Creating a Road Network Graph

Course 4, Module 3, Lesson 1

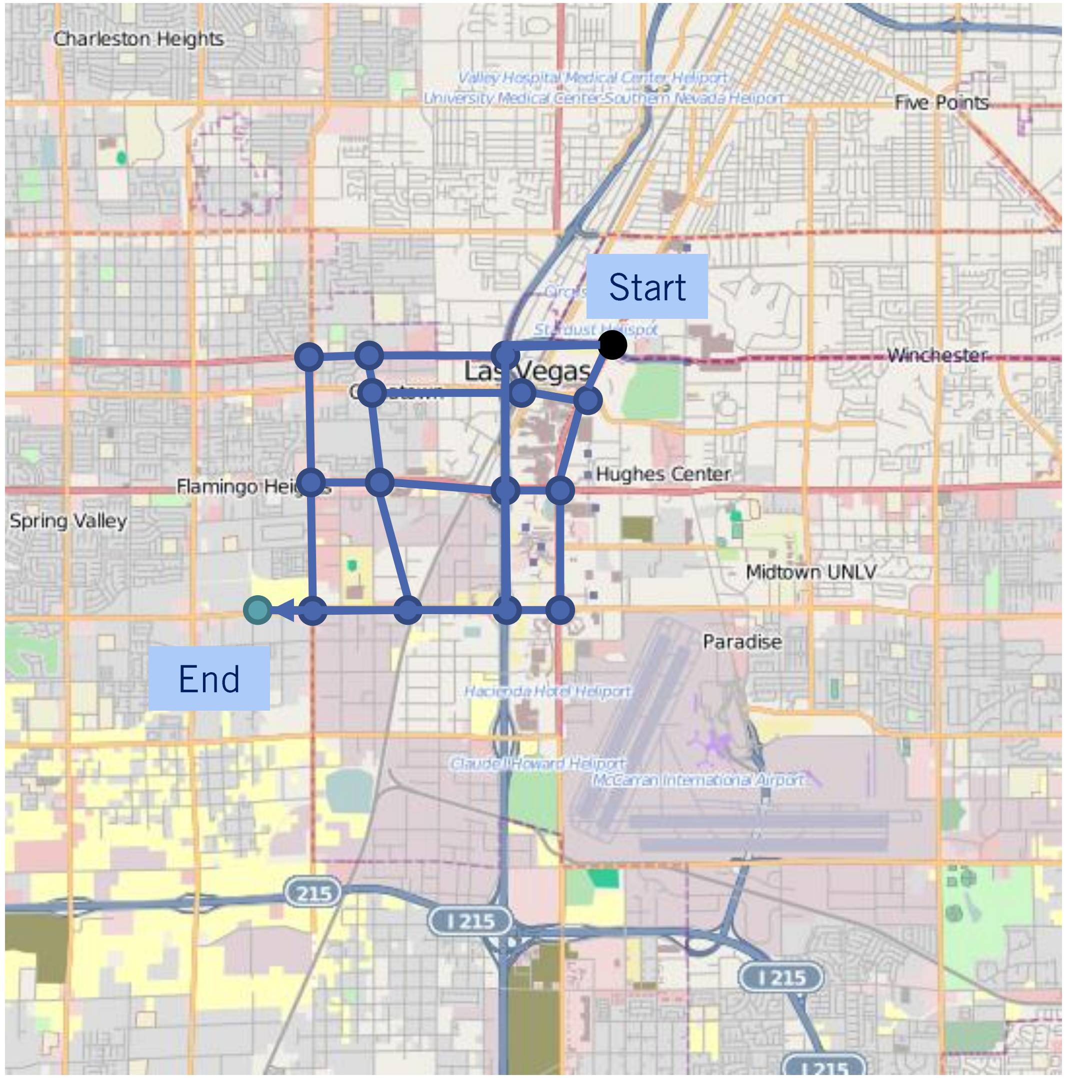
UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE & ENGINEERING

Mission Planning

Highest level planning problem.

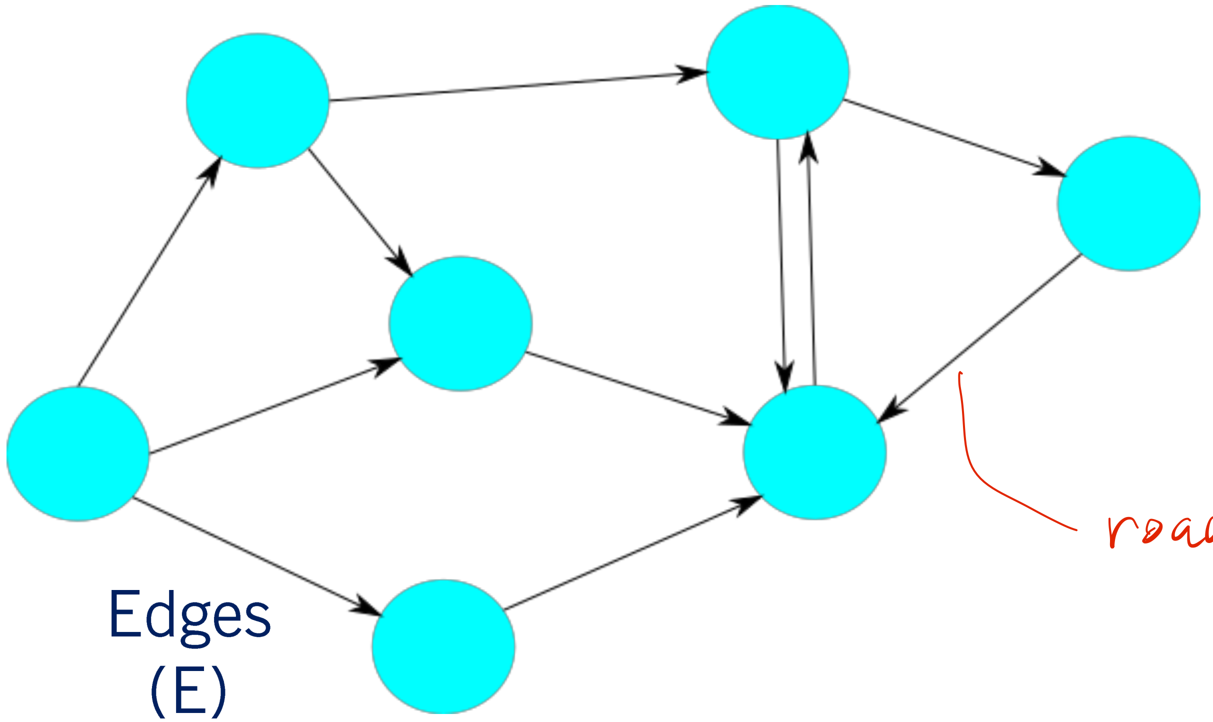
abstracting away lower-level details, like rules of the road and other agents present

- Order of km.
- Focus on speed limit, road length, traffic flow rate, & road closures



Graphs

Graph: $G = (V, E)$



Vertices
(V)

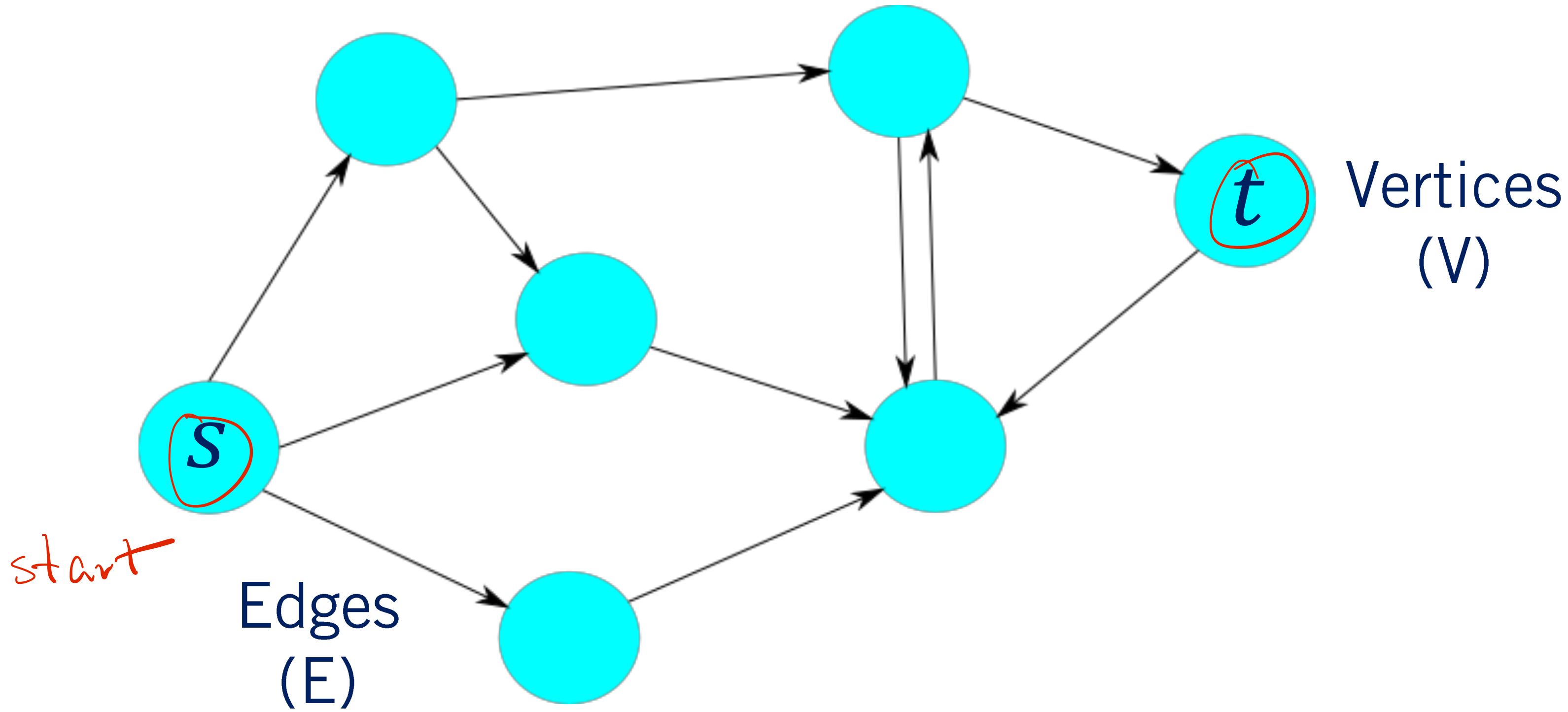
point on the road network

road segment

Edges
(E)

Graphs

Graph: $G = (V, E)$



Breadth First Search (BFS)

Algorithm BFS(G,s,t)

```
open ← Queue()    First in First out
closed ← Set()
predecessors ← Dict()
open.enqueue(s)
while !open.isEmpty() do
    u ← open.dequeue()
    if isGoal(u) then
        return extractPath(u, predecessors)
    for all v ∈ u.successors()
        if v ∈ closed or v ∈ open then
            continue
        open.enqueue(v)
        predecessors[v] ← u
    closed.add(u)
```

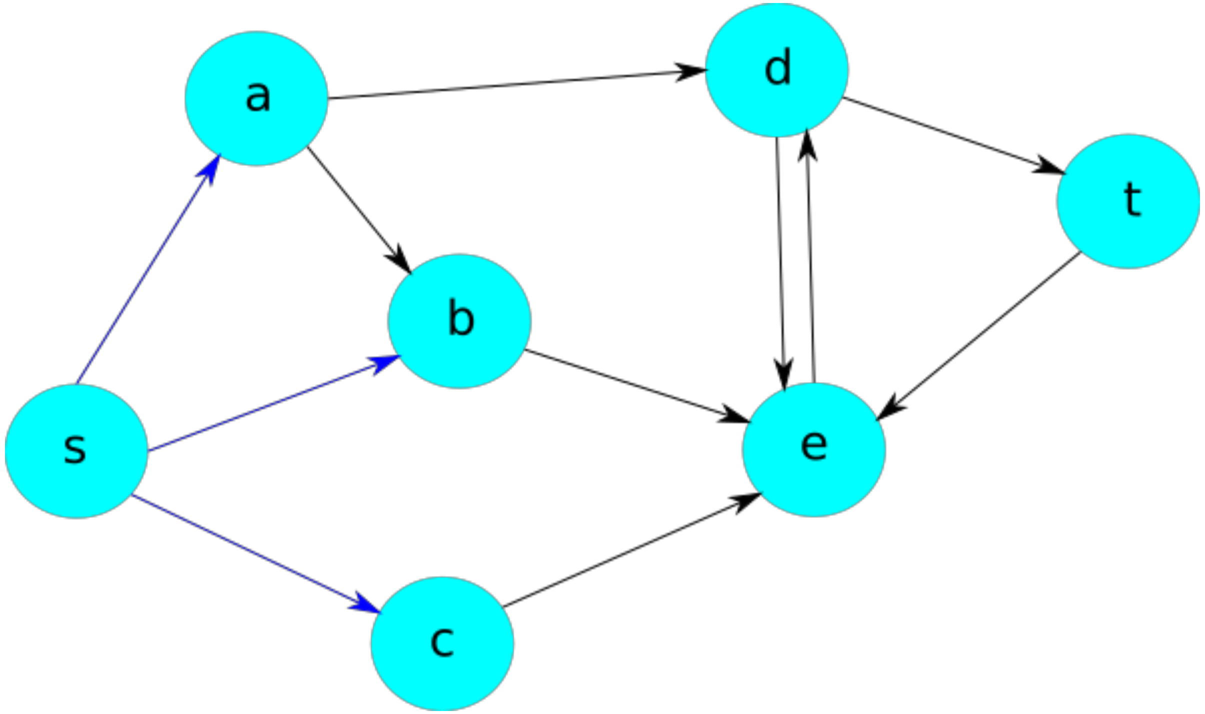
Example - First Wavefront

Open Queue:

a
b
c

Closed Set: s

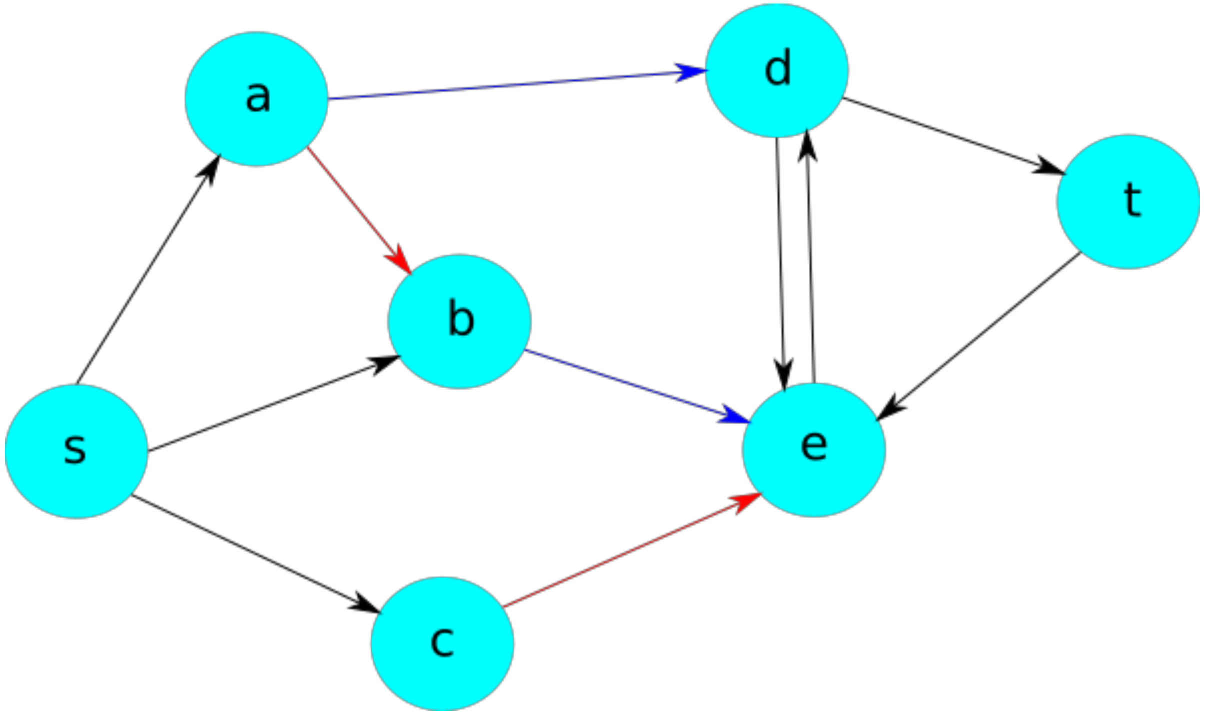
Predecessors: S



Example - Second Wavefront

Open Queue:
d
e

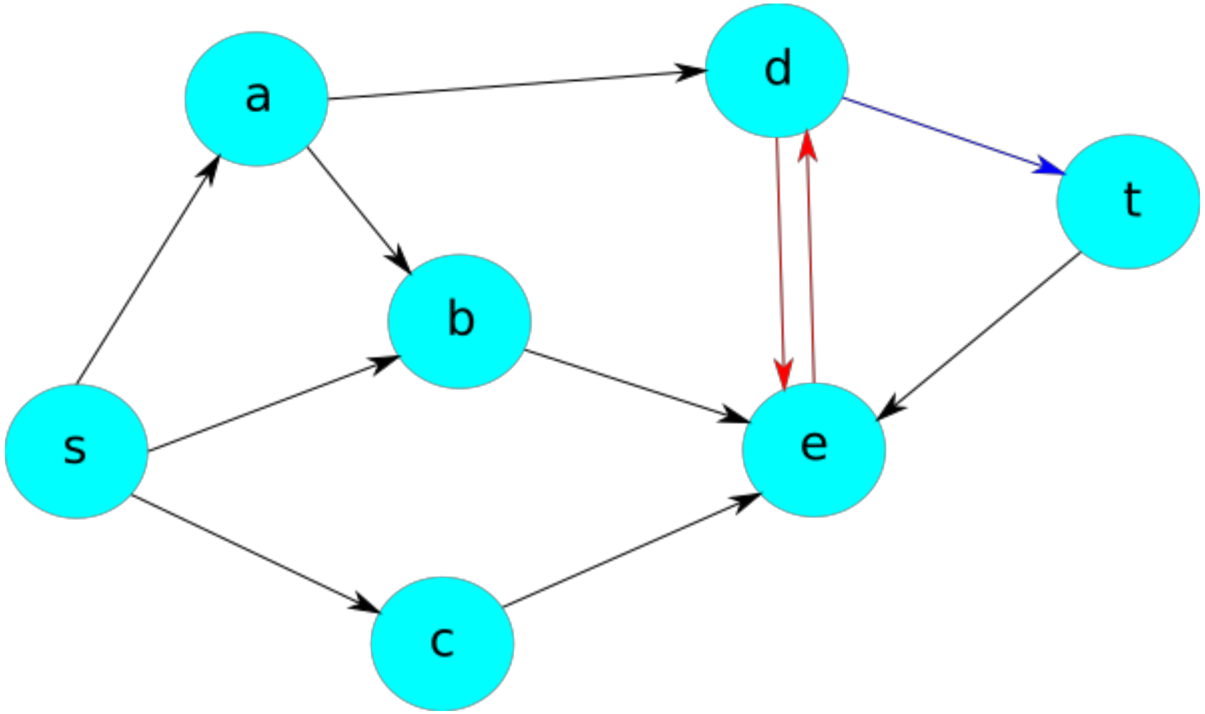
Closed Set: s
a
b
c



Example - Third Wavefront

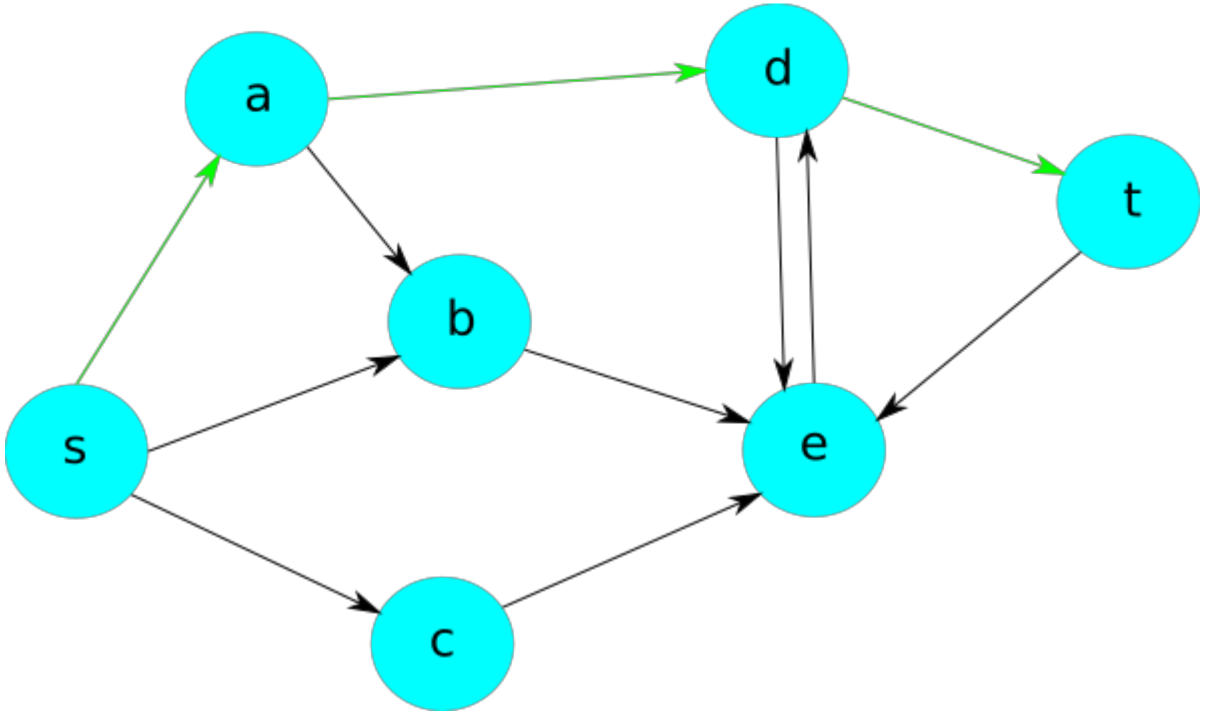
Open Queue:
t

Closed Set: s
a
b
c
d
e



Example - Optimal Path

Final Path: s
a
d
t



Summary

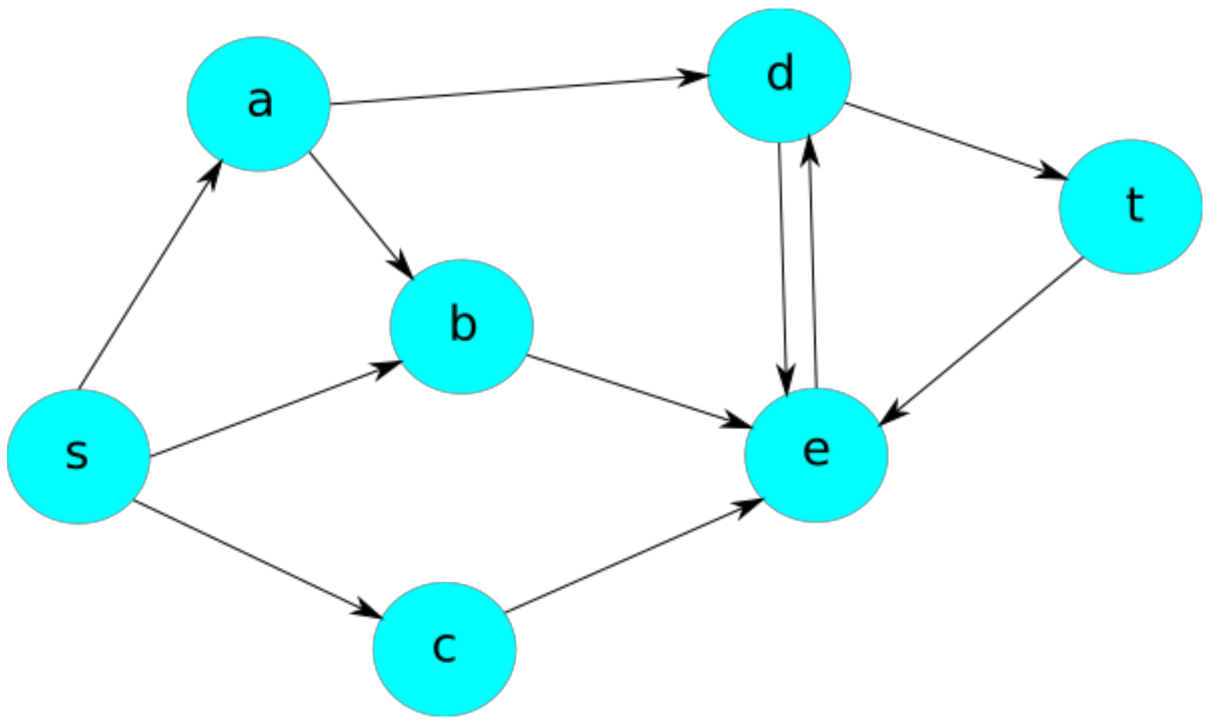
- Recognize the mission planning problem as a map-level navigation problem
- Learned how to embed a graph in the map
  - Vertices connected by road segments, which correspond to edges
- Learned how to use BFS to search an unweighted graph for the shortest path to the destination

Dijkstra’s Shortest Path Search

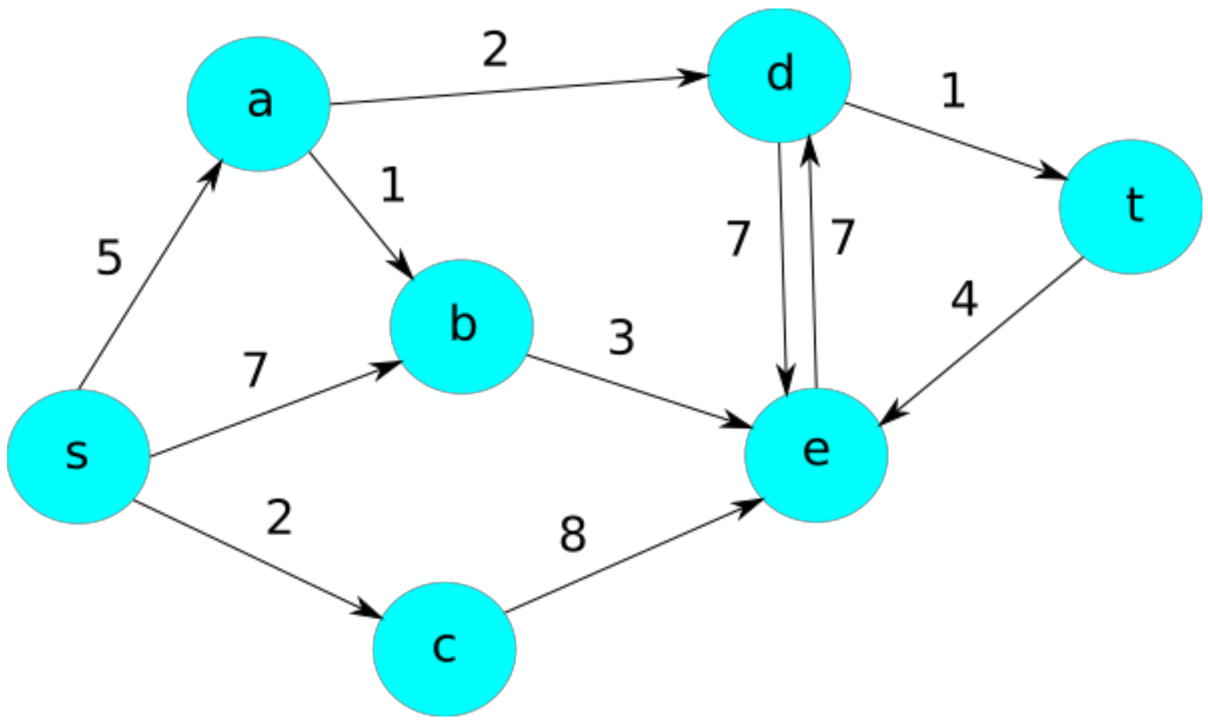
Course 4, Module 3, Lesson 2

UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE & ENGINEERING

Unweighted Graph



Weighted Graph



Dijkstra's Algorithm

Algorithm Dijkstra's(G,s,t)

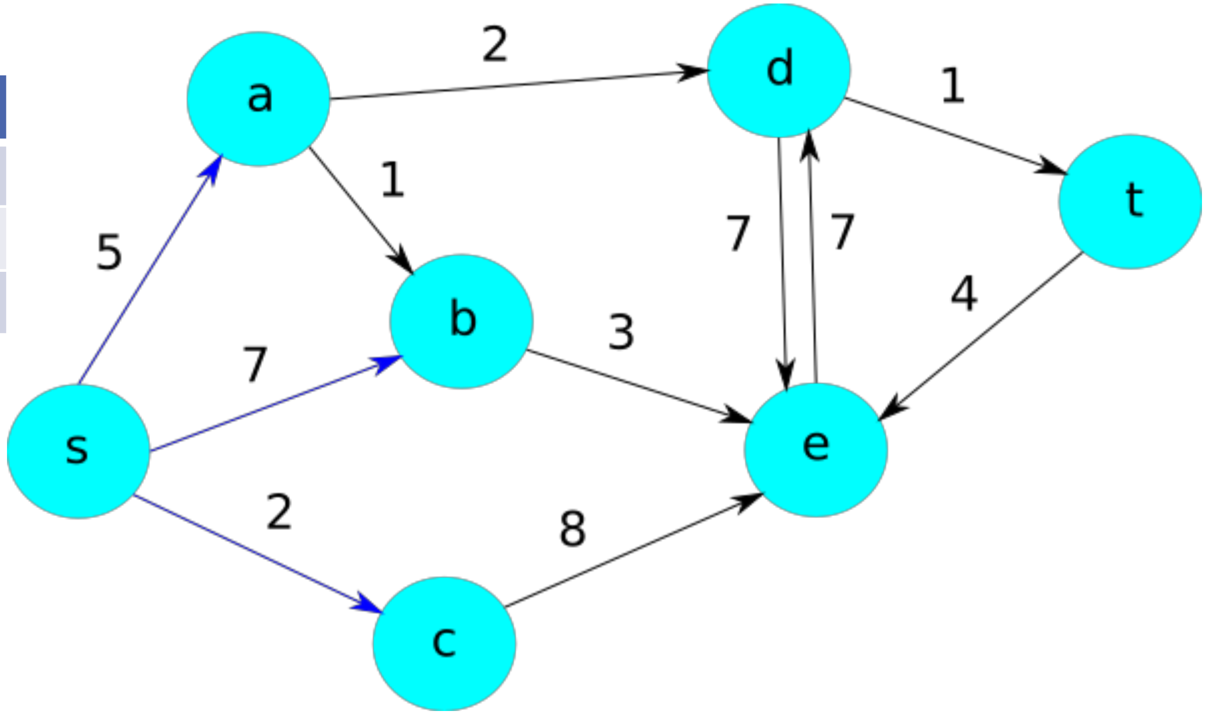
```
open ← MinHeap()
closed ← Set()
predecessors ← Dict()
open.push(s, 0)
while !open.isEmpty() do
  u, uCost ← open.pop()
  if isGoal(u) then
    return extractPath(u, predecessors)
  for all v ∈ u.successors()
    if v ∈ closed then
      continue
    uvCost ← edgeCost(G, u, v)
    if v ∈ open then
      if uCost + uvCost < open[v] then
        open[v] ← uCost + uvCost
        predecessors[v] ← u
    else
      open.push(v, uCost + uvCost)
      predecessors[v] ← u
  closed.add(u)
```

Example - Processing s

Open Min Heap:

Node	Cost to vertex
c	2
a	5
b	7

Closed Set: s



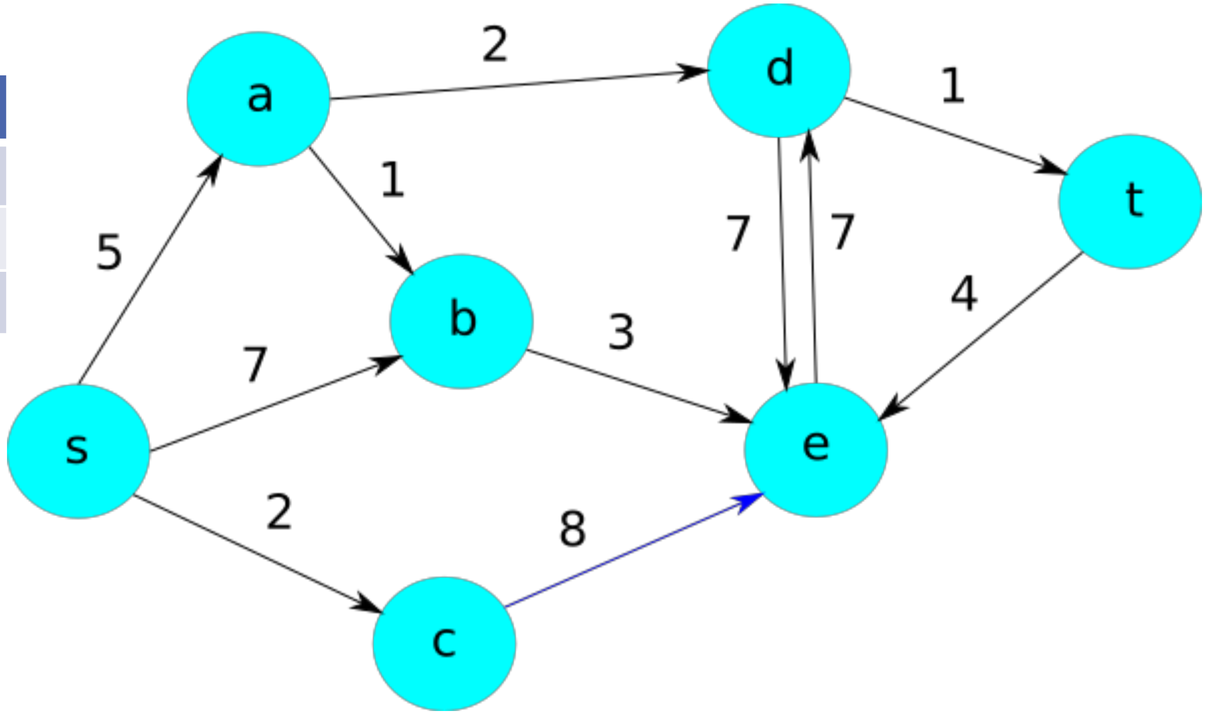
Example - Processing c

Open Min Heap:

Node	Cost to vertex
a	5
b	7
e	10

Closed Set: s

c

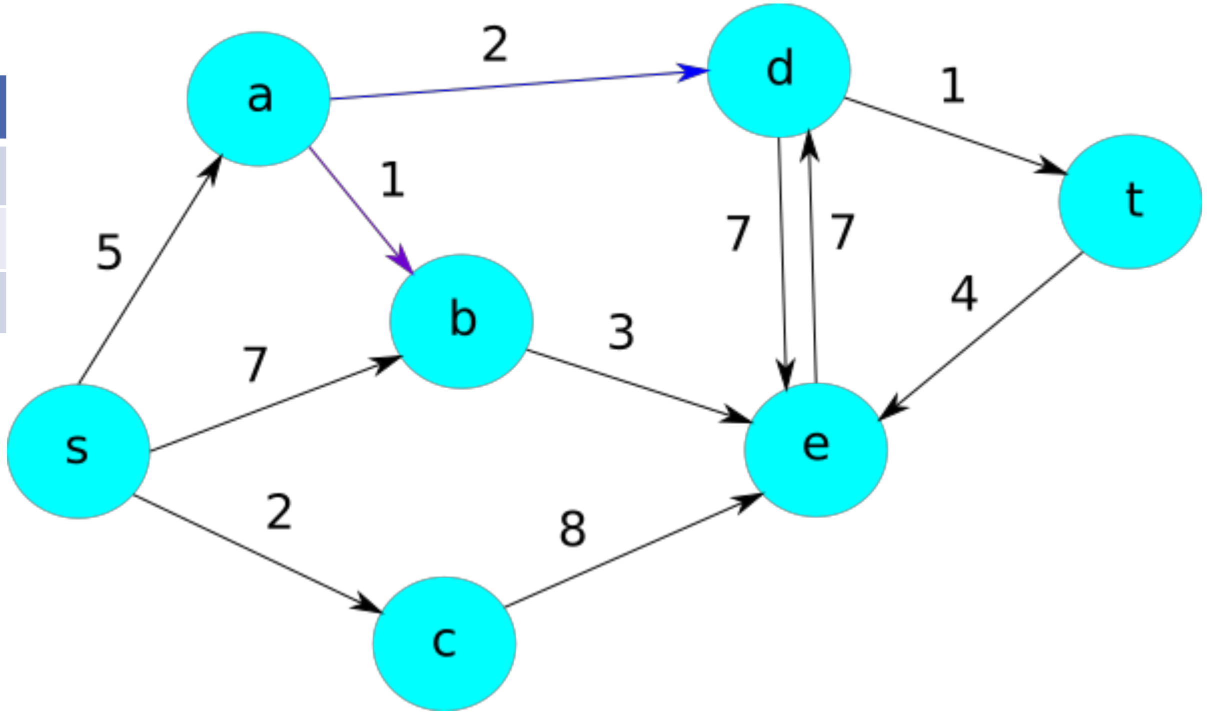


Example - Processing a

Open Min Heap:

Node	Cost to go
b	6
d	7
e	10

Closed Set: s
c
a

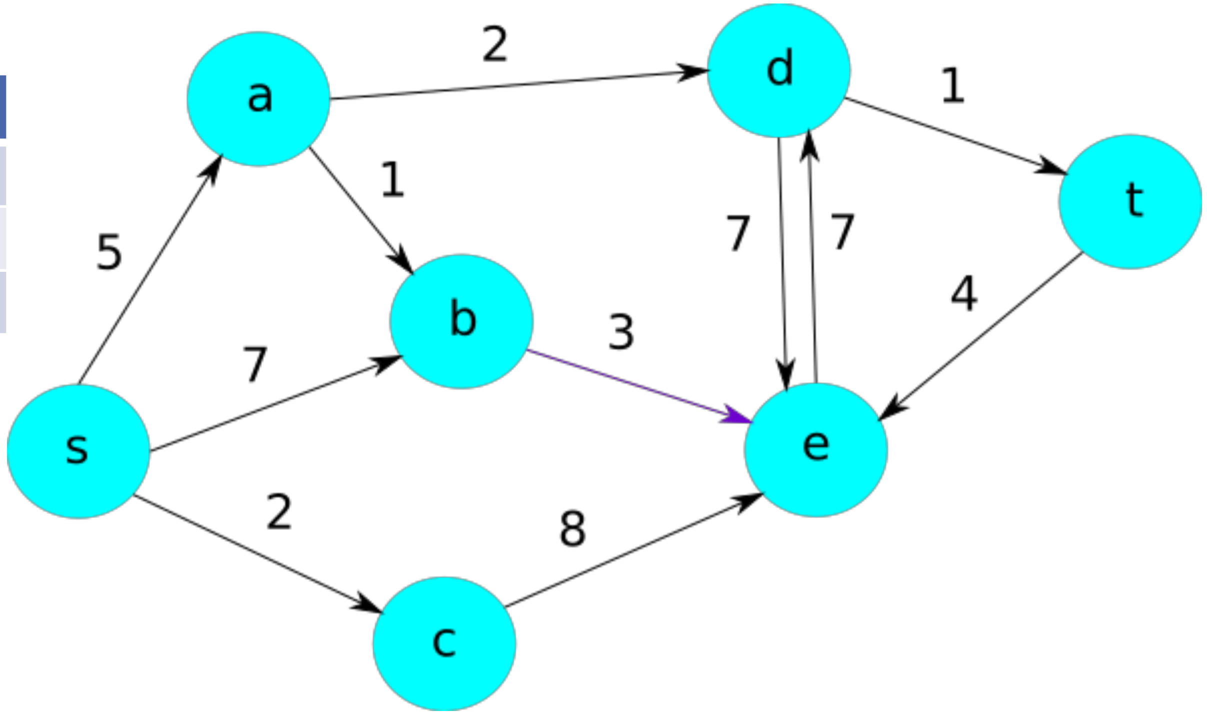


Example - Processing b

Open Min Heap:

Node	Cost to go
d	7
e	9

Closed Set: s
c
a
b

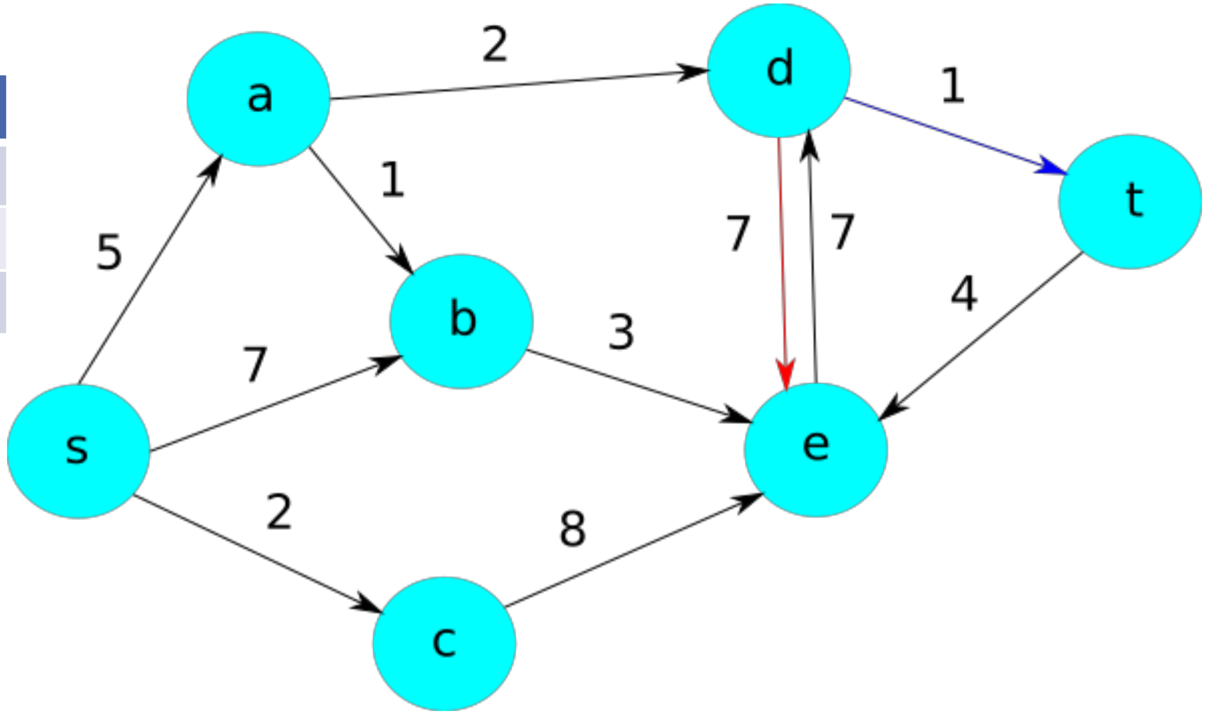


Example - Processing d

Open Min Heap:

Node	Cost to go
t	8
e	9

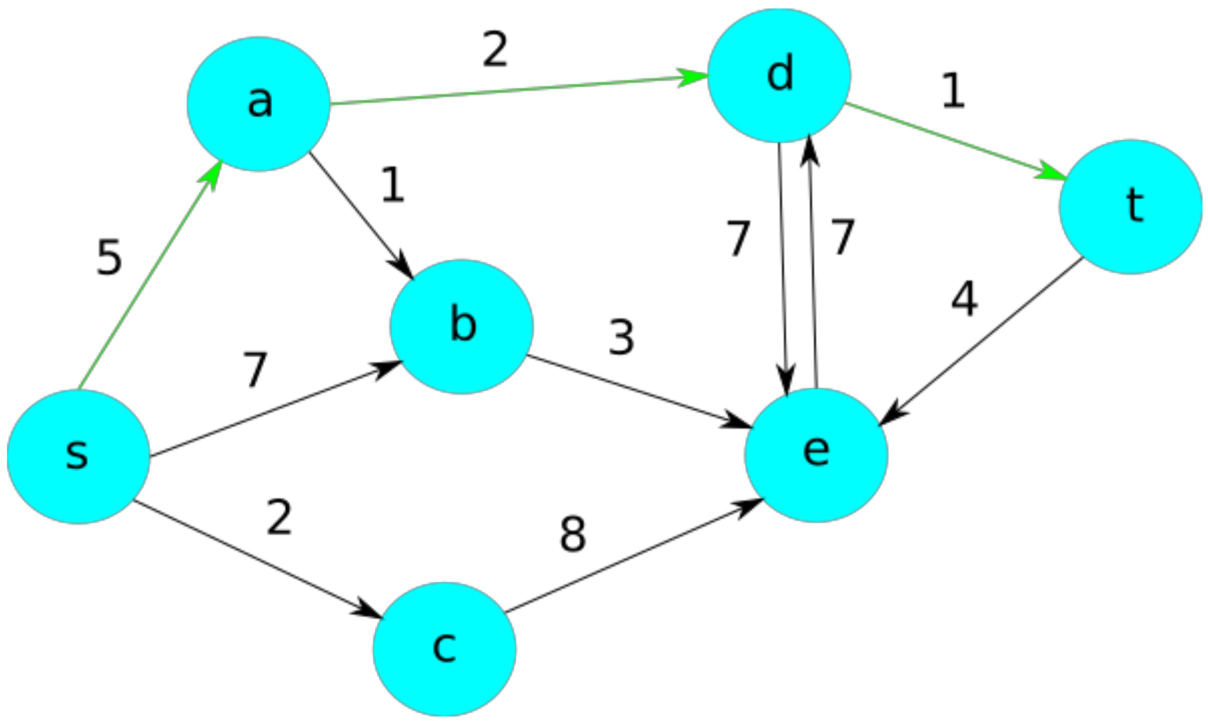
Closed Set: s
c
a
b
d



Example - Optimal Path

Final path: s
a
d
t

Closed Set: s
c
a
b
d



Search on a Map

- Example - map of Berkeley, California
  - 2,097 vertices
  - 5,740 edges

- Example - map of New York City, New York
  - 54,837 vertices
  - 140,497 edges