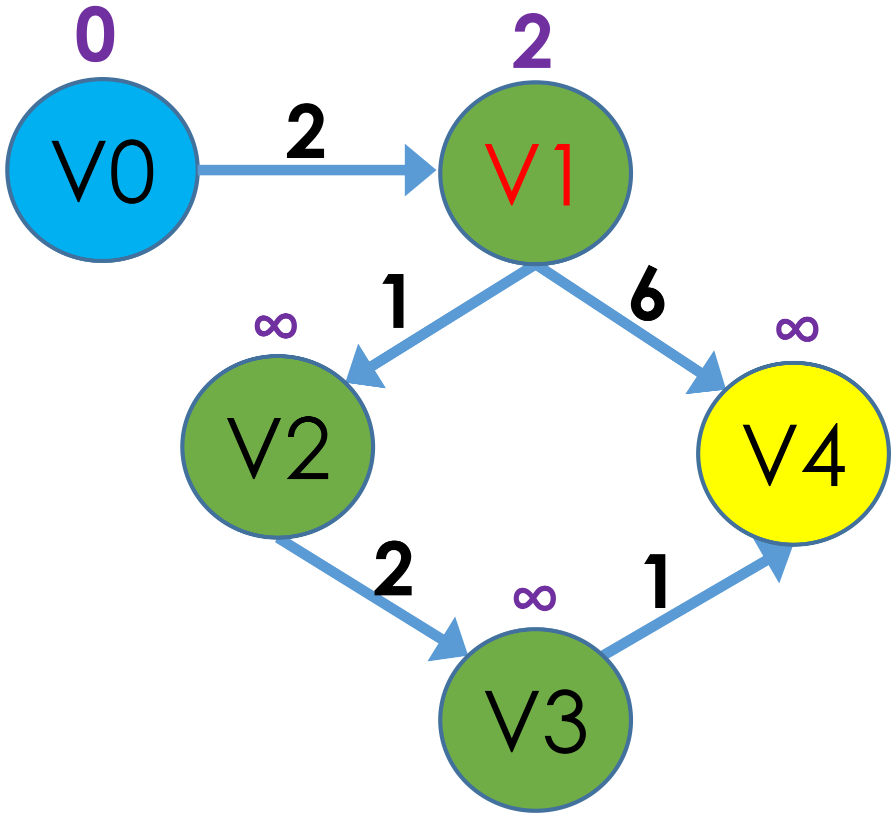
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ: {V1, 2}
curr: V0
visited: V0

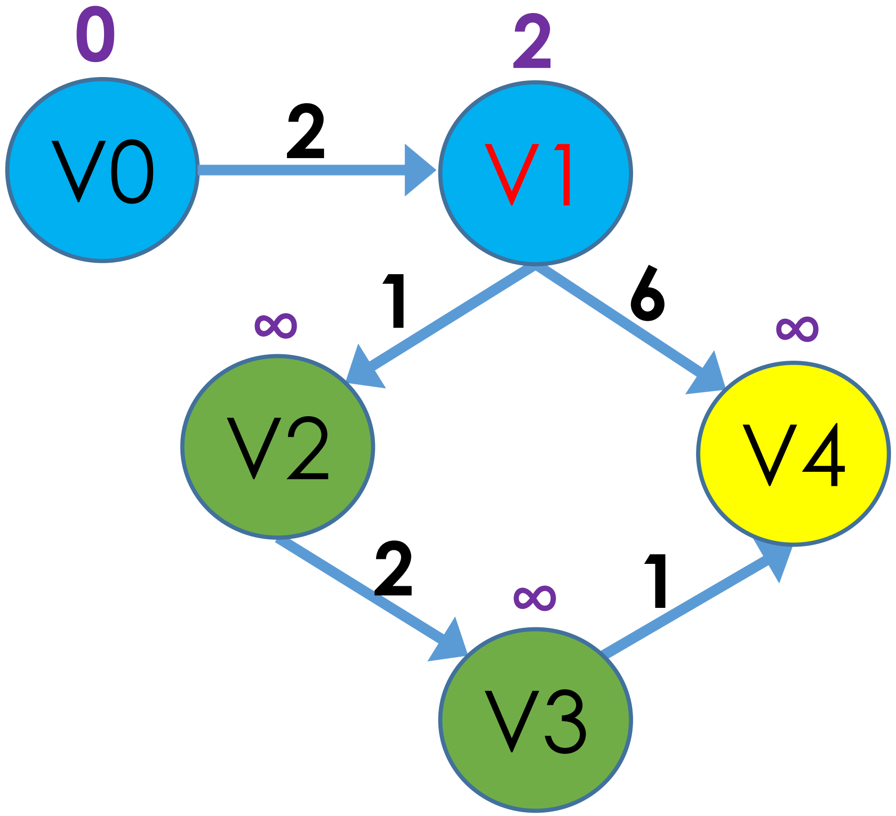
Dijkstra: Algorithm



```
Dijkstra(S, G):
  Initialize: Priority queue (PQ), visited HashSet,
              parent HashMap, and distances to infinity
  Enqueue {S, 0} onto the PQ
  while PQ is not empty:
    dequeue node curr from front of queue
    if(curr is not visited)
      add curr to visited set
      If curr == G return parent map
      for each of curr's neighbors, n, not in visited set:
        if path through curr to n is shorter
          update curr as n's parent in parent map
          enqueue {n, distance} into the PQ
  // If we get here then there's no path
```

PQ: ~~{V1, 2}~~
curr: V1
visited: V0

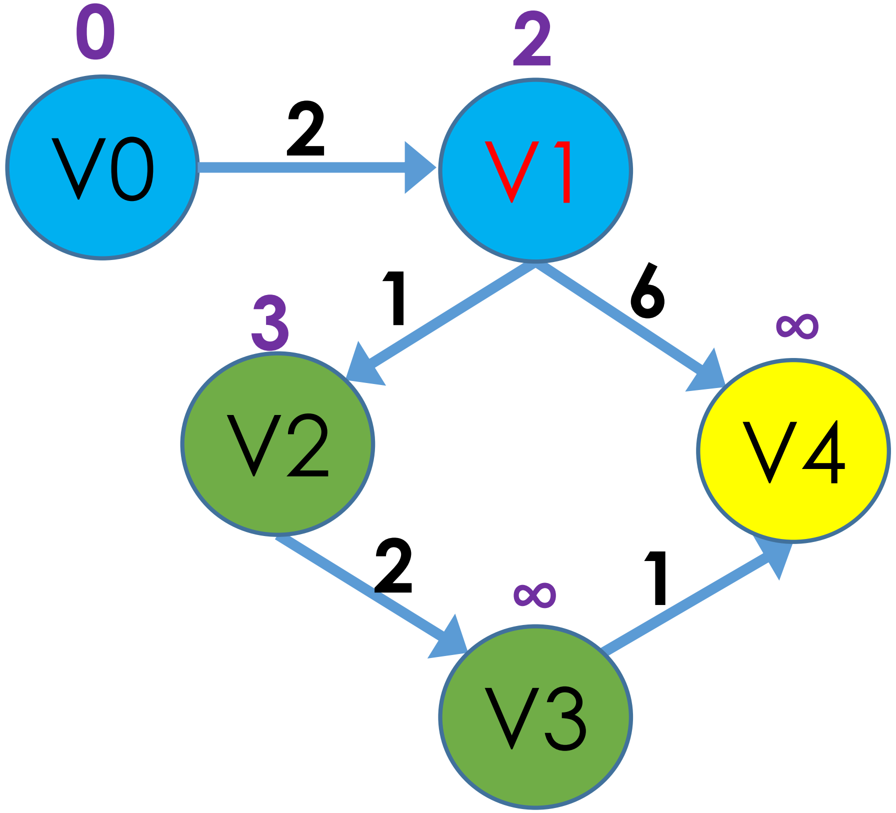
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ:
curr: V1
visited: V0, V1

Dijkstra: Algorithm



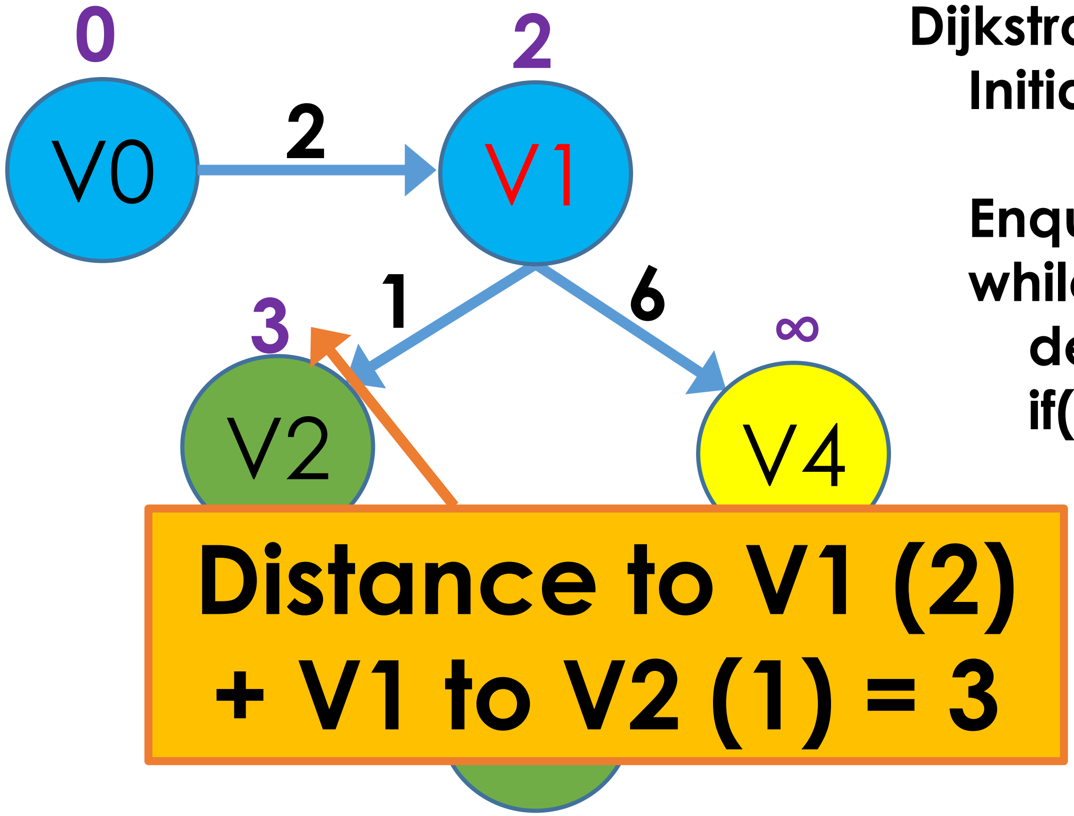
```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ:

curr: V1

visited: V0, V1

Dijkstra: Algorithm

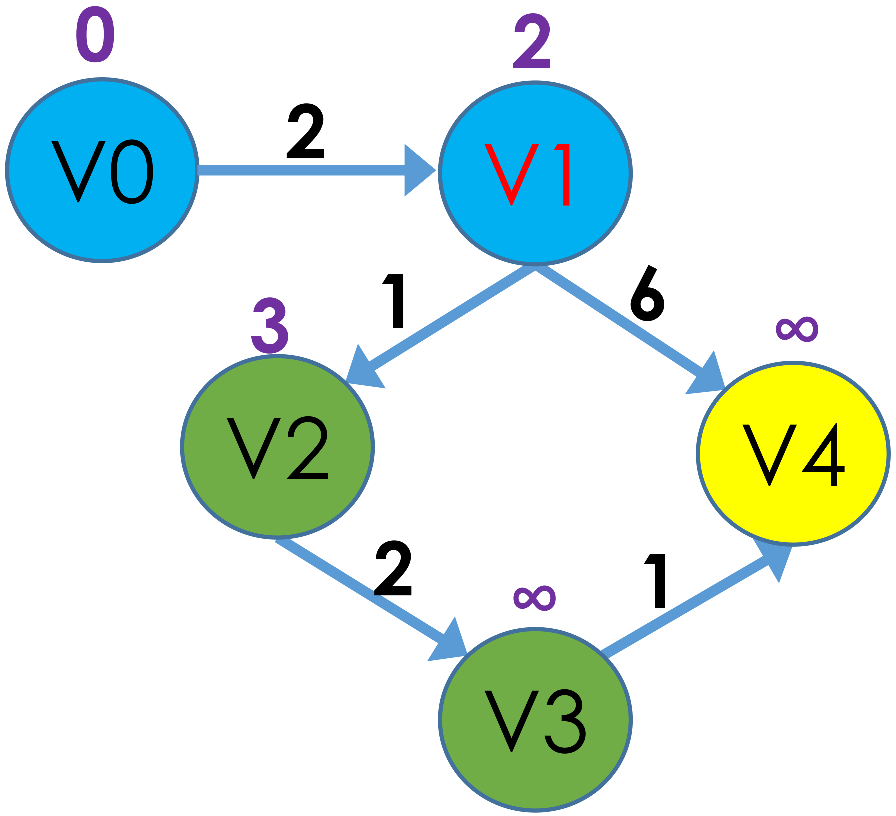


Distance to V1 (2) + V1 to V2 (1) = 3

```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ: {V2, 3}
curr: V1
visited: V0, V1

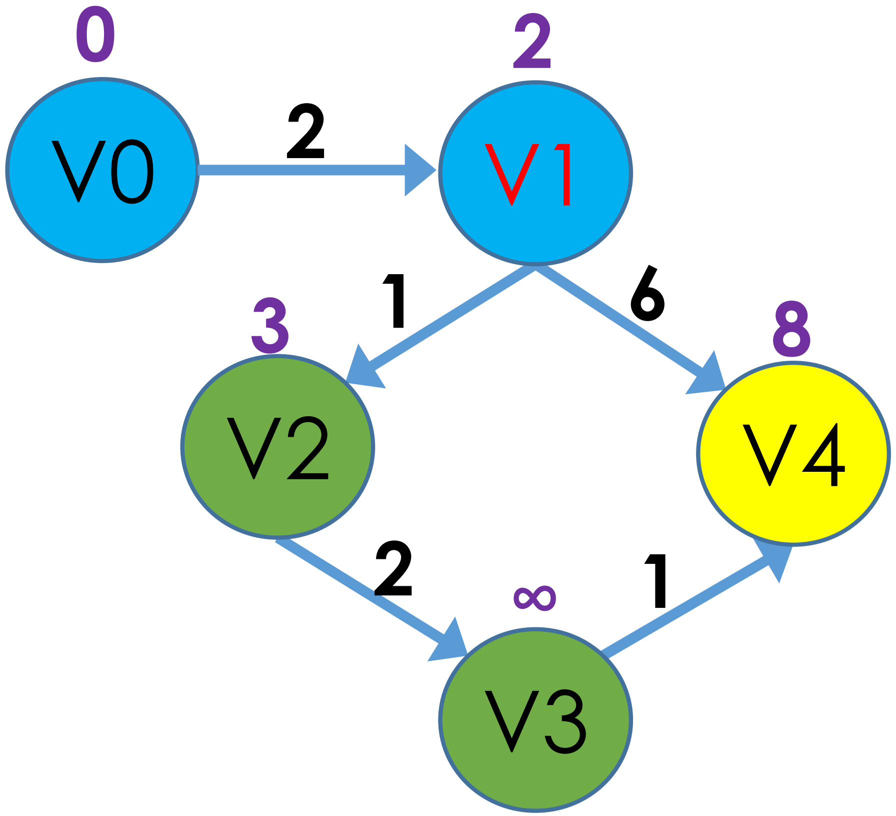
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ: {V2, 3}
curr: V1
visited: V0, V1

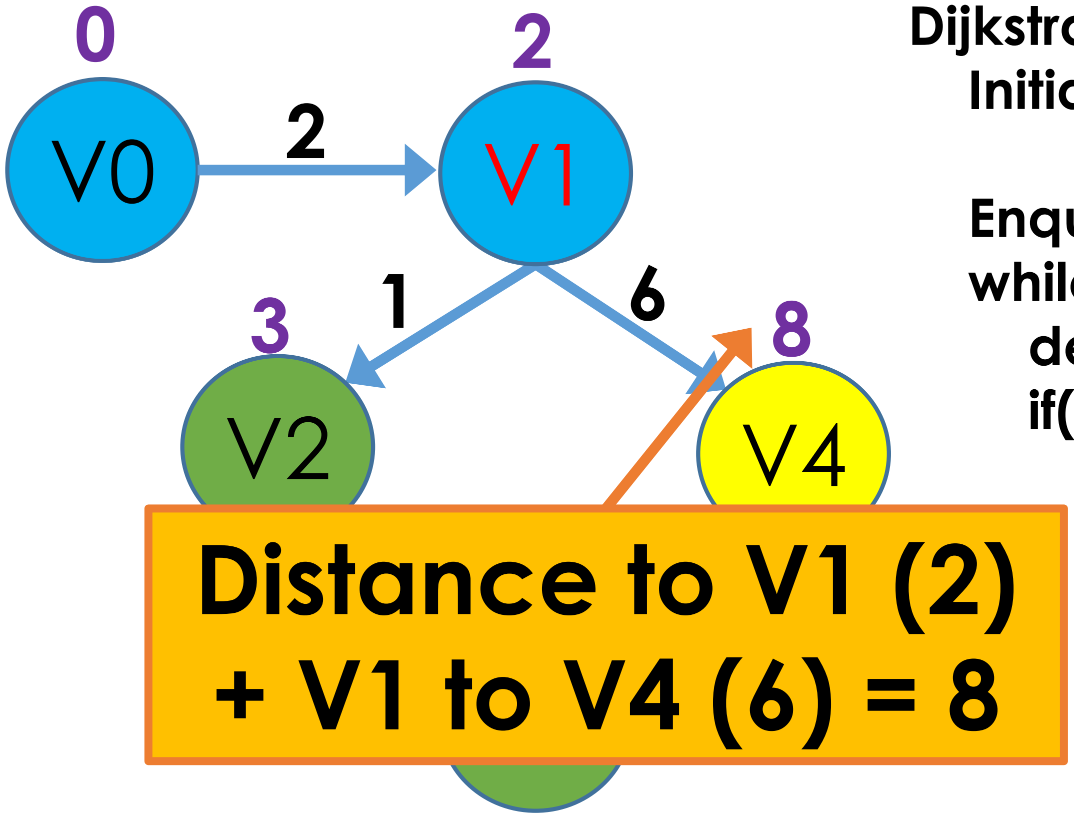
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ: {V2, 3}
curr: V1
visited: V0, V1

Dijkstra: Algorithm

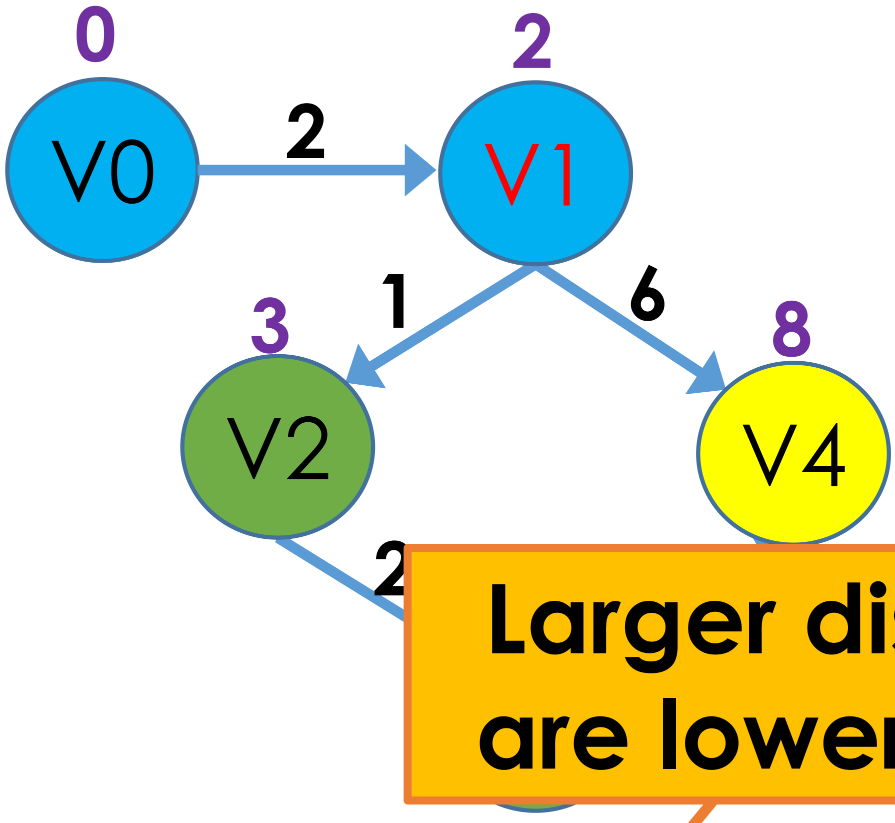


Distance to V1 (2) + V1 to V4 (6) = 8

```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ: {V2, 3}, {V4, 8}
curr: V1
visited: V0, V1

Dijkstra: Algorithm

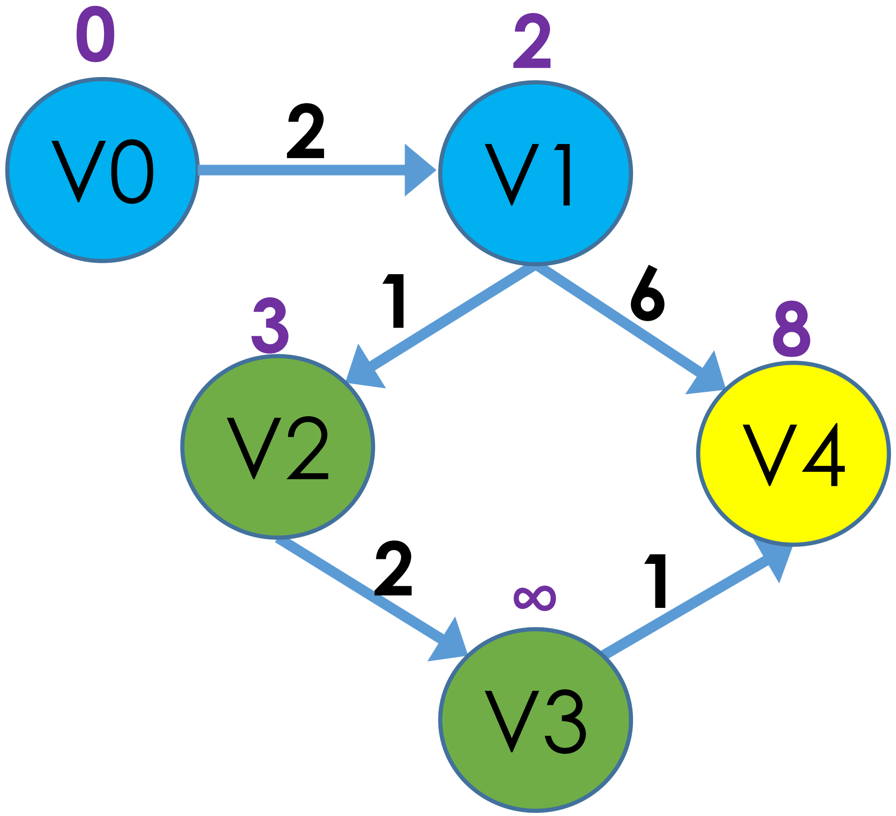


```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
                  of curr's neighbors, n, not in visited set:
                   through curr to n is shorter
                  ate curr as n's parent in parent map
                  enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

Larger distances are lower priority

PQ: {V2, 3}, {V4, 8}
curr: V1
visited: V0, V1

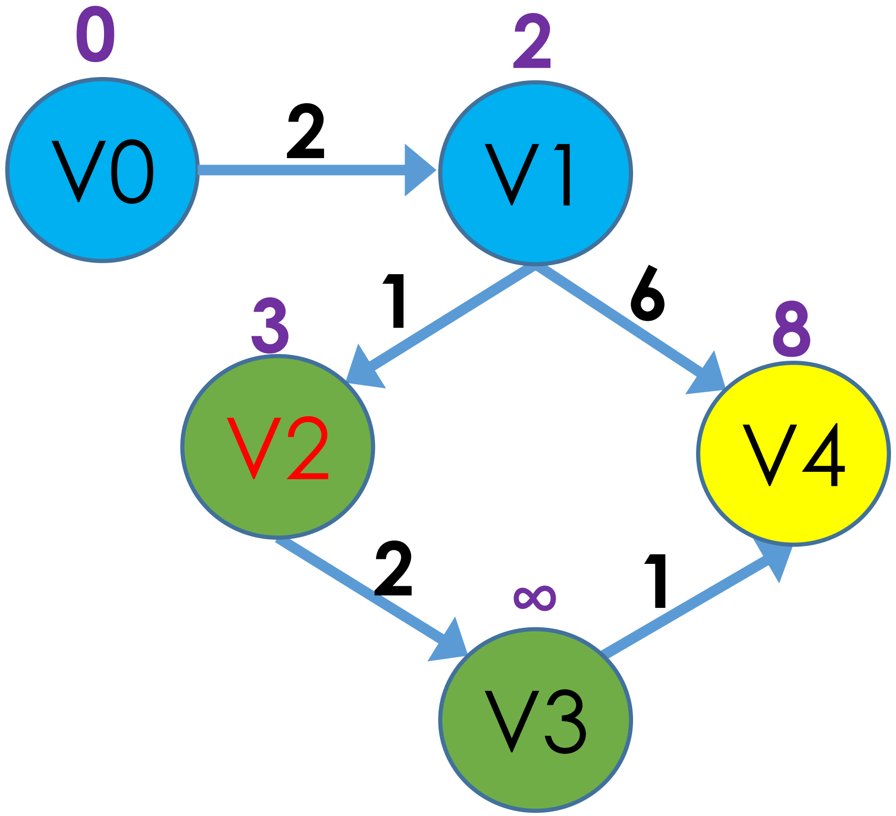
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ: {V2, 3}, {V4, 8}
curr: V1
visited: V0, V1

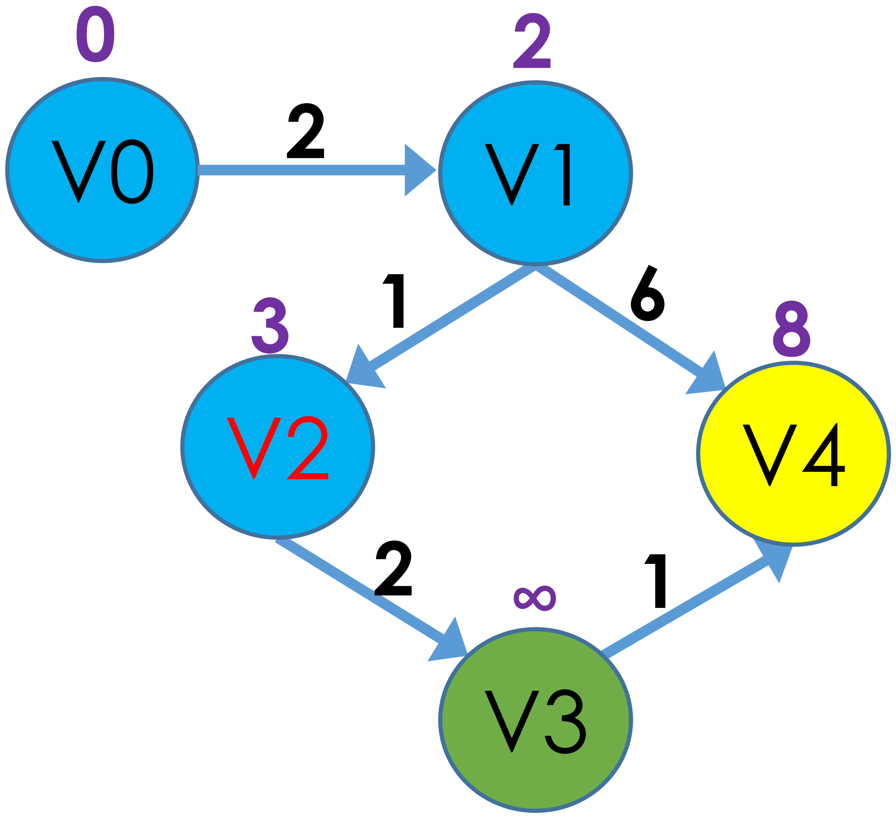
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ: ~~{V2, 3}~~, {V4, 8}
curr: V2
visited: V0, V1

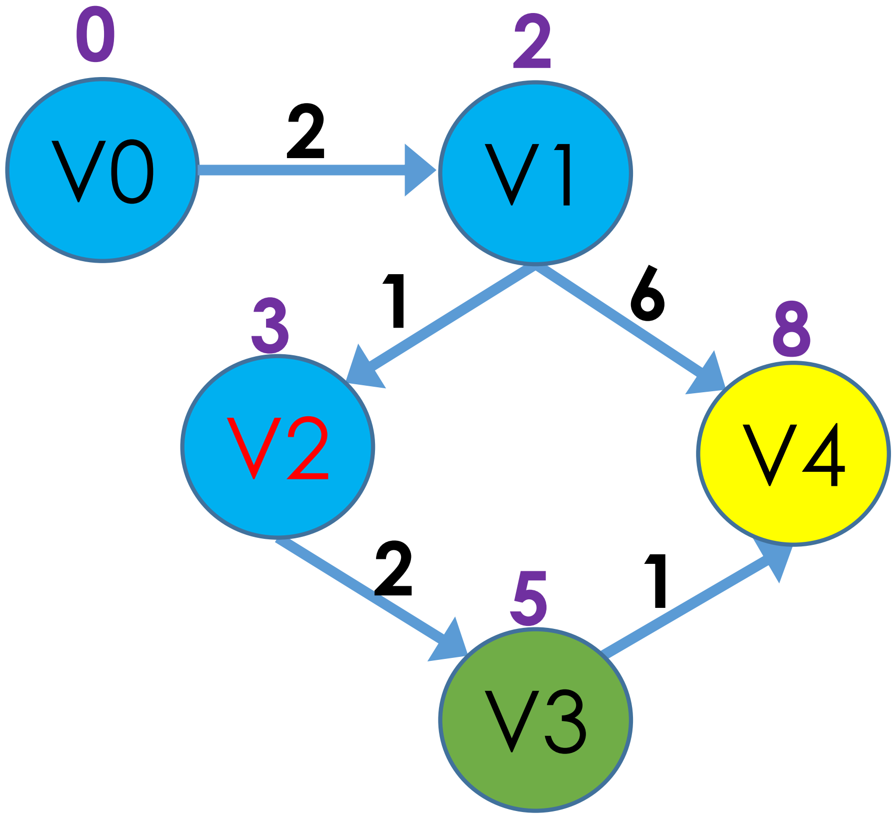
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ: {V4, 8}
curr: V2
visited: V0, V1, V2

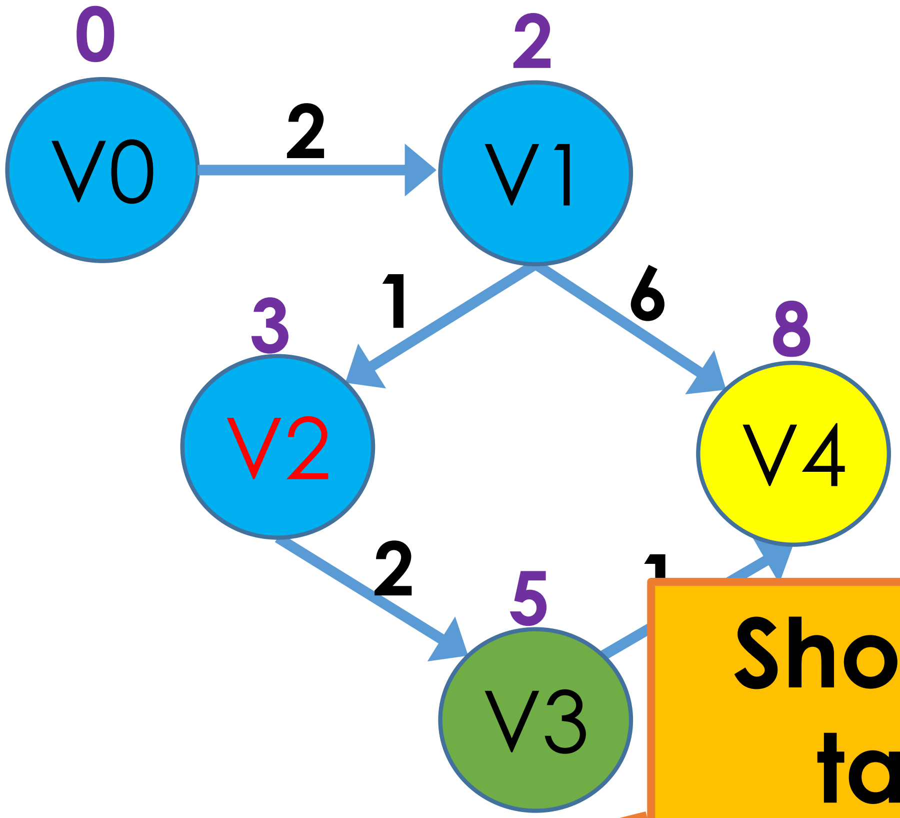
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ: {V4, 8}
curr: V2
visited: V0, V1, V2

Dijkstra: Algorithm

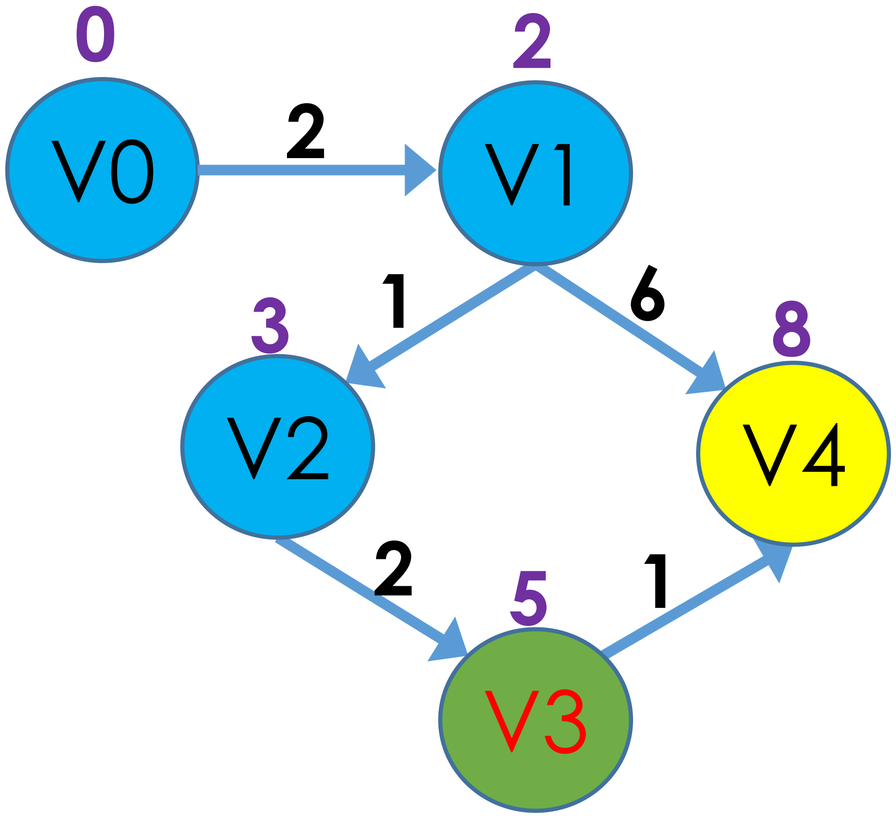


```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            eighbors, n, not in visited set:
            curr to n is shorter
            s n's parent in parent map
            distance} into the PQ
    // If we get here then there's no path
```

Shorter Distance takes Priority

PQ: {V3, 5}, {V4, 8}
curr: V2
visited: V0, V1, V2

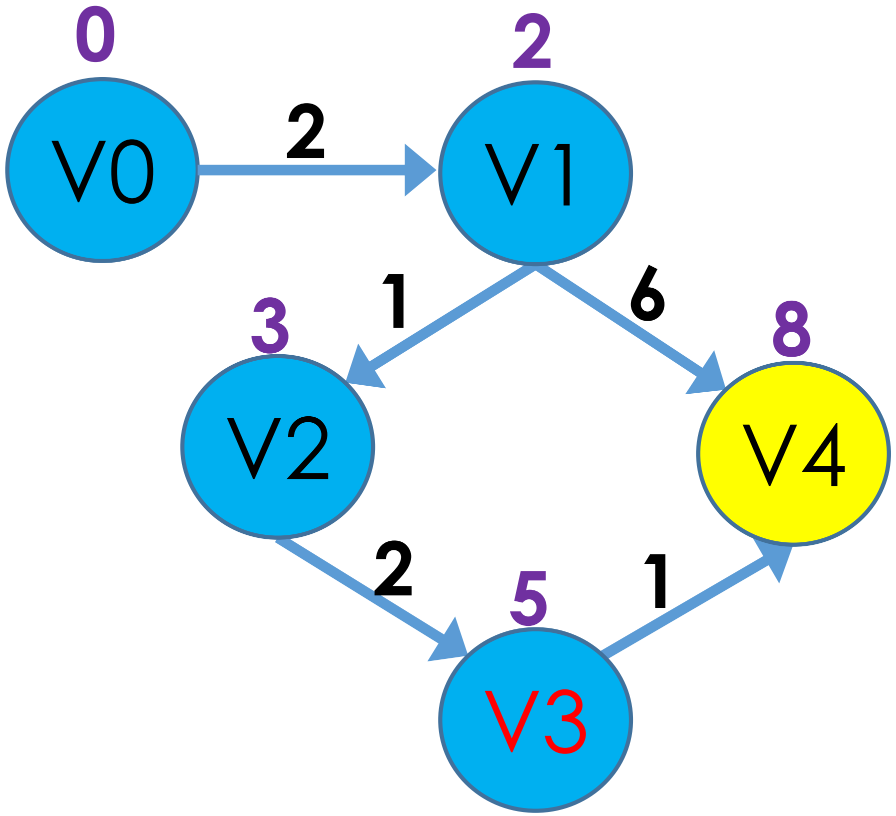
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ: ~~{V3, 5}~~, {V4, 8}
curr: V3
visited: V0, V1, V2

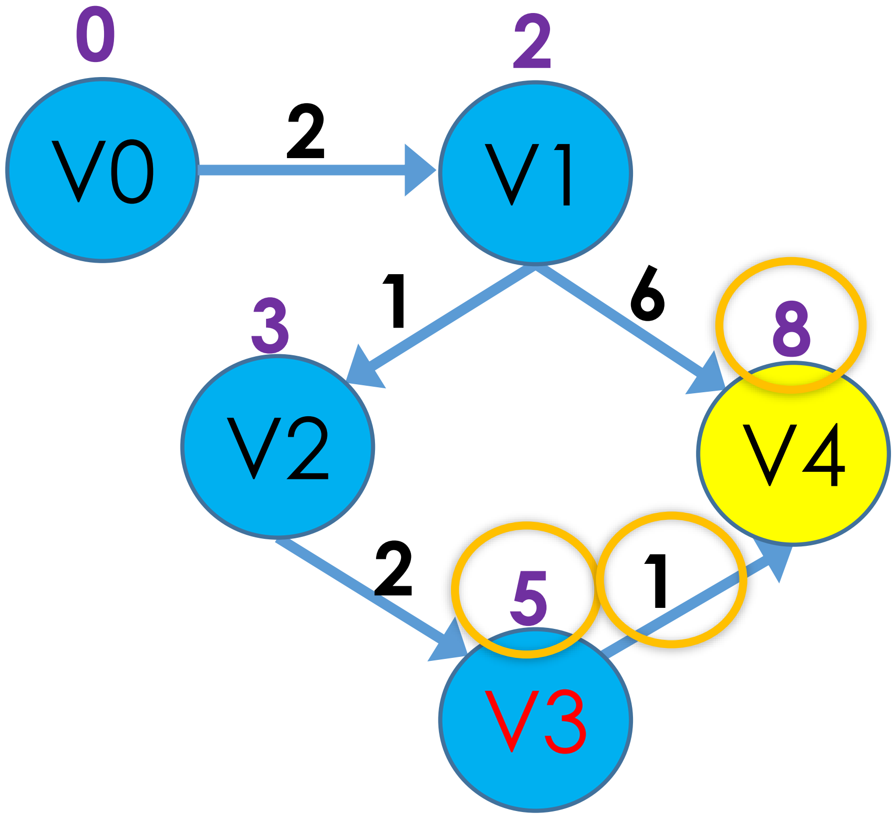
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ: {V4, 8}
curr: V3
visited: V0, V1, V2, V3

Dijkstra: Algorithm

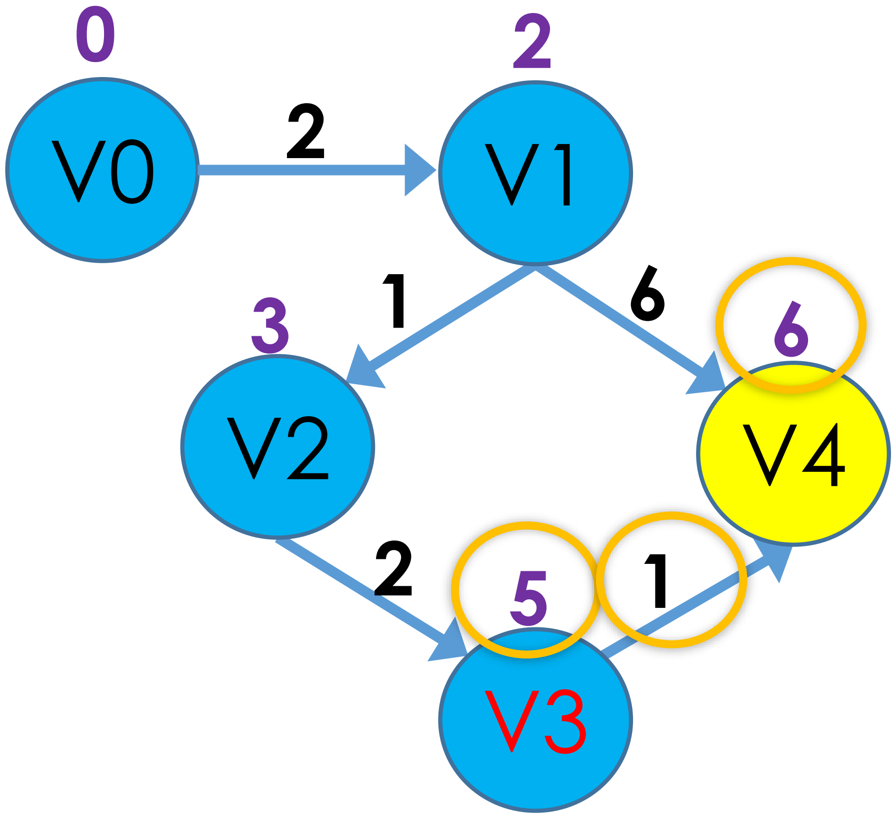


```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
                                                    n visited set:

                                                    urent map
                                                    PQ
    // If we get here then there's no path
```

5+1 is shorter than 8!

PQ: {V4, 8}
curr: V3
visited: V0, V1, V2, V3

Dijkstra: Algorithm

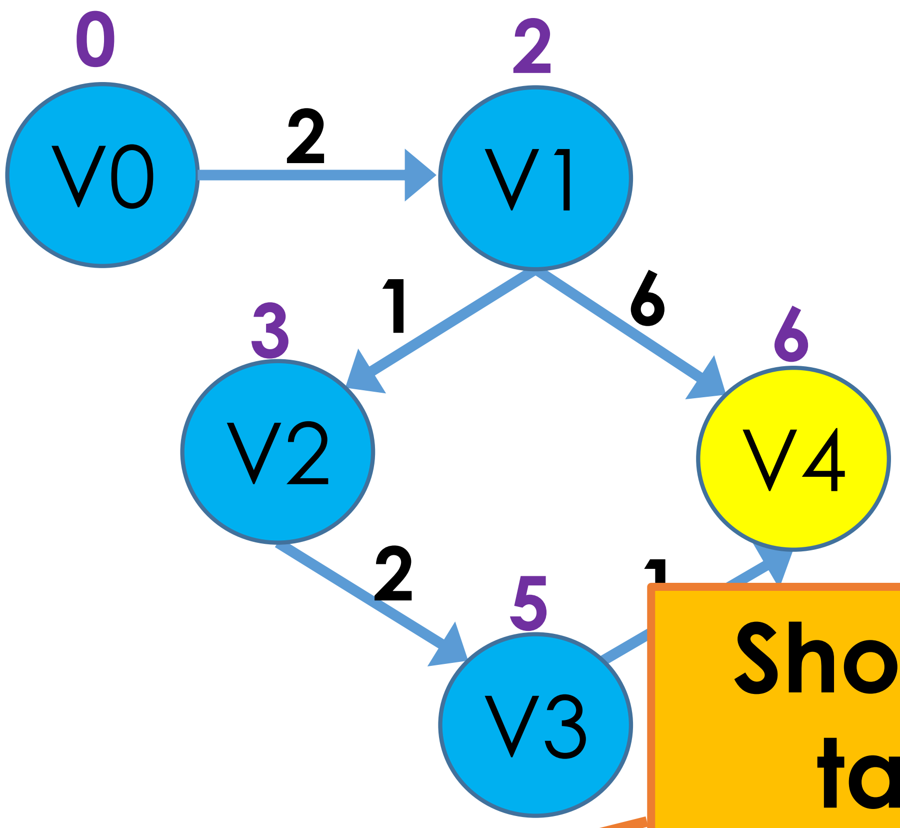


```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
                                                   n visited set:

                                                   rent map
                                                   PQ
    // If we get here then there's no path
```

5+1 is shorter than 8!

PQ: {V4, 8}
curr: V3
visited: V0, V1, V2, V3

Dijkstra: Algorithm

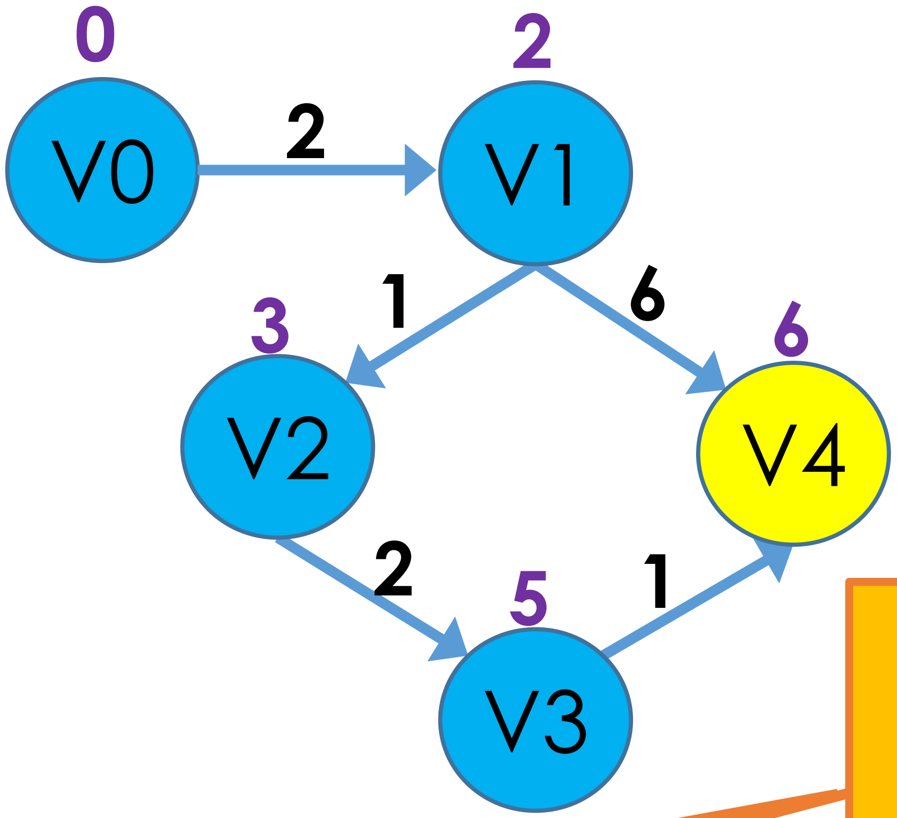


```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
                                eighbors, n, not in visited set:
                                curr to n is shorter
                                s n's parent in parent map
                                distance} into the PQ
    // If we get here then there's no path
```

Shorter Distance takes Priority

PQ: {V4, 6}, {V4, 8}
curr: V3
visited: V0, V1, V2, V3

Dijkstra: Algorithm

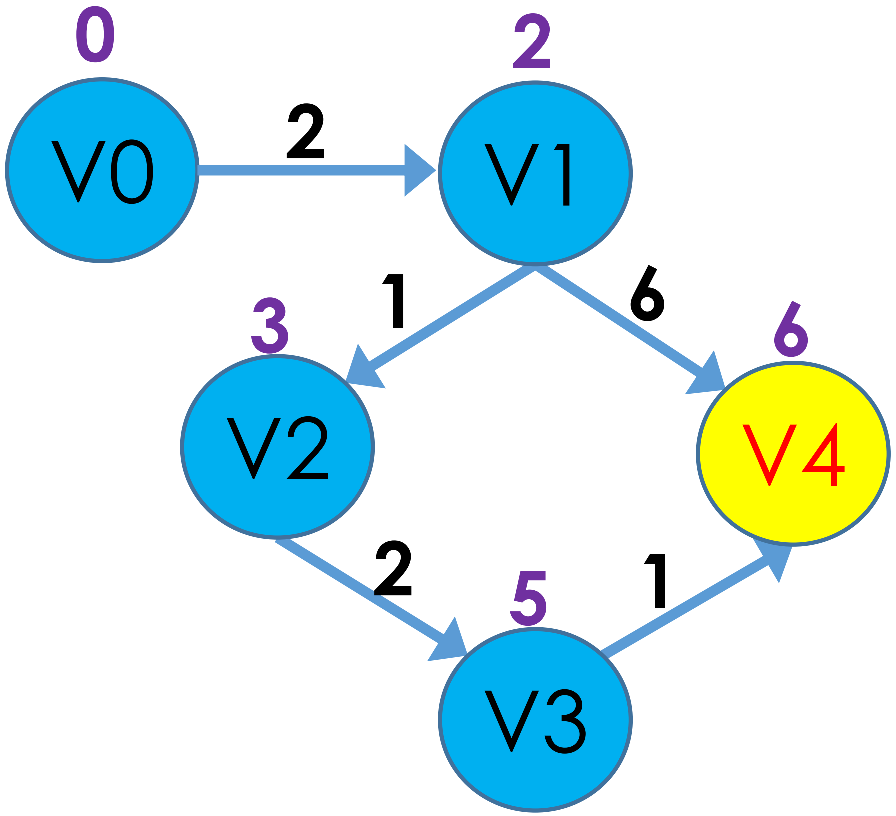


```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
                                                            set:

                                                            p

    // If we get here then there's no path
```

Two entries for V4 in the queue? That's okay!

PQ: {V4, 6}, {V4, 8}
curr: V3
visited: V0, V1, V2, V3

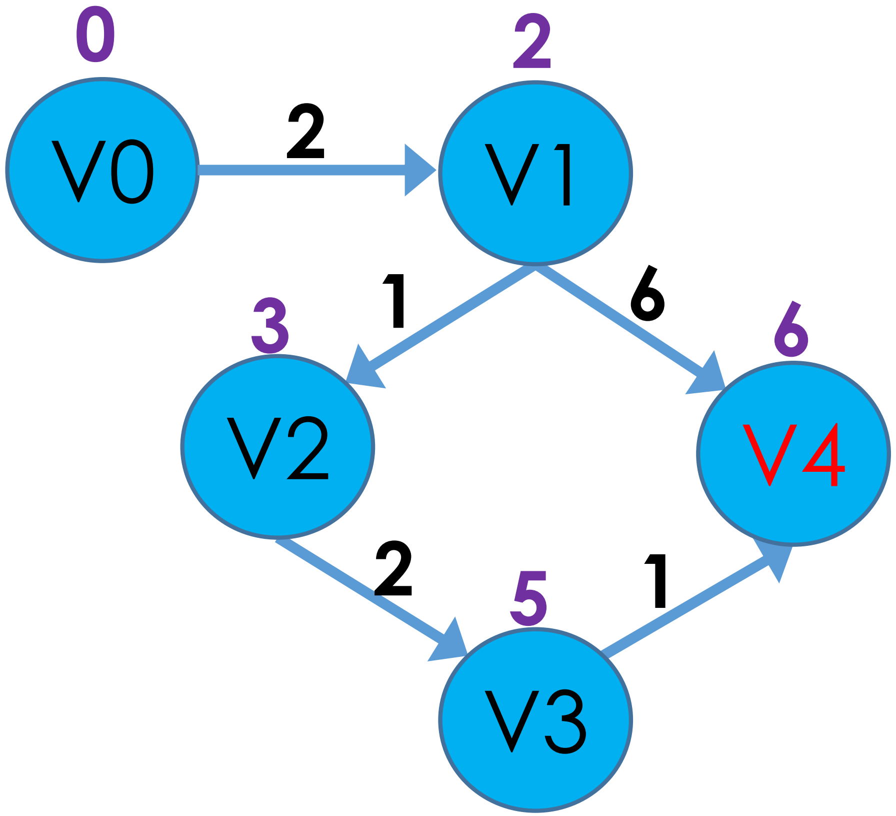
Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ: ~~{V4, 6}~~, {V4, 8}
curr: V4
visited: V0, V1, V2, V3

Dijkstra: Algorithm



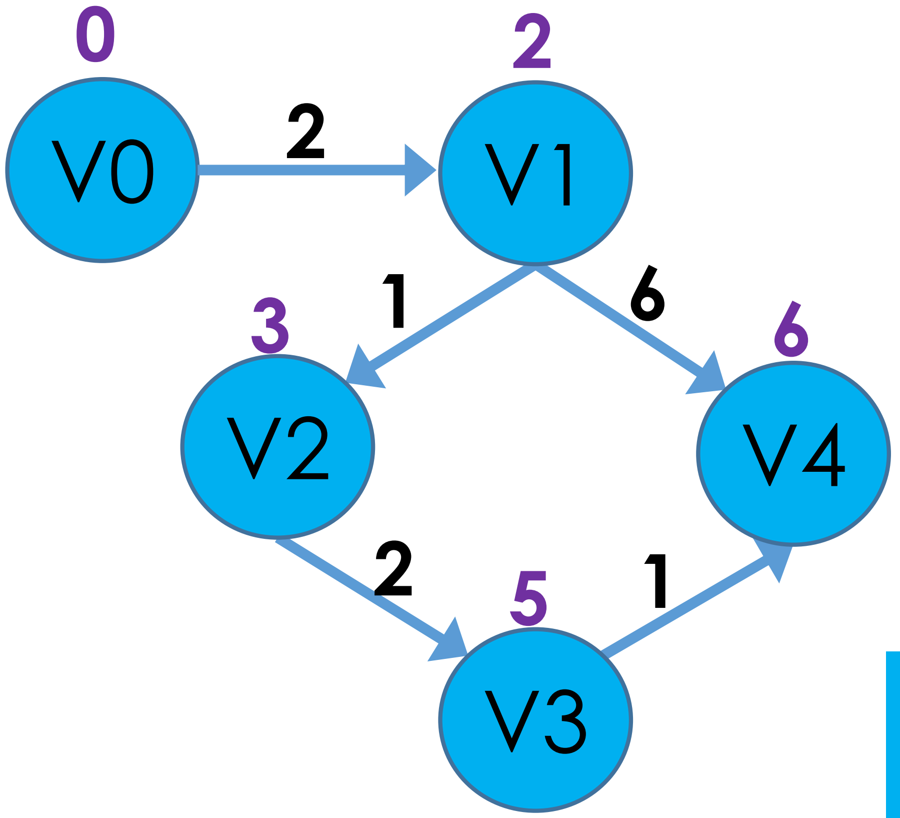
```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    enqueue {n, distance} into the PQ
    // If we get here then there's no path
```

PQ: ~~{V4, 6}~~, {V4, 8}

curr: V4

visited: V0, V1, V2, V3, V4

Dijkstra: Algorithm

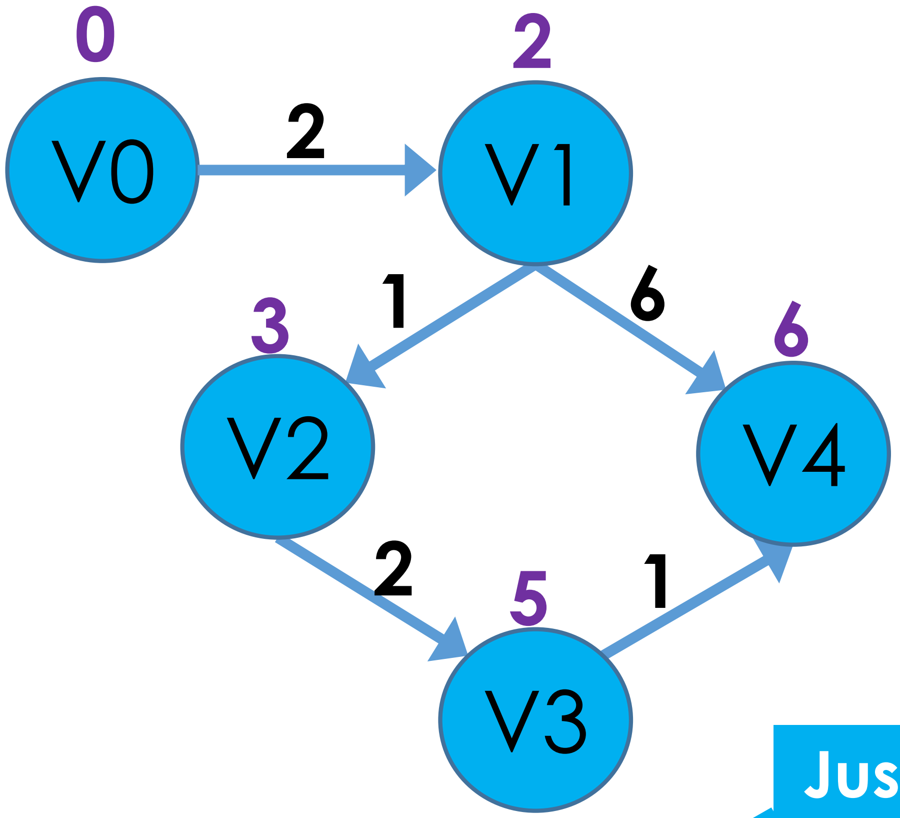


```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                                                    map
```

Done!
Shortest Path to V4 has a distance of 6.

PQ: ~~{V4, 6}~~, {V4, 8}
curr: V4
visited: V0, V1, V2, V3, V4

Dijkstra: Algorithm

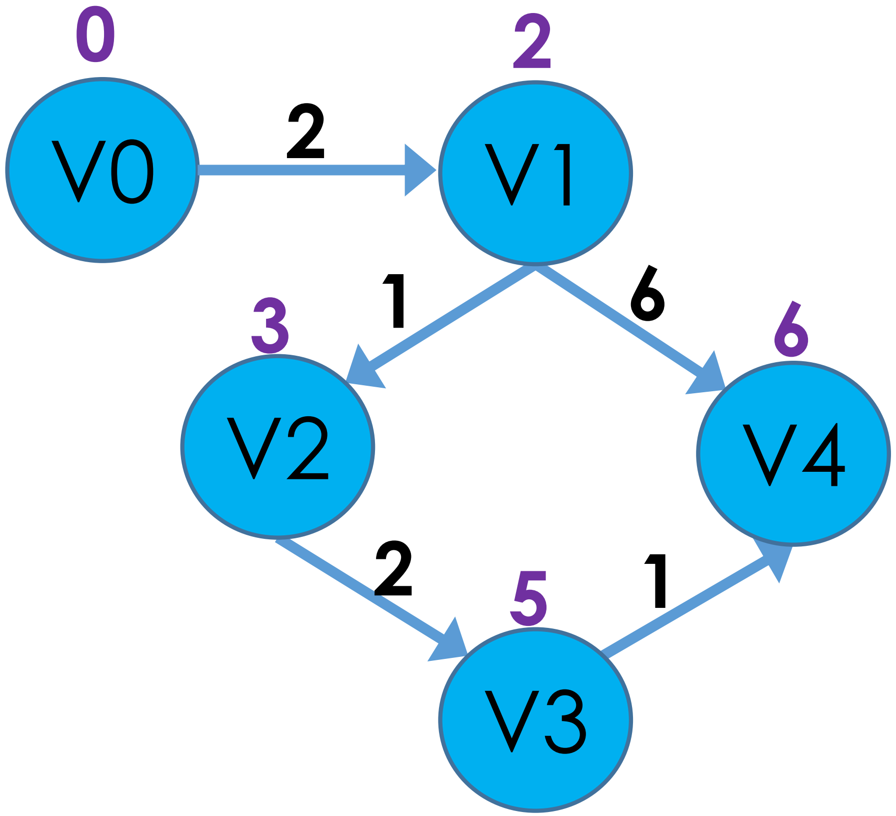


```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
            for each of curr's neighbors, n, not in visited set:
                if path through curr to n is shorter
                    update curr as n's parent in parent map
                    PQ
```

Just like BFS, everything in the queue is "longer" than what we've seen already

PQ: ~~{V4, 6}~~, {V4, 8}
curr: V4
visited: V0, V1, V2, V3, V4

Dijkstra: Algorithm



```
Dijkstra(S, G):
    Initialize: Priority queue (PQ), visited HashSet,
                parent HashMap, and distances to infinity
    Enqueue {S, 0} onto the PQ
    while PQ is not empty:
        dequeue node curr from front of queue
        if(curr is not visited)
            add curr to visited set
            If curr == G return parent map
```

I encourage you to think through how to maintain the "parent" HashMap on your own!

PQ: ~~{V4, 6}~~, {V4, 8}
curr: V4
visited: V0, V1, V2, V3, V4