INFS4205/7205 Advanced Techniques for High Dimensional Data Management

2020-2021 Spring

Route Planning in Road Network

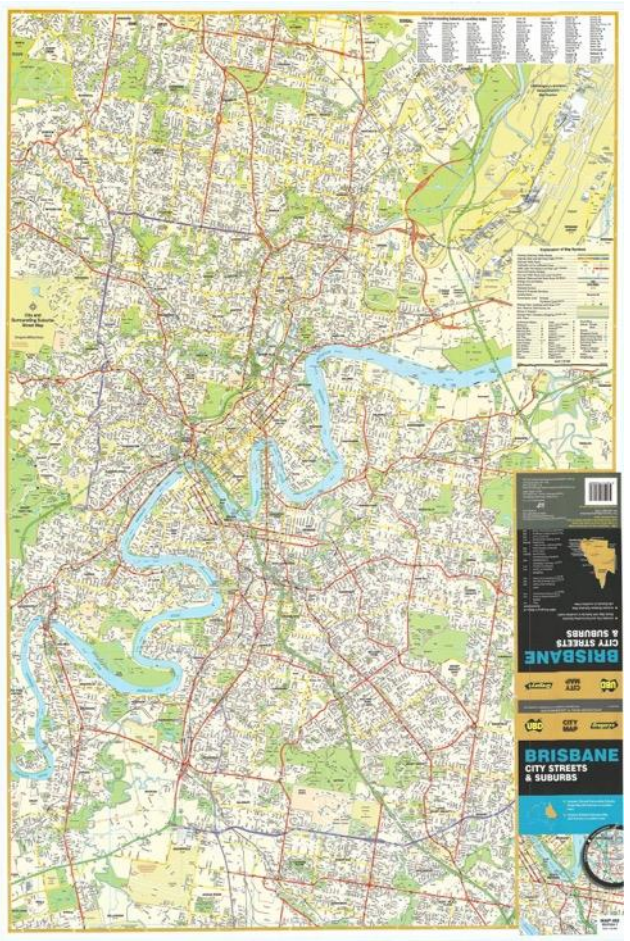
Mengxuan Zhang

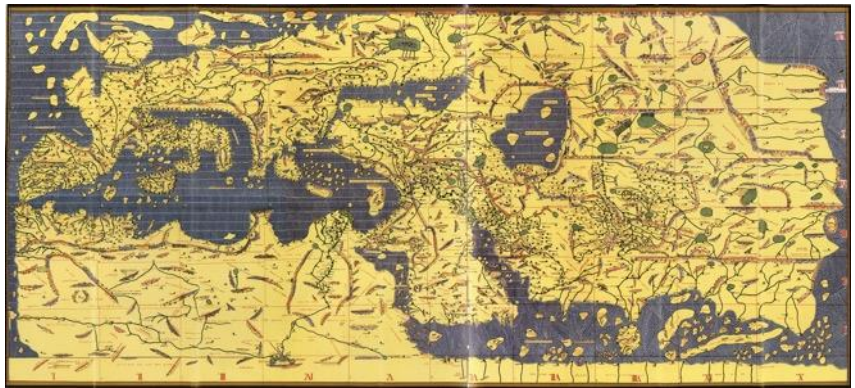
+ Outline

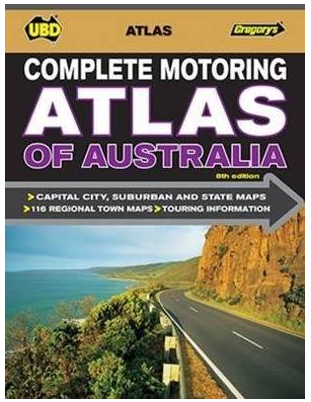
- Background & Path Problems
- Index-Free Algorithms
  - Dijkstra's Algorithms and Its Variations
  - Goal Directed Based Methods
- Index-Based Algorithms
  - Online Search Index Methods
  - Hop Based Methods

+ Background

- Classic Maps
  - Districts, Roads, Streets, POIs, Bus Routes/Stops,…
  - Seas, Mountains, Rivers, …
  - Fixed Accuracy (Scale)
  - Basic Navigation
    - Approximate Distance
    - Approximate Direction
    - Approximate Current Location

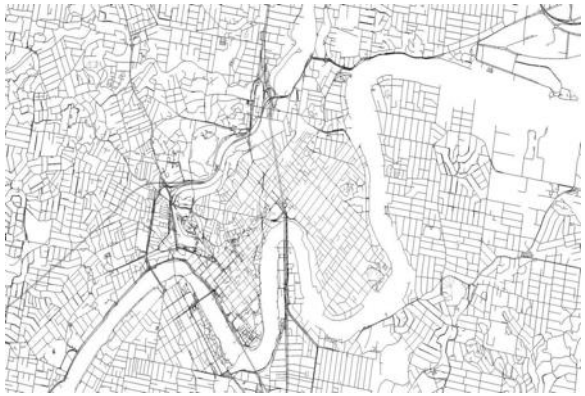


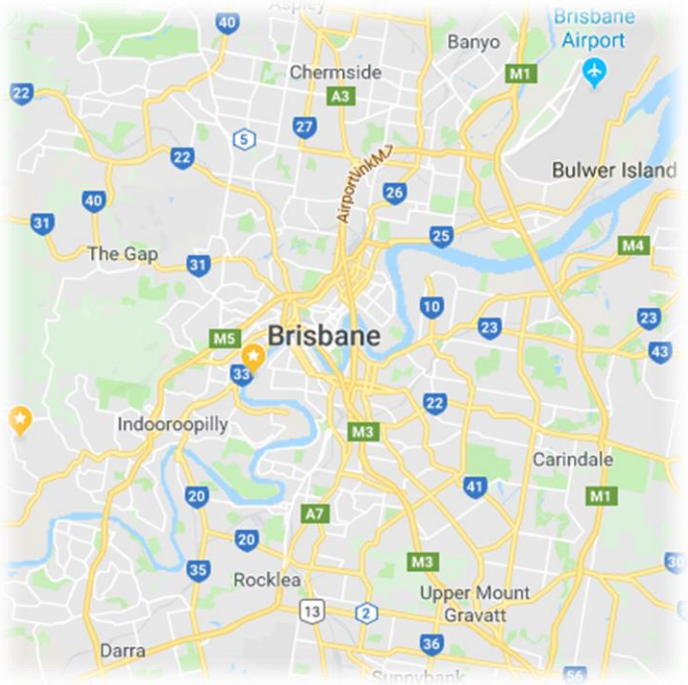




+ Background

- Digital Maps & GPS
  - Any resolution
  - Accurate location information
  - Accurate distance
  - Traffic information
  - Road Network
    - Roads, Intersections
    - Path planning with algorithms



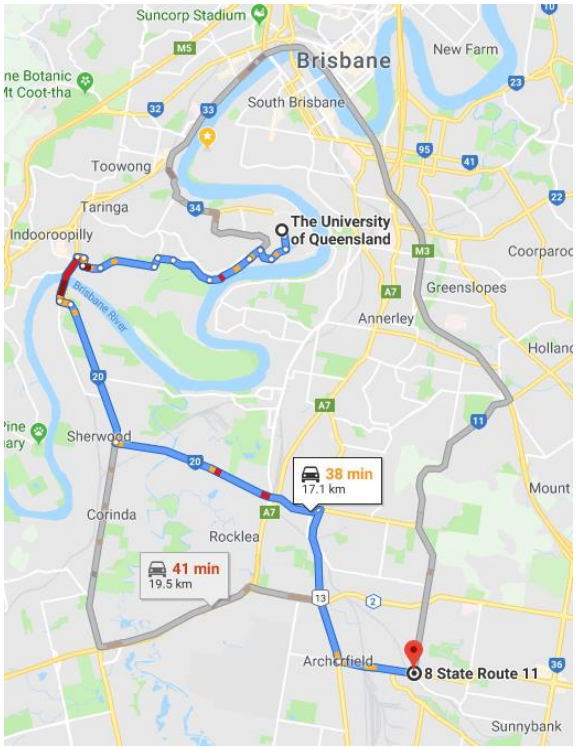


+ Background

- Where there is a network, there is a path
  - Where there is a path, there is a connection, a chain of interactions, a distance, a flow of transmission, …
- Building block for many other applications
  - kNN, Skyline,…
  - Route / Trip scheduling
  - Carpooling assignment and management
  - Transportation system optimization of smart city
  - …

+ Path Problems

- Examples
  1. Shortest path from UQ to Coopers Plain
  2. Fastest path from UQ to Coopers Plain when I depart at 5pm
  3. Fastest path from UQ to Coopers Plain when I depart between 5pm and 7pm
  4. Paths from Indooroopilly, City and Woolloongaba to UQ such that everybody can arrive by 10am
  5. Fastest path from West End to Airport with toll fee under $6 (Go Between? M7?)
  6. Earliest to arrive Coopers Plain by bus when I depart at 5pm with exchange times under 3
  7. I want to spend at least 3 hours at UQ, 1 hour at Garden City, 1 hour at Southbank Parkland and arrive home by 9pm, what's the schedule to waste the least time for transportation?



+ Path Problems

- Examples
  1. Shortest path from UQ to Coopers Plain
  2. Fastest path from UQ to Coopers Plain when I depart at 5pm
  3. Fastest path from UQ to Coopers Plain when I depart between 5pm and 7pm
  4. Paths from Indooroopilly, City and Woolloongaba to UQ such that everybody can arrive by 10am
  5. Fastest path from West End to Airport with toll fee under $6 (Go Between? M7?)
  6. Earliest to arrive Coopers Plain by bus when I depart at 5pm with exchange times under 3.
  7. I want to spend at least 3 hours at UQ, 1 hour at Garden City, 1 hour at Southbank Parkland and arrive home by 9pm, what's the schedule to waste the least time for transportation?

- Distance
- Time, Fixed Departure
- Time, Flexible Departure
- Time, Multi-Source/Single Destination
- Time, Cost Constraint
- Timetable, Fixed Departure, Constraint
- Time, Visiting/Staying Allowed

+ Path Problems

- Dimensions
  - Objectives
    - The main value to minimize
      - Distance: Static value
      - Time: Value changes over time
        - Continuous / Discrete (Timetable)
        - Departure time
          - Fixed
          - Flexible: Choose the best departure time
        - Staying
  - Source / Destination Number
    - Single / Multiple
    - 1-N, N-1, M-N, All-Pair
  - Objective Numbers
    - Single
    - Multiple: Constraints over the main objective

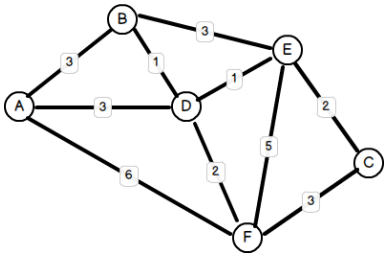
+ Path Problems

Objective	Departure Time	O/D	Constraint	Problem	Example
Static	Fixed	1-1 / 1-N	No	Single Source Shortest Path	Shortest path from UQ to Coopers Plain
		M-N		All Pair Shortest Path	Shortest Paths from Indooroopilly, City and Woolloongaba to UQ
		1-1	Yes	Constraint Shortest Path	Shortest path from West End to Airport with toll fee under $6
Time - Dependent	Fixed	1-1 / 1-N	No	Single Starting Fastest Path	Fastest path from UQ to Coopers Plain when I depart at 5pm
	Interval	1- 1		Interval Starting Fastest Path	Fastest path from UQ to Coopers Plain when I depart between 5pm and 7pm
		1 - N			Paths from Indooroopilly, City and Woolloongaba to UQ such that everybody can arrive by 10am
		1-M-1		Minimal On-Road Time Path	I want to spend at least 3 hours at UQ, 1 hour at Garden City, 1 hour at Southbank Parkland and arrive home by 9pm, what's the schedule to waste the least time for transportation?
		1-1/1-N	Yes	Constraint Fastest Path	Fastest path from West End to Airport with toll fee under $6
Timetable - Dependent	…	…	…	Timetable Paths	Earliest to arrive Coopers Plain by bus when I depart at 5pm with exchange times under 3

+ Road Network Models

Static Graph

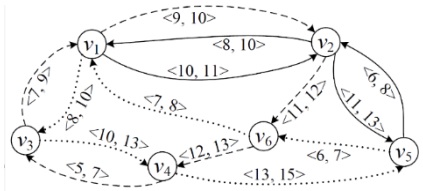
Edge has only 1 weight



Shortest Path

Timetable Graph

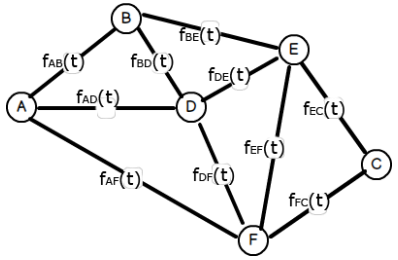
Edge has discrete Weight



Bus, Airplane, Train, Subway, Ferry, …

Time Dependent Graph

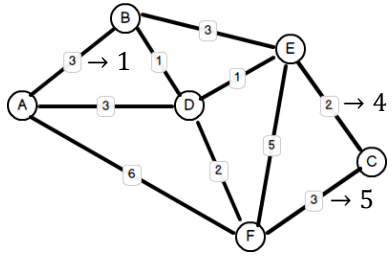
Edge has continuous Weight



Fastest Path

Dynamic Graph

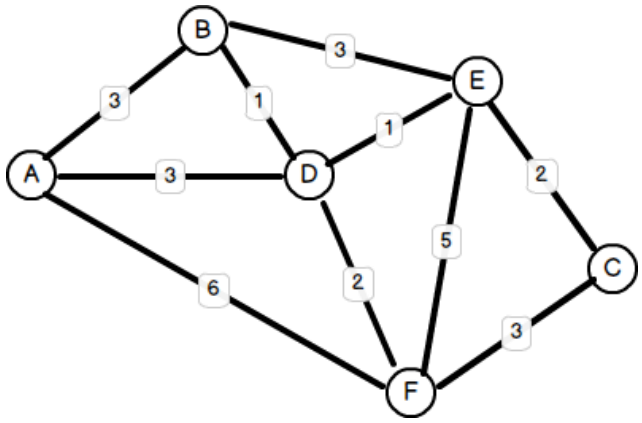
Edge weight change is ad-hoc



Real Life

+ Static Graph

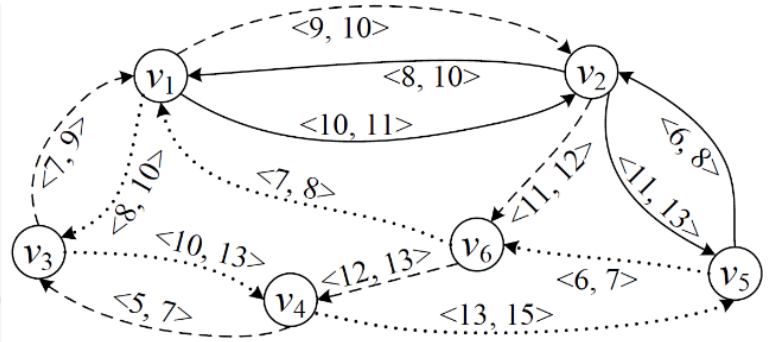
- $G(V, E, W)$
  - $V = \{v_i\}$ Vertex set Intersections
  - $E \subseteq V \times V$ Edge set Roads
    - Directed / Undirected
    - (v_i, v_j)
      - v_j is $v_i's$ out-neighbor
        - v_j is adjacent to v_i
      - v_i is $v_j's$ in-neighbor
        - v_i is adjacent to v_j



  - W A set of static weights
    - $\forall(v_i, v_j) \in E\ , w(v_i, v_j) \in W$
    - Weighted / Unweighted

+ Timetable Graph

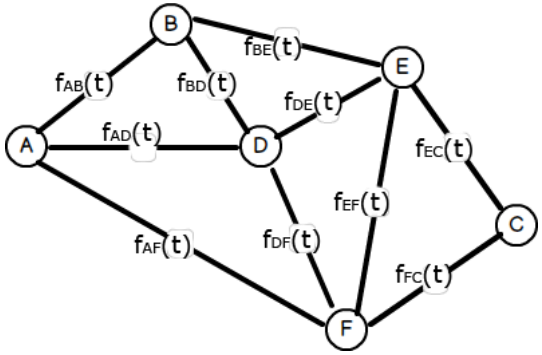
- $G(V, E, W)$
  - $V = \{v_i\}$ Vertex set Stops
  - $\forall(v_i, v_j, t, \lambda) \in E$ Edge set Transportations
    - v_i, v_j: End points
    - t: Departure time
    - λ: Travel cost
    - Multiple edges between two vertices
    - Edge existence is time-dependent

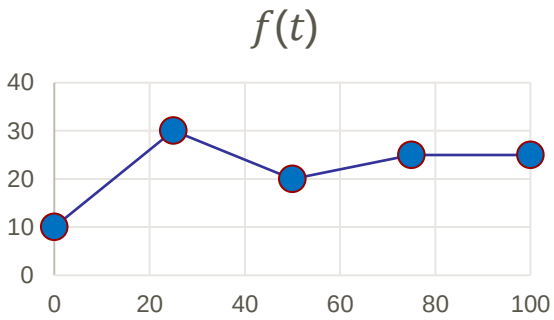


Stop	Trip details
University QLD Lakes Station	2:10 pm
Dutton Park Place Busway	2:11 pm
Boggo Road Station	2:13 pm
PA Hospital Station	2:14 pm
Mater Hill Station	2:18 pm
South Bank Busway Station	2:20 pm
Cultural Centre Station	2:24 pm
King George Square Bus Station	2:28 pm
Roma Street Busway Station	2:30 pm
Normanby Station	2:33 pm
QUT Kelvin Grove Station	2:36 pm
Herston Station	2:37 pm
RBWH Station	2:40 pm

+ Time-Dependent Graph

- $G_T(V, E, W)$
  - $V = \{v_i\}$ Vertex set Stops
  - $E \subseteq V \times V$ Edge set Roads
  - F A set of functions
    - $\forall (v_i, v_j) \in E$, $f_{i,j}(t) \in W$, $t \in \mathbb{T}$
    - Tells how much time it costs to travel from v_i to v_j at time t





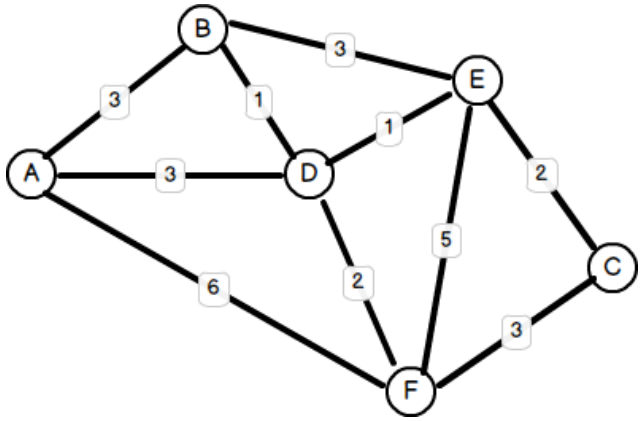
+ Dynamic Graph

- $G_D(V, E, W, \{\Delta W\})$
  - $V = \{v_i\}$ Vertex set Stops
  - $E \subseteq V \times V$ Edge set Roads
  - W A set of current weights
  - ΔW A set of weight updates
    - A subset of W
    - Unknow beforehand

+ Graph Representations

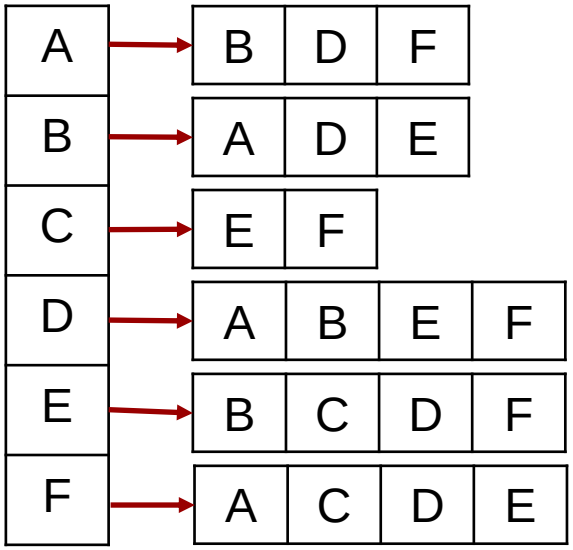
- Adjacency Matrix
  - Store weights of all vertex pairs
    - Space
      - $|V|^2$

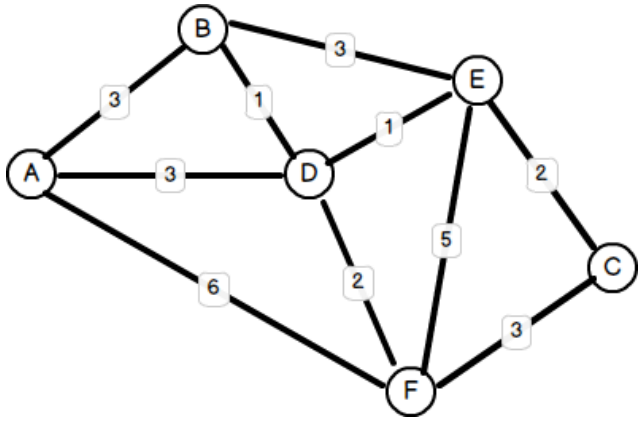
	A	B	C	D	E	F
A	0	3	∞	3	∞	6
B	3	0	∞	1	3	∞
C	∞	∞	0	∞	2	3
D	3	1	∞	0	1	2
E	∞	3	2	1	0	5
F	6	∞	3	2	5	0



+ Graph Representations

- Adjacency List
  - Store the list of neighbors of each vertex
    - Space
      - Directed: $|V| + |E|$
      - Undirected: $|V| + 2|E|$





+ Shortest Path

- Path: A sequence of consecutive vertices
  - $p = < v_0, v_1, \dots, v_k >$
    - $(v_i, v_{i+1}) \in E, \quad \forall i \in (0, k-1)$
  - Path Length: $w(p) = \sum_0^{k-1} w(v_i, v_{i+1})$
  - Shortest path from s to t
    - $\min(w(p_{s,t}))$

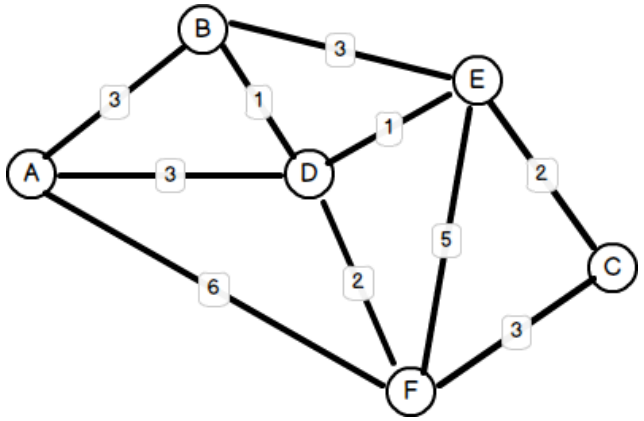
Path from A to C

$w(A \to F \to C) = 9$

$w(A \to B \to E \to C) = 8$

$w(A \to B \to D \to E \to C) = 7$

$w(A \to D \to E \to C) = 6$



+ Shortest Path: Dijkstra's Algorithm

- Priority Queue Q
  - Current shortest distance from s to each vertex
  - Top value: smallest distance in Q
  - Initialization
    - $d(s) = 0$, push s in Q
    - $d(v) = \infty, \forall v \in V - s$

v	d
A	0
B	∞
C	∞
D	∞
E	∞
F	∞

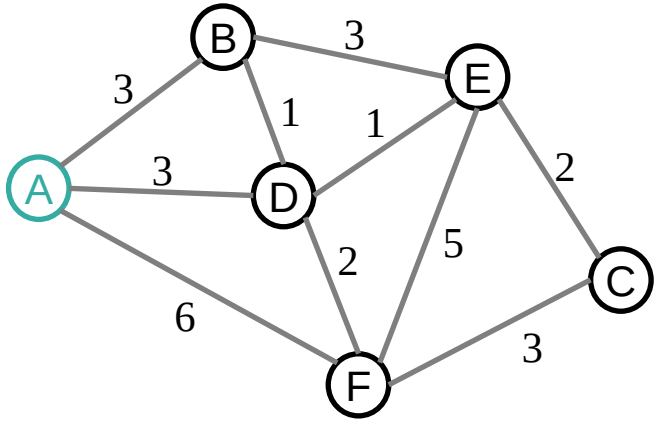
$A(0)$

(V) Updated Vertex: In Q with distance $< \infty$

(V) Top Vertex: Minimum in Q

(V) Visited Vertex: Not in Q, Found Shortest

(V) Unvisited Vertex: Not in Q, still ∞



+ Shortest Path: Dijkstra's Algorithm

- Neighbor Update
  - Pop out top vertex A from Q
  - Update A's out-neighbors
    - $d(B) = 0 + 3 < \infty, B \notin Q$, push B in Q
    - $d(D) = 0 + 3 < \infty, D \notin Q$, push D in Q
    - $d(F) = 0 + 6 < \infty, F \notin Q$, push F in Q

v	d
A	0
B	∞
C	∞
D	∞
E	∞
F	∞

v	d
~~A~~	~~0~~
B	3
C	∞
D	3
E	∞
F	6

$A(0)$ → $B(\mathbf{3})$

$D(3)$

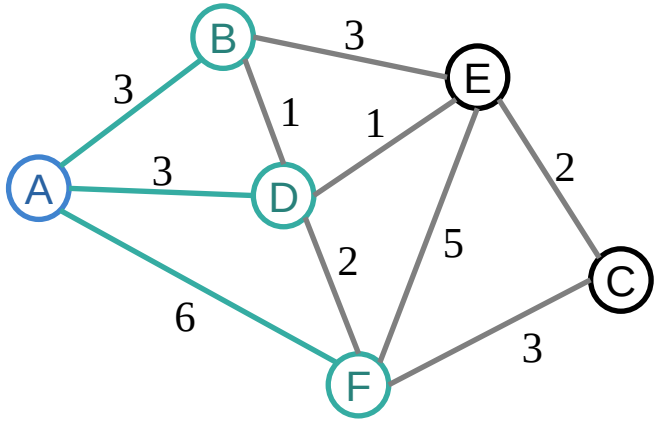
$F(6)$

V Updated Vertex: In Q with distance $< \infty$

V Top Vertex: Minimum in Q

V Visited Vertex: Not in Q, Found Shortest

V Unvisited Vertex: Not in Q, still ∞



+ Shortest Path: Dijkstra's Algorithm

- Neighbor Update
  - Pop out top vertex B from Q
  - Update B's out-neighbors
    - A already popped
    - $d(D) = 3 + 1 > 3$, skip
    - $d(E) = 3 + 3 < \infty, E \notin Q$, push E in Q

v	d
~~A~~	~~0~~
B	3
C	∞
D	3
E	∞
F	6

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	∞
D	3
E	6
F	6

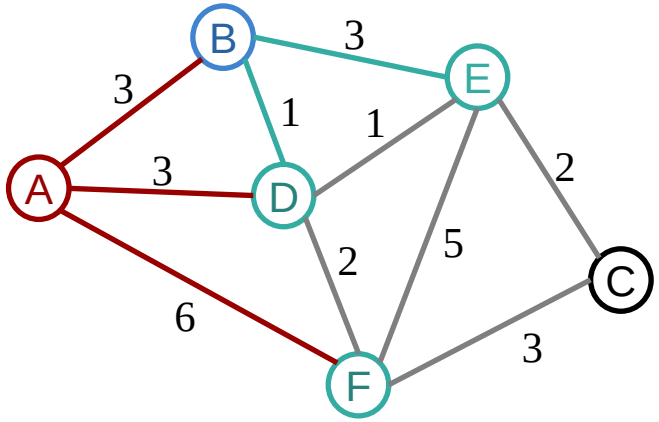
$B(3)$	$D(3)$
$D(3)$	$E(6)$
$F(6)$	$F(6)$

V Updated Vertex: In Q with distance $< \infty$

V Top Vertex: Minimum in Q

V Visited Vertex: Not in Q, Found Shortest

V Unvisited Vertex: Not in Q, still ∞



Shortest Path: Dijkstra's Algorithm

- Neighbor Update
  - Pop out top vertex D from Q
  - Update D's out-neighbors
    - A, B already popped
    - $d(E) = 3 + 1 < 6$, update $d(E) = 4$
    - $d(F) = 3 + 2 < 6$, update $d(F) = 5$

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	∞
D	3
E	6
F	6

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	∞
~~D~~	~~3~~
E	4
F	5

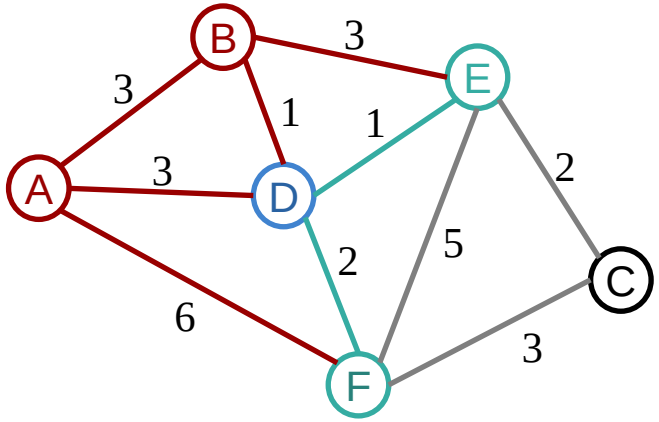
$D(3)$	$E(4)$
$E(6)$	$F(5)$
$F(6)$	

V Updated Vertex: In Q with distance $< \infty$

V Top Vertex: Minimum in Q

V Visited Vertex: Not in Q, Found Shortest

V Unvisited Vertex: Not in Q, still ∞



+ Shortest Path: Dijkstra's Algorithm

- Neighbor Update
  - Pop out top vertex E from Q
  - Update E's out-neighbors
    - B, D already popped
    - $d(C) = 4 + 2 < \infty$, push in Q
    - $d(F) = 4 + 5 > 5,$ skip

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	∞
~~D~~	~~3~~
E	4
F	5

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	6
~~D~~	~~3~~
~~E~~	~~4~~
F	5

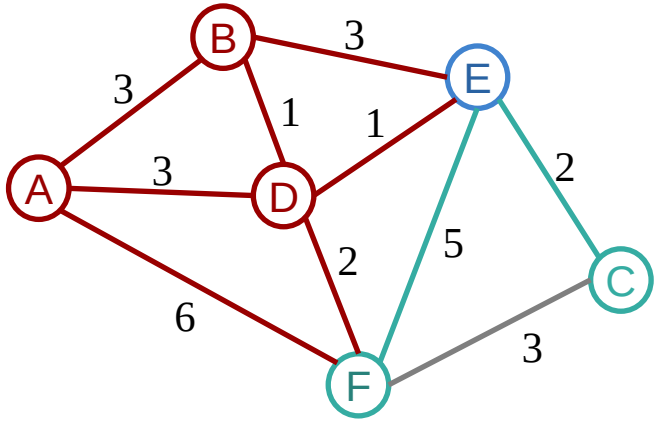
$E(4)$	$F(5)$
$F(5)$	$C(6)$

(V) Updated Vertex: In Q with distance $< \infty$

(V) Top Vertex: Minimum in Q

(V) Visited Vertex: Not in Q, Found Shortest

(V) Unvisited Vertex: Not in Q, still ∞



+ Shortest Path: Dijkstra's Algorithm

- Neighbor Update
  - Pop out top vertex F from Q
  - Update F's out-neighbors
    - A, D, E already popped
    - $d(C) = 5 + 3 > 6$, skip

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	6
~~D~~	~~3~~
~~E~~	~~4~~
F	5

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	6
~~D~~	~~3~~
~~E~~	~~4~~
~~F~~	~~5~~

$F(5)$ → $C(6)$

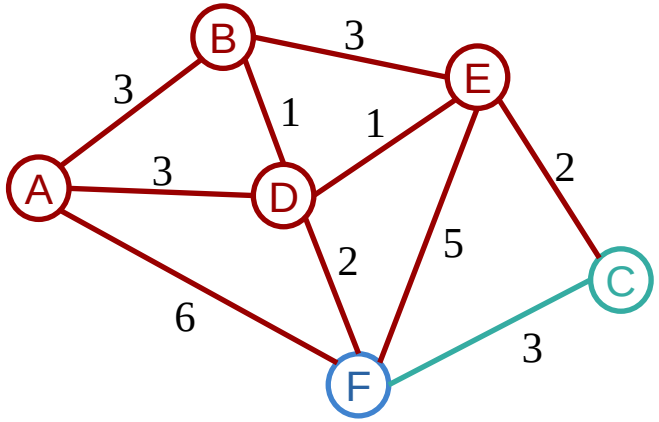
$C(6)$

(V) Updated Vertex: In Q with distance $< \infty$

(V) Top Vertex: Minimum in Q

(V) Visited Vertex: Not in Q, Found Shortest

(V) Unvisited Vertex: Not in Q, still ∞



Shortest Path: Dijkstra's Algorithm

- Neighbor Update
  - Pop out top vertex C from Q
    - Got the shortest Distance to $d(A, C) = 6$

V Updated Vertex: In Q with distance $< \infty$

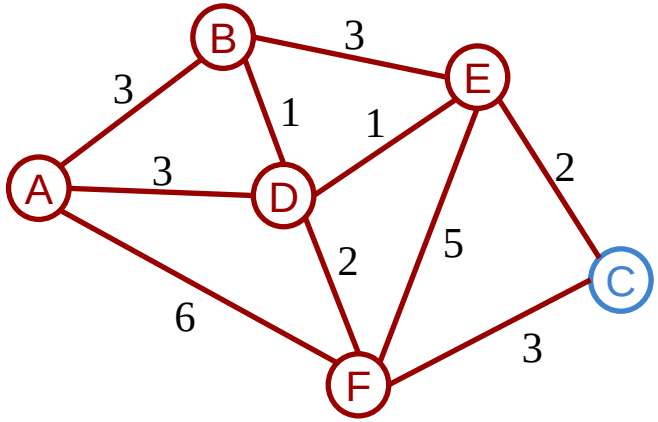
V Top Vertex: Minimum in Q

V Visited Vertex: Not in Q, Found Shortest

V Unvisited Vertex: Not in Q, still ∞

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	6
~~D~~	~~3~~
~~E~~	~~4~~
~~F~~	~~5~~

$C(6)$



Shortest Path: Dijkstra's Algorithm

- Correctness
  - The top value $d(u)$ in Q is the shortest distance from s to u
    - $\forall v \neq u, v$ has three cases:
      1. Visited
      2. Updated
      3. Unvisited

Ⓥ Updated Vertex: In Q with distance $< \infty$

~~Ⓥ Top Vertex: Minimum in Q~~

Ⓥ Visited Vertex: Not in Q, Found Shortest

Ⓥ Unvisited Vertex: Not in Q, still ∞

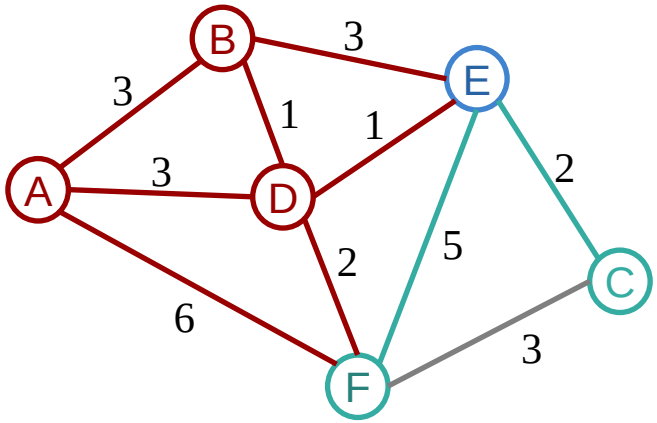
v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	∞
~~D~~	~~3~~
E	4
F	5

⇨

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	6
~~D~~	~~3~~
~~E~~	~~4~~
F	5

$E(4)$ ⇨ $F(5)$

$F(5)$ $C(6)$



Shortest Path: Dijkstra's Algorithm

- Correctness

Ⓥ Visited Vertex: Not in Q, Found Shortest

  - The top value $d(u)$ in Q is the shortest distance from s to u
    1. $\forall\, v$ that have been popped out, it will not appear in Q again
       - u will not be updated by v
       - Example:
         - $A(0), B(3)$ and $D(3)$ are popped out
         - They will not update $E(4)$, their contributions are already considered

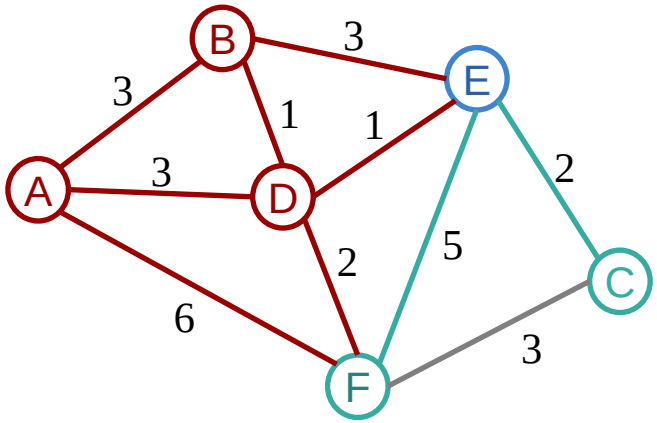
v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	∞
~~D~~	~~3~~
E	4
F	5

⇨

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	6
~~D~~	~~3~~
~~E~~	~~4~~
F	5

$E(4)$ ⇨ $F(5)$

$F(5)$ $C(6)$



Shortest Path: Dijkstra's Algorithm

- Correctness

V Updated Vertex: In Q with distance $< \infty$

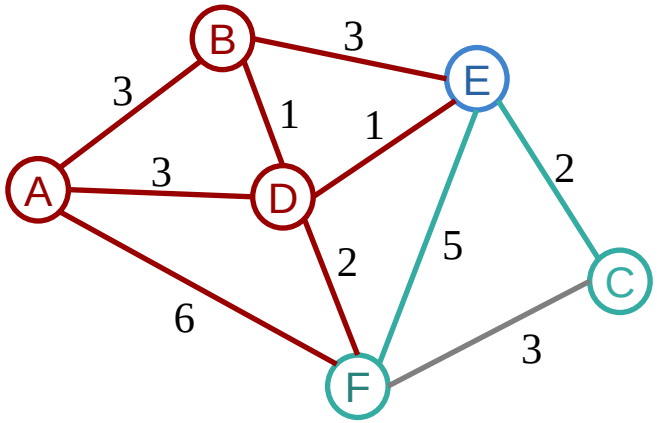
  - The top value $d(u)$ in Q is the shortest distance from s to u
    2. $\forall\, v \in Q, d(v) \geq d(u)$
       - $\forall x \in \{(v,x)\}, d(v) + w(v,x) > d(u)$
       - Example:
         - $F(5)$: $F(5) > E(4) \quad \rightarrow \quad F(5) + w > E(4)$

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	∞
~~D~~	~~3~~
E	4
F	5

⇨

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	6
~~D~~	~~3~~
~~E~~	~~4~~
F	5

$E(4)$	$F(5)$
$F(5)$	$C(6)$



+ Shortest Path: Dijkstra's Algorithm

- Correctness

Ⓥ Unvisited Vertex: Not in Q, still ∞

  - The top value $d(u)$ in Q is the shortest distance from s to u
    3. $\forall\, v \notin Q$ and not popped out
       1. Will be eventually updated by $x \in Q$ in the future $\Rightarrow d(v) > d(u)$
       2. Or will remain ∞
       - E and C are unvisited
         - They will be updated by B, D and F, which are longer than A

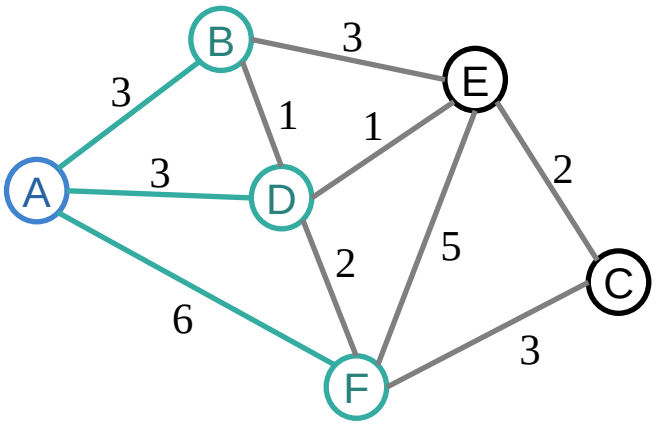
v	d
A	0
B	∞
C	∞
D	∞
E	∞
F	∞

v	d
~~A~~	~~0~~
B	3
C	∞
D	3
E	∞
F	6

$A(0)$ ⇨ $B(\mathbf{3})$

$D(3)$

$F(6)$



Shortest Path: Dijkstra's Algorithm

- Correctness

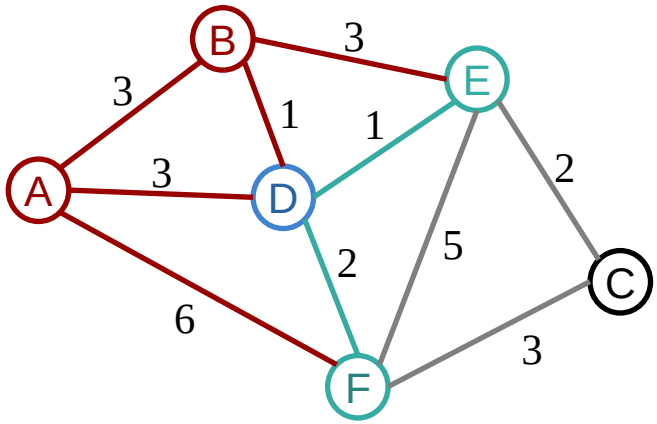
Ⓥ Unvisited Vertex: Not in Q, still ∞

  - The top value $d(u)$ in Q is the shortest distance from s to u
    - $u's$ shortest distance is found
    - $D's$ shortest distance is found
  - When t is the top, terminate search
    - Destination's shortest distance is found

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	∞
D	3
E	6
F	6

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	∞
~~D~~	~~3~~
E	4
F	5

$D(3)$	$E(4)$
$E(6)$	$F(5)$
$F(6)$	

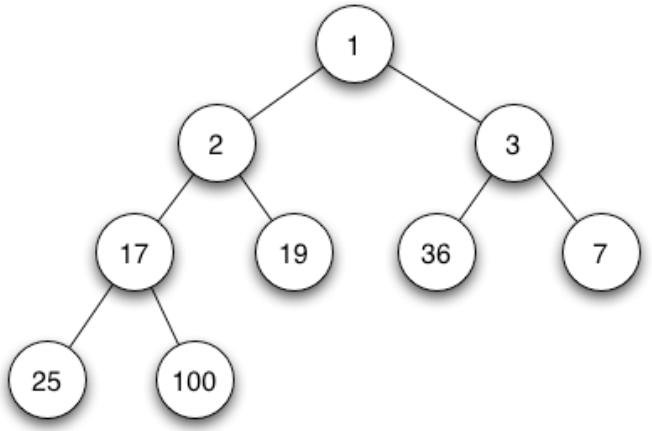


+ Shortest Path: Dijkstra's Algorithm

- Complexity
  - Queue
    - Push: $|V|$ times
    - Deletion: $|V|$ times
    - Update: $|E|$ times

Operation	Operation in Dijkstra's	Binary Heap
Insert	Push	$O(1)$
Delete Min	Pop	$O(logN)$
Decrease Key	Update	$O(logN)$

  - Binary Heap
    - $O(|V| \times 1 + |V| \times \log|V| + E \times \log|V|)$
    - $O((|V| + |E|)\log|V|)$



Shortest Path: Dijkstra's Algorithm

- Single Source Multiple Destinations
  - From s to $T = \{t_1, t_2, \dots, t_n\}$
  - Stop when $t_1, t_2, \dots, t_n$ all become visited

    Ⓥ Visited Vertex: Not in Q, Found Shortest

- What we have got is the shortest distance
  - How to retrieve the shortest path?

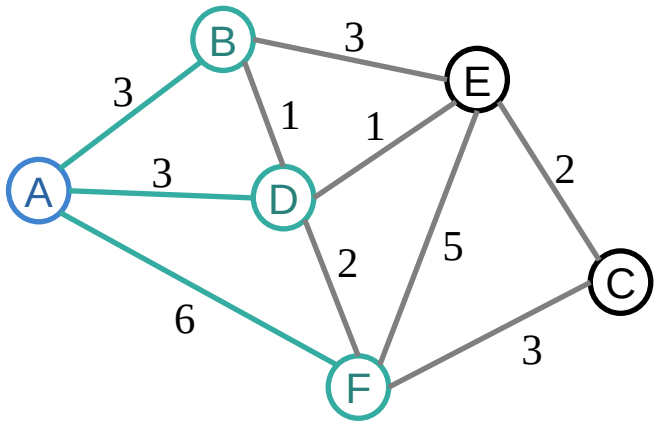
Shortest Path: Dijkstra's Algorithm

- Shortest Path
  - Additional information
    - Store who updated u
    - A updates B, D and F

v	d
A	0
B	∞
C	∞
D	∞
E	∞
F	∞

v	d
~~A~~	~~0~~
B	3
C	∞
D	3
E	∞
F	6

v	P
A	
B	A
C	
D	A
E	
F	A



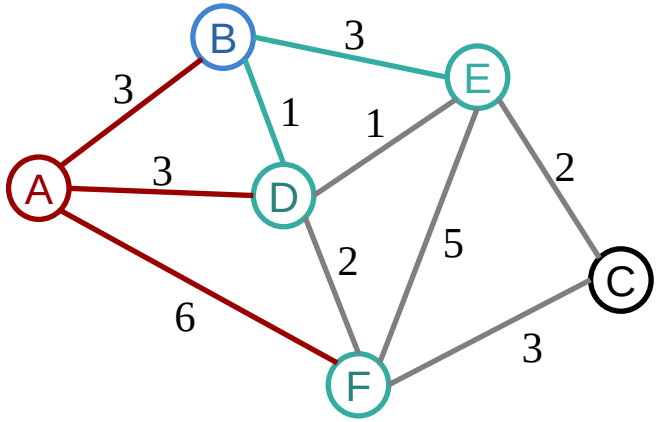
Shortest Path: Dijkstra's Algorithm

- Shortest Path
  - Additional information
    - Store who updated u
    - B updates E

v	d
~~A~~	~~0~~
B	3
C	∞
D	3
E	∞
F	6

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	∞
D	3
E	6
F	6

v	P
A	
B	A
C	
D	A
E	B
F	A



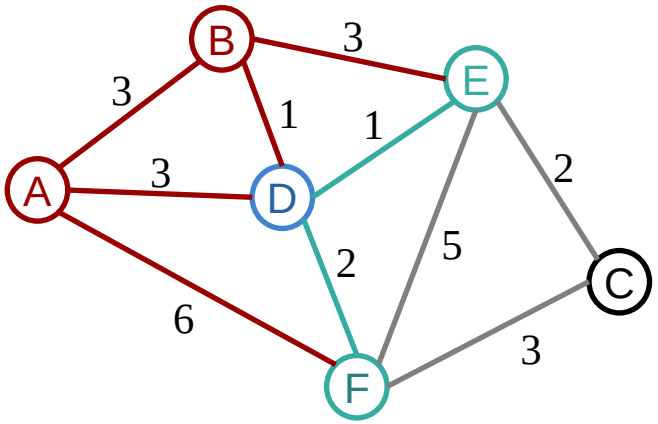
Shortest Path: Dijkstra's Algorithm

- Shortest Path
  - Additional information
    - Store who updated u
    - D updates E and F

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	∞
D	3
E	6
F	6

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	∞
~~D~~	~~3~~
E	4
F	5

v	P
A	
B	A
C	
D	A
E	D
F	D



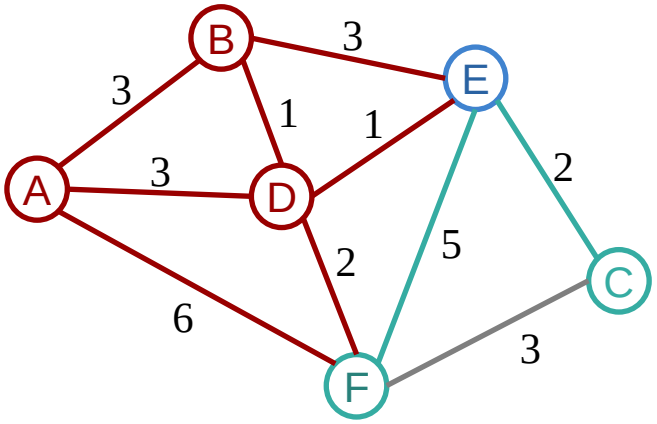
Shortest Path: Dijkstra's Algorithm

- Shortest Path
  - Additional information
    - Store who updated u
    - E updates C

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	∞
~~D~~	~~3~~
E	4
F	5

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	6
~~D~~	~~3~~
~~E~~	~~4~~
F	5

v	P
A	
B	A
C	E
D	A
E	D
F	D



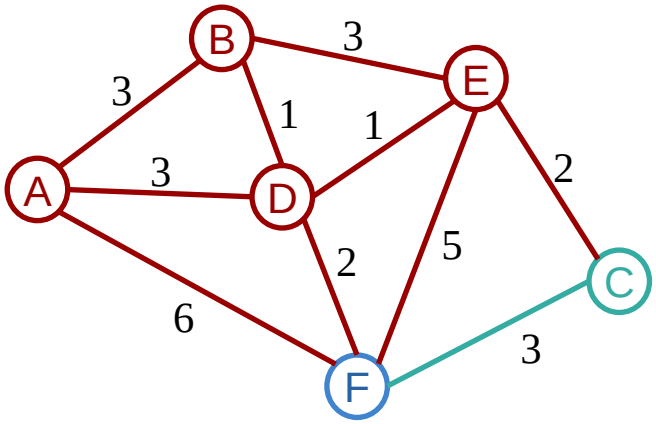
Shortest Path: Dijkstra's Algorithm

- Shortest Path
  - Additional information
    - Store who updated u
    - F updates nothing

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	6
~~D~~	~~3~~
~~E~~	~~4~~
F	5

v	d
~~A~~	~~0~~
~~B~~	~~3~~
C	6
~~D~~	~~3~~
~~E~~	~~4~~
~~F~~	~~5~~

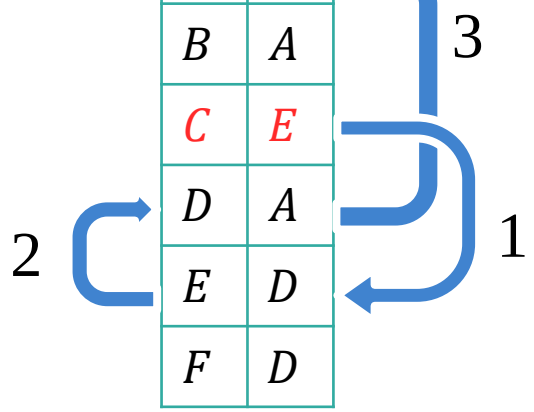
v	P
A	
B	A
C	E
D	A
E	D
F	D

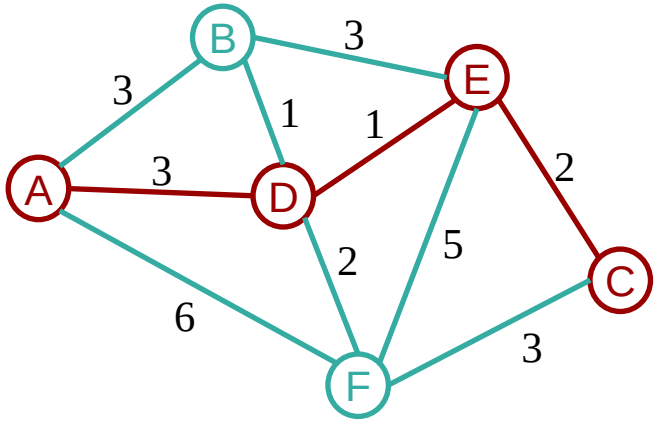


Shortest Path: Dijkstra's Algorithm

- Shortest Path
  - Traverse back from C
    - $C \leftarrow E \leftarrow D \leftarrow A$

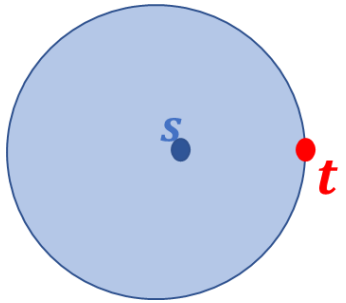
v	P
A	
B	A
C	E
D	A
E	D
F	D



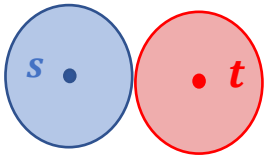


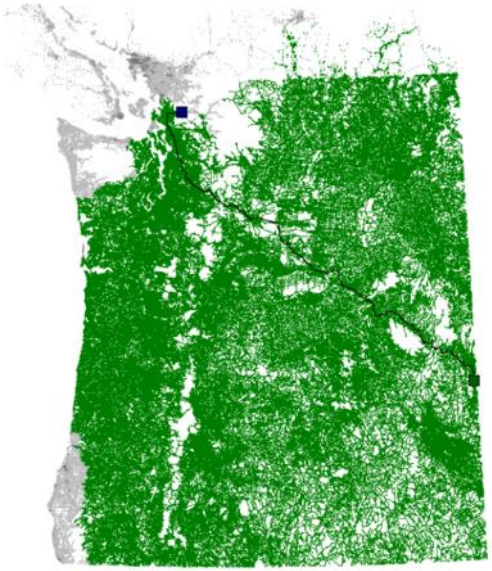
+ Shortest Path: Bi-Dijkstra's

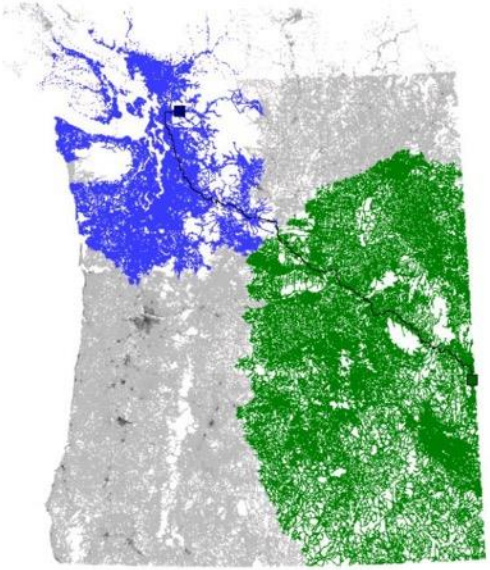
- Dijkstra's Search Space
  - Expand as a circle
  - Most of the search space is useless



- Bi-Directional Dijkstra's
  - Replace a large circle with 2 smaller ones





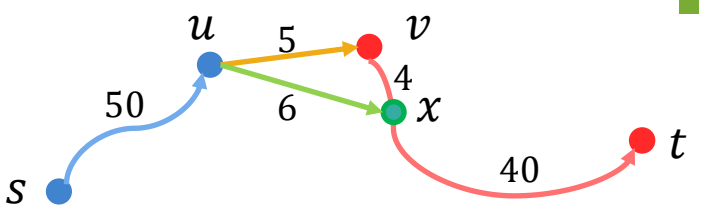


+ Shortest Path: Bi-Dijkstra's

- Forward Search
  - From s with distance $d_f(u)$
  - Visit out-neighbors (original graph)
- Backward Search
  - From t with distance $d_b(v)$
  - Visit in-neighbors (reverse graph)
- Termination
  - u is popped both forwardly and backwardly
  - $d(s,t) = d_f(u) + d_b(u)$
  - NO !

+ Shortest Path: Bi-Dijkstra's

- Example
  - Forward Search
    - u has been popped out
      - $d_f(u) = 50$
    - x is updated
      - $d_f(x) = 56$
    - Current top is v
      - $d_f(v) = 55$
  - Backward Search
    - v has been popped out
      - $d_b(v) = 44$
    - x has been popped out
      - $d_b(x) = 40$
    - u has been updated
      - $d_b(u) = 46$



$u(50)$

$v(55)$

$x(56)$

$x(40)$

$v(44)$

$u(46)$

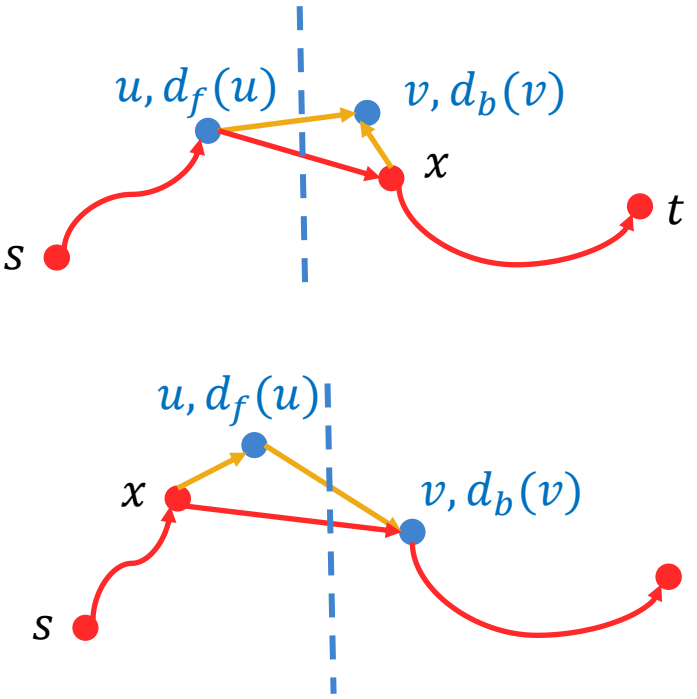
- Forward Settled: u, v
- Backward Settled: v, x

- $d_f(v) + d_b(v) = 55 + 44 = 99$
- $d_u(u) + w(u,x) + d_b(x)$

= 50 + 6 + 40 = 96 < 99

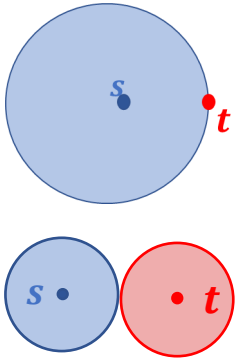
+ Shortest Path: Bi-Dijkstra's

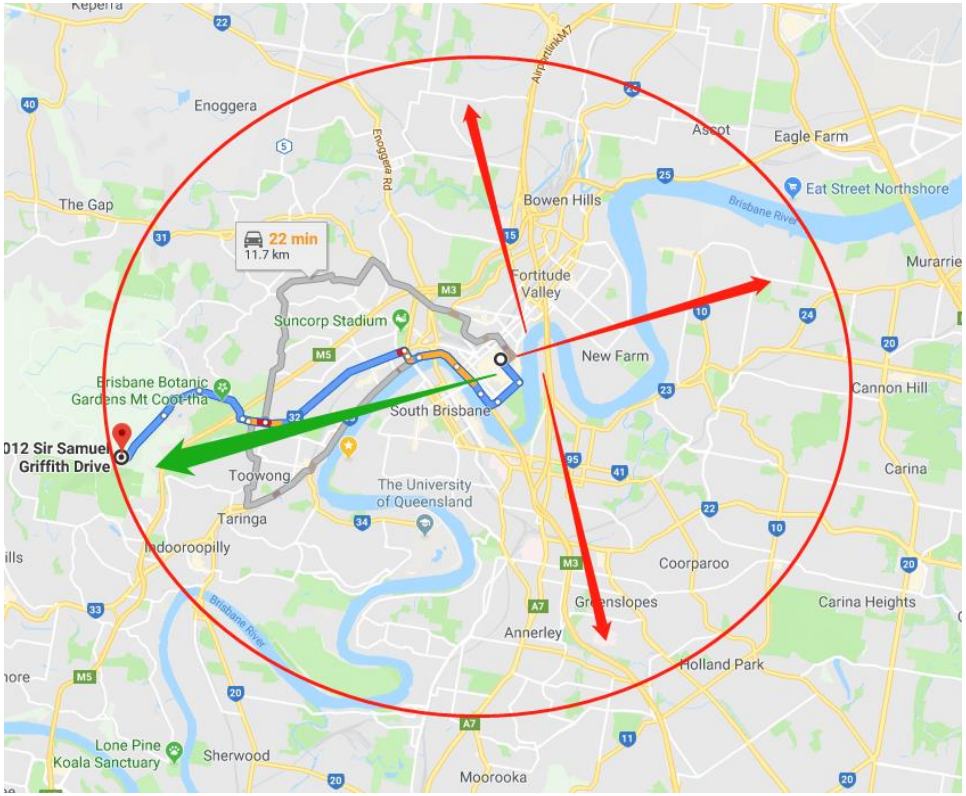
- Maintain μ as the best distance so far
  - $\mu = \infty$
    - Forward Search
      - u is top of Q_f
      - $\forall x \in \{(u, x)\}$ and $d_b(x) \neq \infty$
      - $\mu = \min\left(\mu, d_f(u) + w(u, x) + d_b(x)\right)$
    - Backward Search
      - v is top of Q_b
      - $\forall x \in \{(x, v)\}$ and $d_f(x) \neq \infty$
      - $\mu = \min\left(\mu, d_f(x) + w(x, v) + d_b(v)\right)$
    - Termination
      - $d_f(top) + d_b(top) \geq \mu$



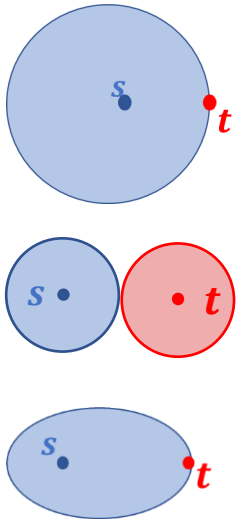
+ Goal Directed: A* Algorithm

- Search Space
  - Still circles
- Heuristic
  - From City to Mt Coot-tha
    - Search towards New Farm?
    - Search towards Bowen Hills?
    - Search towards Annerley?
    - Search towards Mt Coot-tha!

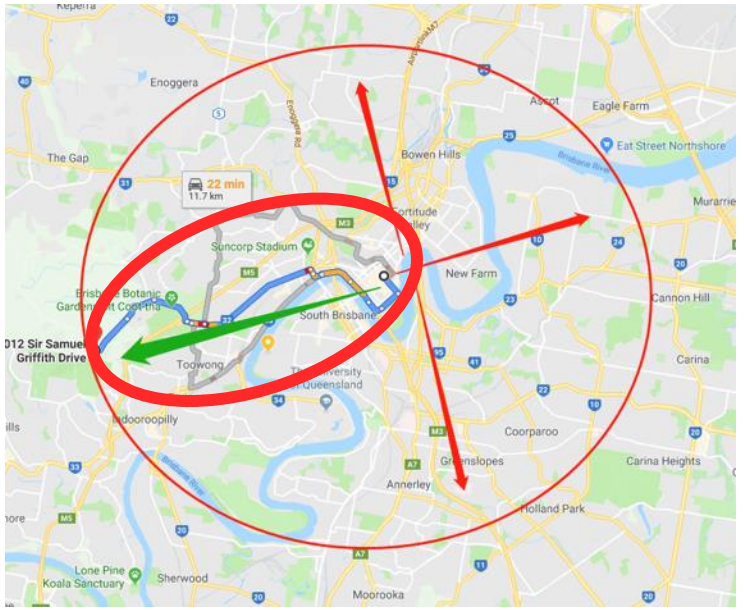




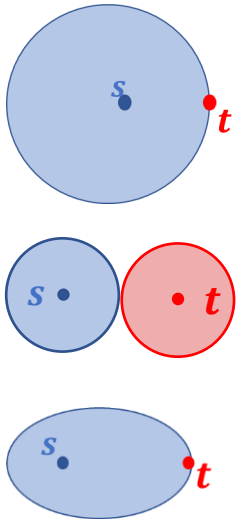
+ Goal Directed: A* Algorithm



- Search Space
  - How to “drag” the search towards the destination
- Heuristic
  - $\pi(u)$: Estimate distance from u to t
    - The vertices nearer to destination are more important!
      - Toowong > Milton > Fortitude Valley > New Farm > Annerley >…
      - $\pi(Toowong) < \pi(Milton) < \pi(\text{Fortitude Valley}) < \cdots$



+ Goal Directed: A* Algorithm

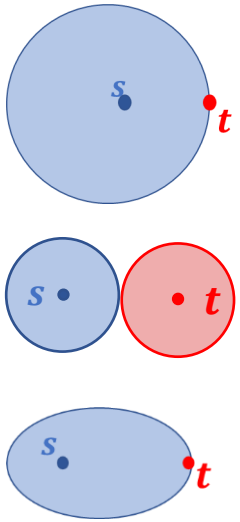


- Search Space
  - How to “drag” the search towards the destination

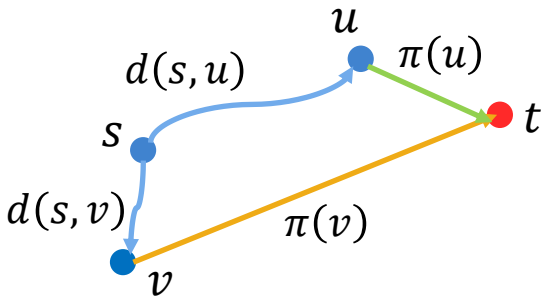
- Distance Importance
  - $Dijkstra's$
    - $d(city, Toowong)$
    - $d(city, Milton)$
    - $d(city, New\ Farm)$
    - $d(city, Fortitude\ Valley)$
  - $Hueristic$
    - $d(city, Toowong) + \pi(Toowong)$
    - $d(city, Milton) + \pi(Milton)$
    - $d(city, Fortitude\ Valley) + \pi(FV)$
    - $d(city, New\ Farm) + \pi(New\ Farm)$

  - We should try Toowong and Milton earlier than FV and New Farm

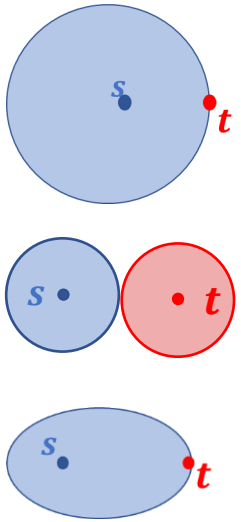
+ Goal Directed: A* Algorithm



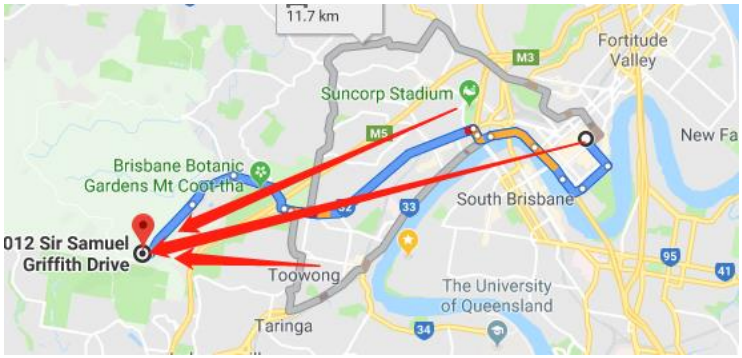
- Search Space
  - How to “drag” the search towards the destination
- Heuristic
  - $\pi(u)$: Estimate distance from u to t
  - If $d(s,u) + \pi(u) < d(s,v) + \pi(v)$
    - $d(city, Toowong) > d(city, New\ Farm)$
    - $d(city, Toowong) + \pi(Toowong) < d(city, New\ Farm) + \pi(New\ Farm)$
    - We should visit u earlier than v even if $d(s,u) > d(s,v)$

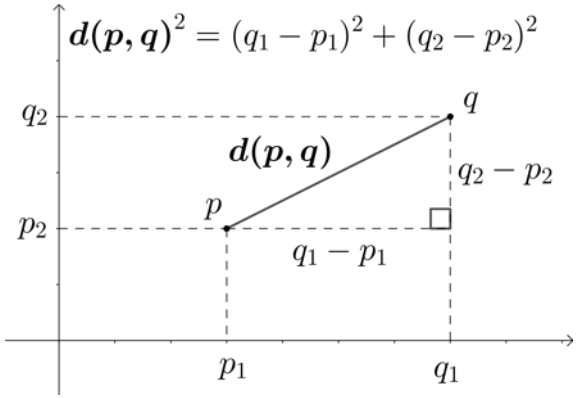


+ Goal Directed: A* Algorithm



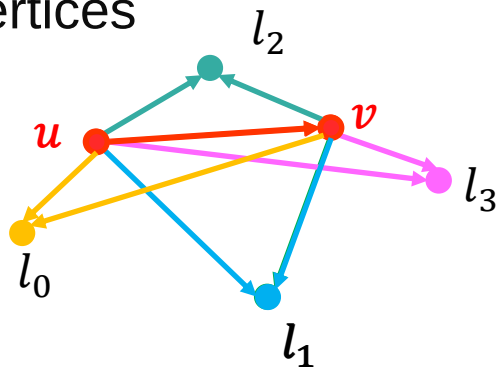
- Search Space
  - How to “drag” the search towards the destination
- Heuristic
  - $\pi(u)$: Estimate distance from u to t
  - Estimation in road network
    - Euclidean Distance
      - Convert the longitude/latitude to distance
      - Shortest straight line distance between 2 vertices
      - Always not longer than the roads/paths
      - Different Implementations



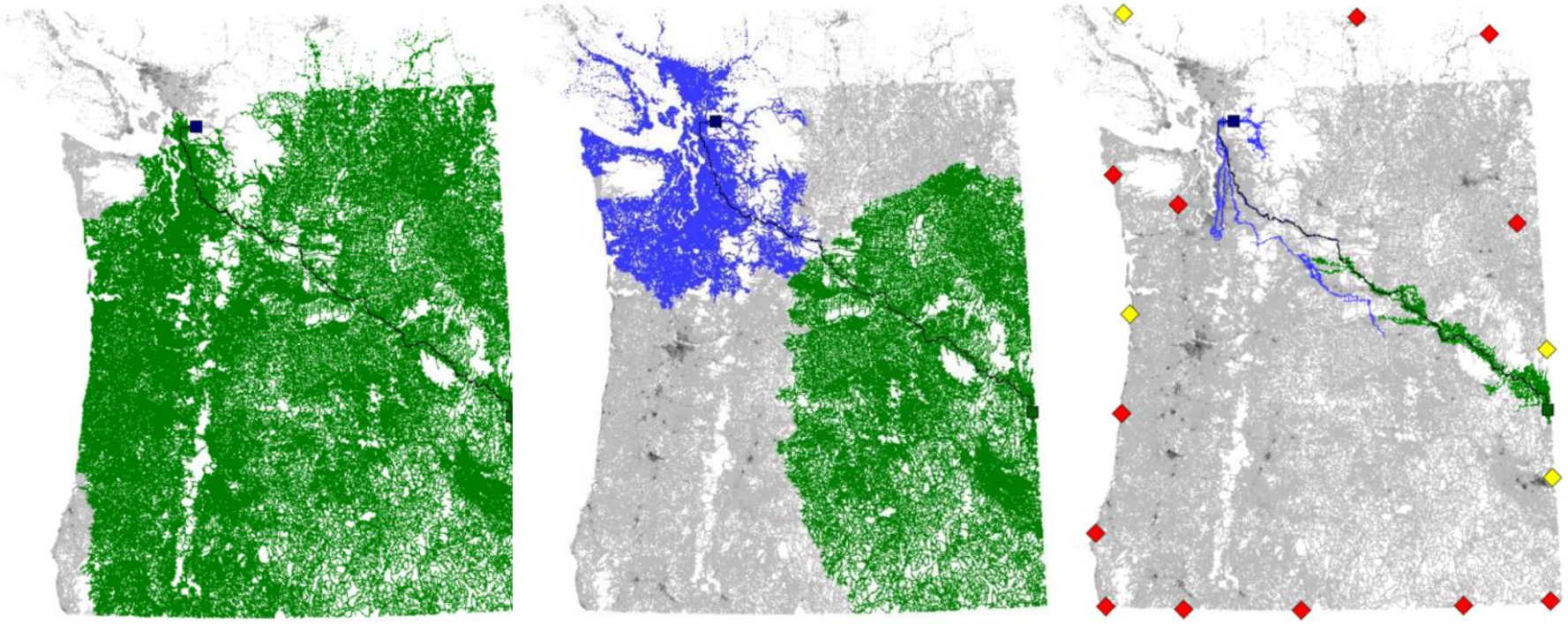


+ Goal Directed: Landmark

- Select a set of vertices $L = \{l_0, l_1, \ldots, l_k\}$
  - Precompute distance from l_i to every other vertices
- Lower-bounds
  - $d(u, v) \geq |d(l_i, u) - d(l_i, v)|$
  - Use the maximum as the estimation
    - $\pi(u, v) = max(|d(l_i, u) - d(l_i, v)|)$
  - A good landmark
    - Before u, like l_0
    - After v, like l_3
- Useful when there is no coordinate for Euclidean distance

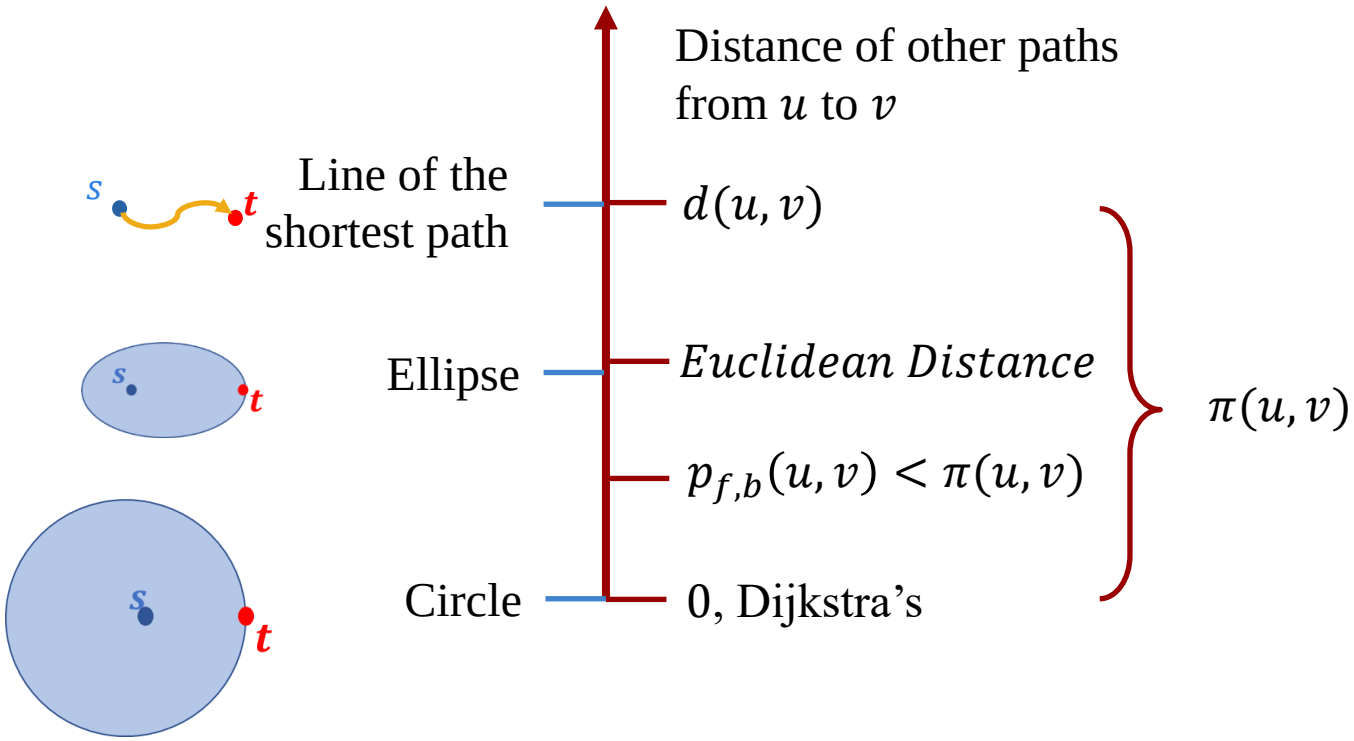


+ Goal Directed: Landmark



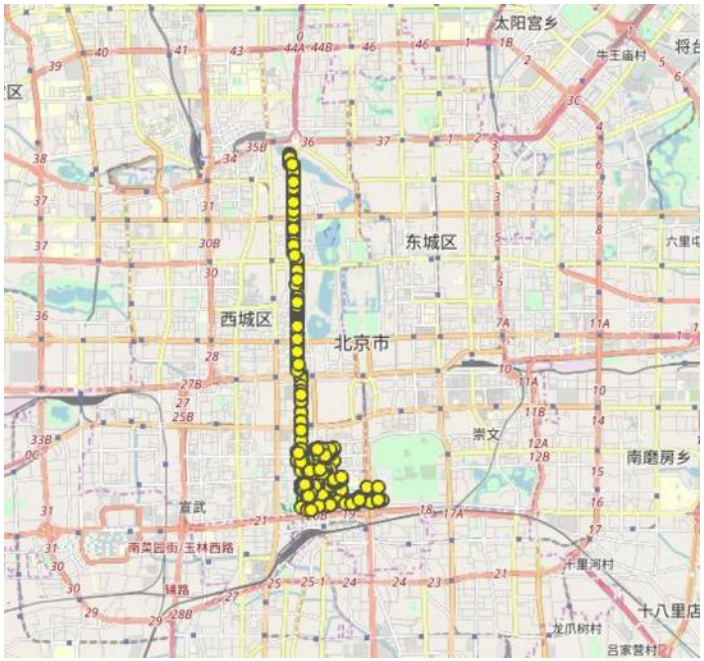
+ Goal Directed

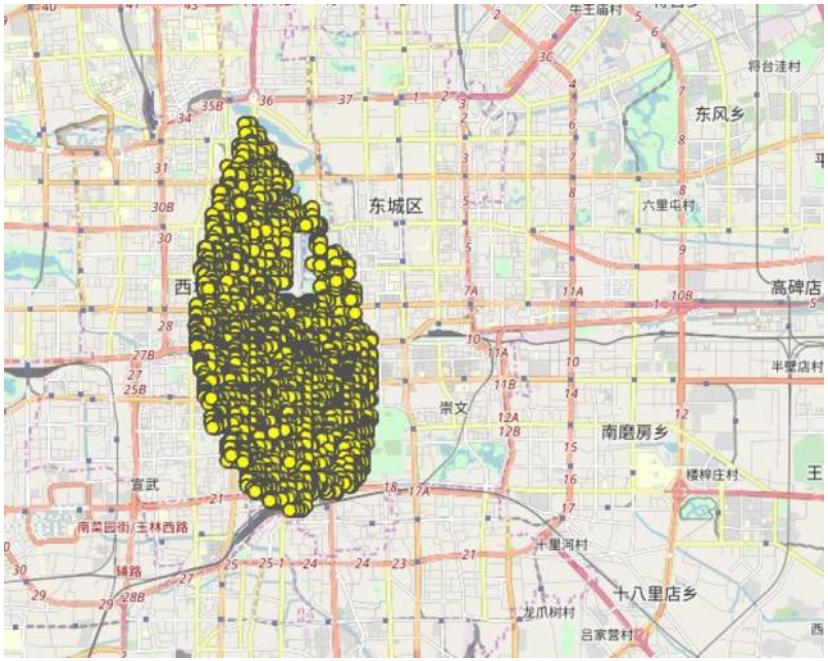
- When does estimation work?
  - Correctness guarantee: $\pi(u, v) \leq d(u, v)$
  - The closer $\pi(u, v)$ is to $d(u, v)$, the smaller search space



+ Goal Directed

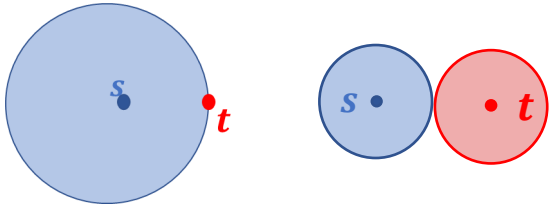
- When does estimation work?
  - Correctness guarantee: $\pi(u, v) \leq d(u, v)$
  - The closer $\pi(u, v)$ is to $d(u, v)$, the smaller search space



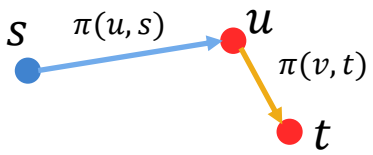


+ Goal Directed: Bi-A* Algorithm

- Dijkstra's → Bi-Dijkstra's
  - Forward search
  - Backward search
  - Weights are consistent



- A* → Bi-A*?
  - Forward: Estimation to t, $\pi(u,t)$
  - Backward: Estimation to $s, \pi(u,s)$
  - Weights are not consistent



+ Shortest Path Query

- All the previous mentioned methods
  - Search only use the original graph
  - Nearly no pre-processing
  - Building brick for the ground truth
  - But slow…

	Dijkstra	A*
2km	0.001096	0.00033454
5km	0.00402889	0.00077822
10km	0.0138615	0.00228061
15km	0.0276688	0.00450654
20km	0.0409668	0.00666603
25km	0.0544285	0.00929229
30km	0.0677643	0.0119041
35km	0.078577	0.014954
40km	0.0900467	0.0194614

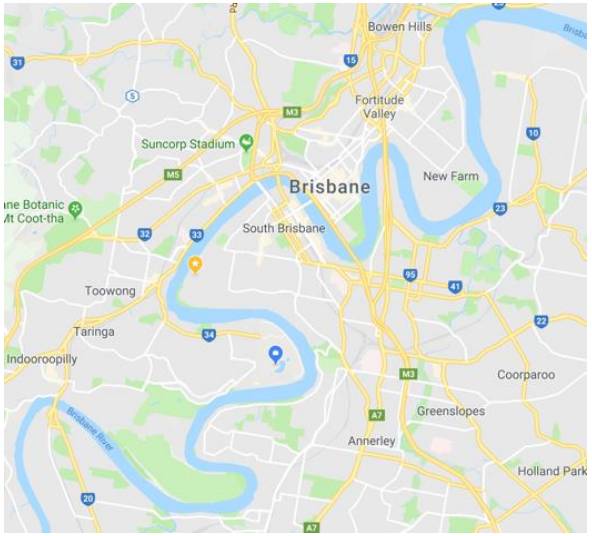
- Dataset
  - Beijing Road Network
  - 302,364 Vertices
  - 387,588 Edges

+ Running Time Example

- Query faster query answering with index
  - All pair distance?
  - Shortest region encoded in edge?
  - Max shortest distance for each node?
  - Connect to highways?
  - **Add shortcuts?**
  - **Cut distance?**

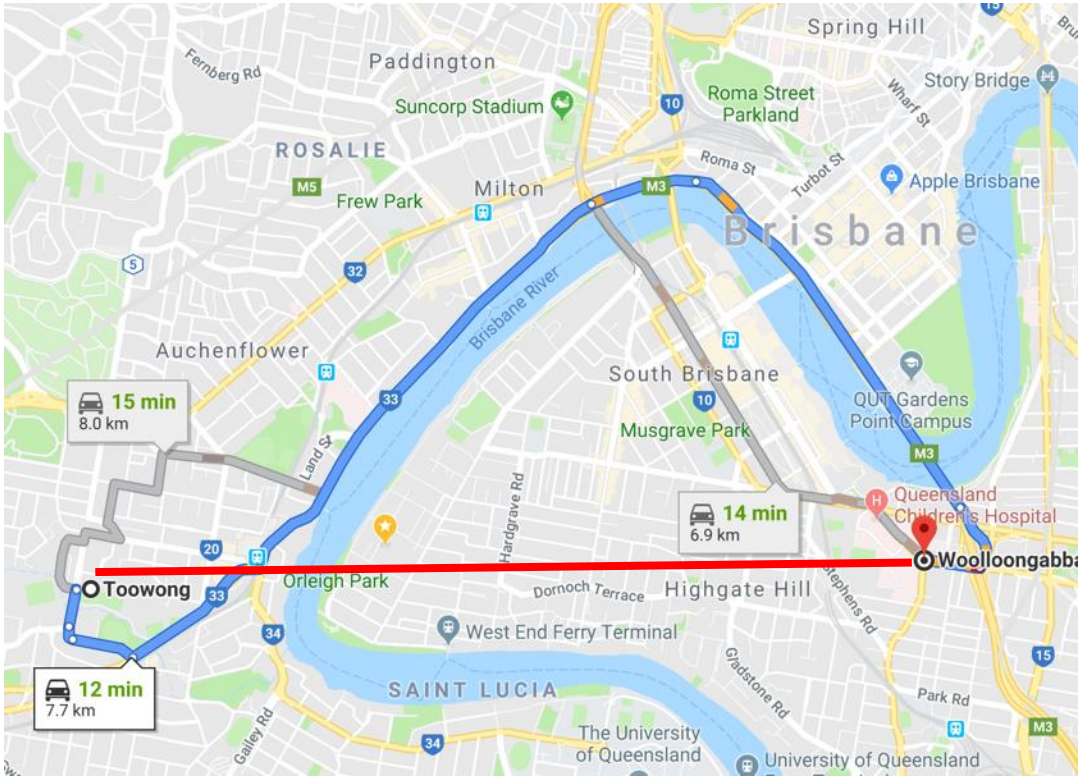
+ Preprocessing

- Let's come back to Brisbane map
  - From Toowong to Woolloongaba, go via City
  - From West End to Bowen Hills, go via City
  - From the Gap to Cannon Hill, go via City
  - …
    - City has several bridges
    - City is connected with M3, M5 and M7
    - City is in the center
    - City plays a more important role in the shortest paths



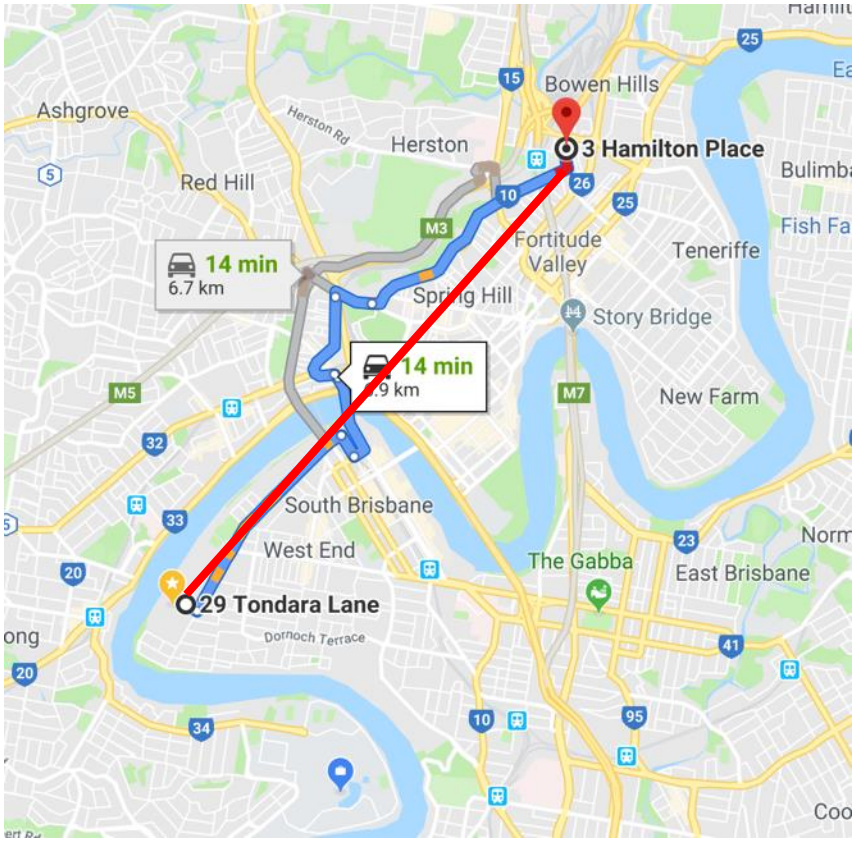
+ Preprocessing

- Let's come back to Brisbane map
  - From Toowong to Woolloongaba, go via City
    - Add a shortcut from Toowong to Woolloongaba?



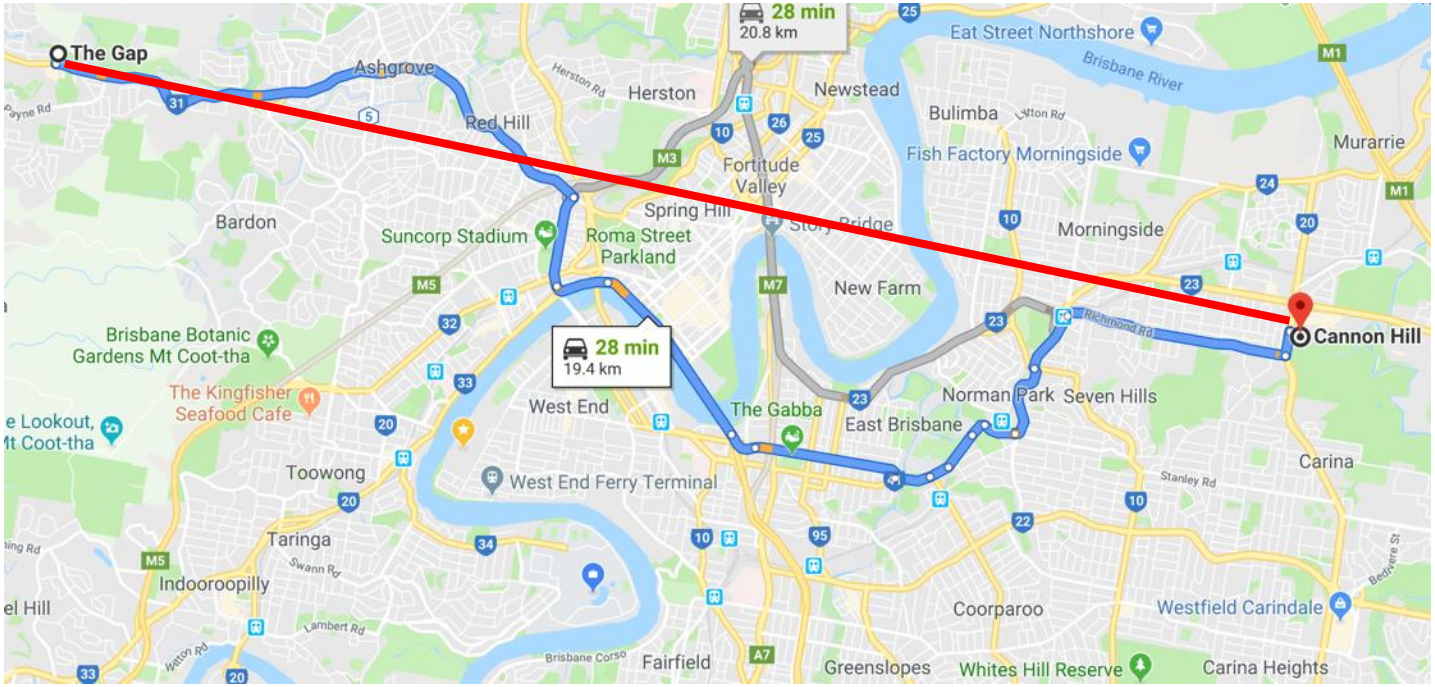
+ Preprocessing

- Let's come back to Brisbane map
  - From West End to Bowen Hills, go via City
    - Add a shortcut from West End to Bowen Hills?



+ Preprocessing

- Let's come back to Brisbane map
  - From the Gap to Cannon Hill, go via City
    - Add a shortcut from the Gap to Cannon Hill?



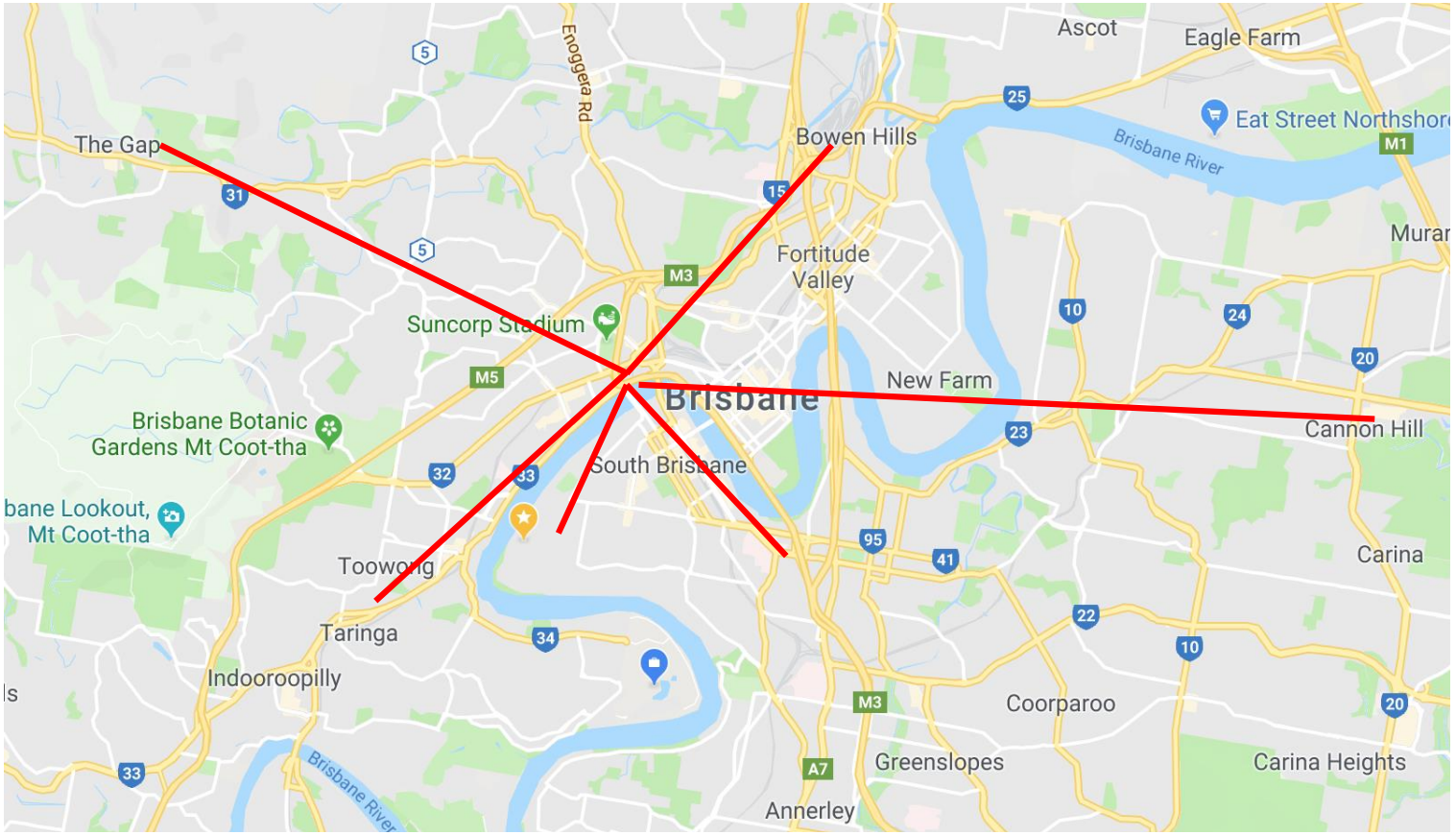
+ Preprocessing

- Let’s come back to Brisbane map
  - And add shortcuts for all the shortest paths that travel via City
  - When searching for a shortest path, we also use these helpful shortcuts without searching the City
    - Search space is reduced!
  - In a way, City is “contracted”

+ Preprocessing

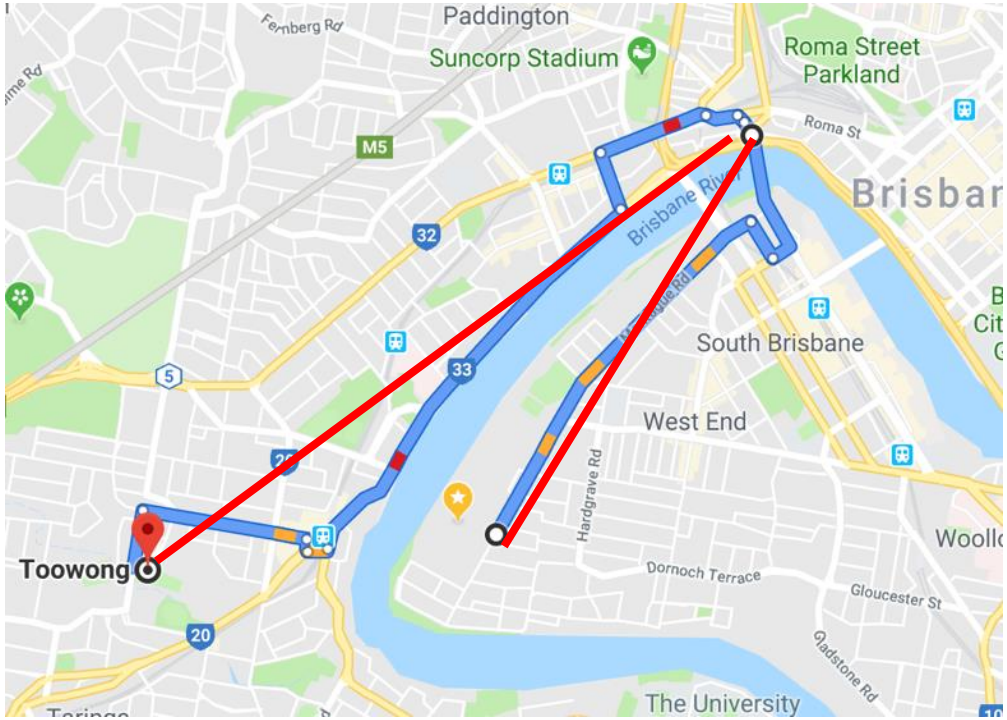
- Let's come back to Brisbane map
  - Or…
  - From Toowong to Woolloongaba, go via City
    1. Store the distance between Toowong and City
    2. Store the distance between Woolloongaba and City
  - From West End to Bowen Hills, go via City
    1. Store the distance between West End and City
    2. Store the distance between Bowen Hills and City
  - From the Gap to Cannon Hill, go via City
    1. Store the distance between the Gap and City
    2. Store the distance between Cannon Hill and City

+ Preprocessing



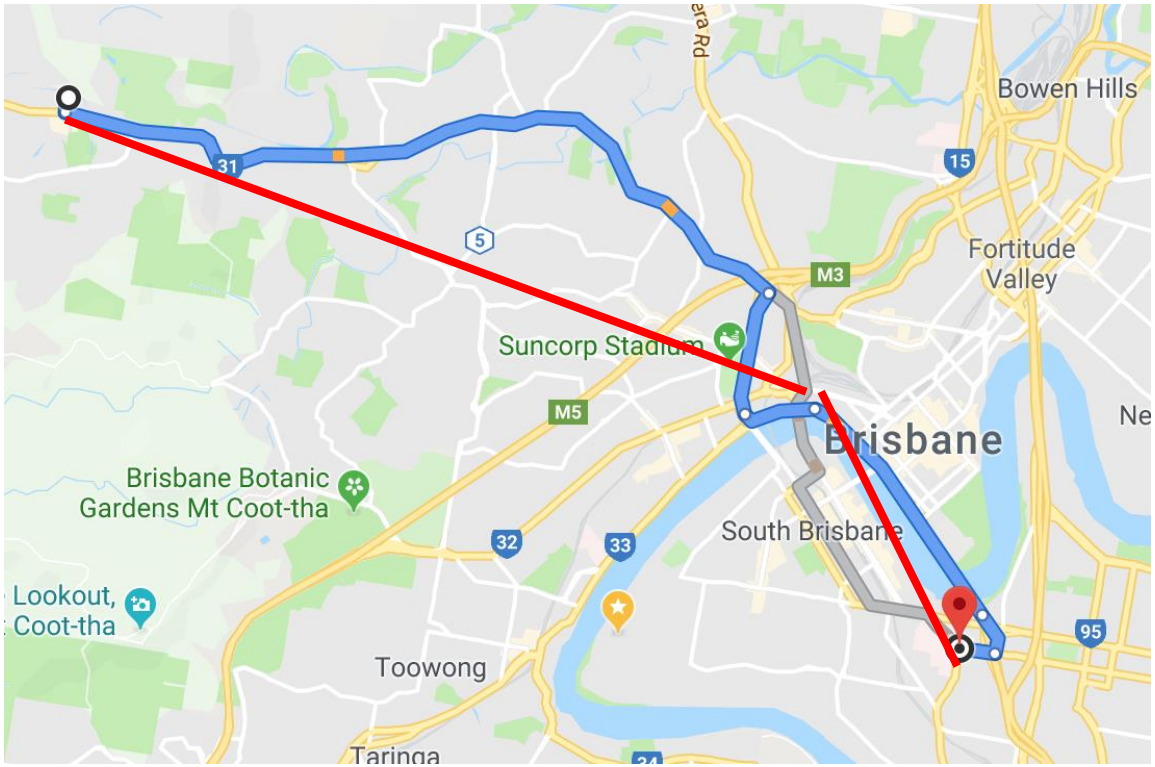
+ Preprocessing

- Let's come back to Brisbane map
  - When compute distance from West End to Toowong
    - Use the distance between West End and City
    - Use the distance between Toowong and City



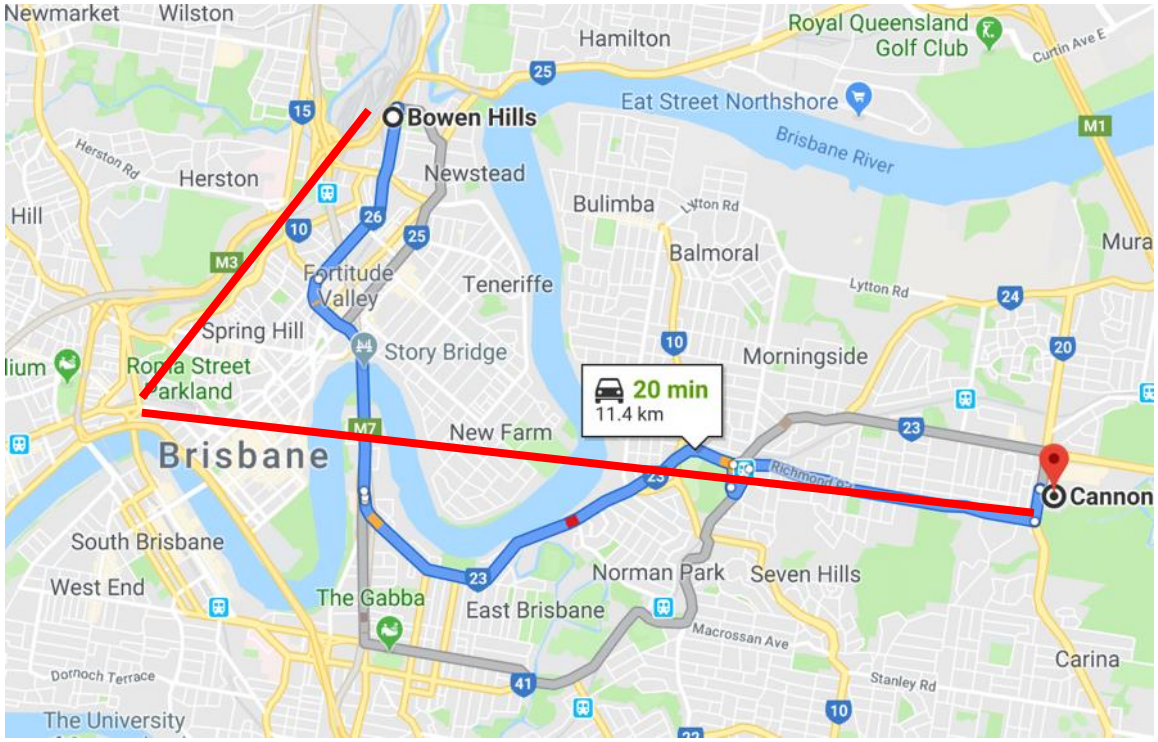
+ Preprocessing

- Let's come back to Brisbane map
  - When compute distance from the Gap to Woolloongaba
    - Use the distance between the Gap and City
    - Use the distance between Woolloongaba and City



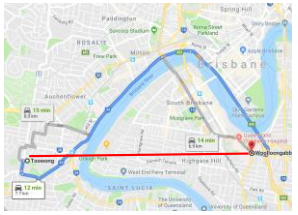
+ Preprocessing

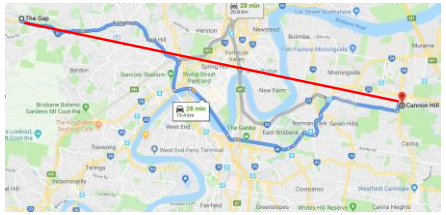
- Let's come back to Brisbane map
  - When compute distance from Cannon Hill to Bowen Hills
    - Use the distance between Cannon Hill and City ?
    - Use the distance between Bowen Hills and City ?



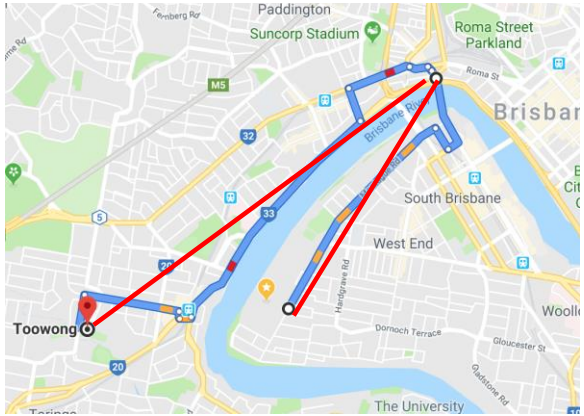
+ Preprocessing

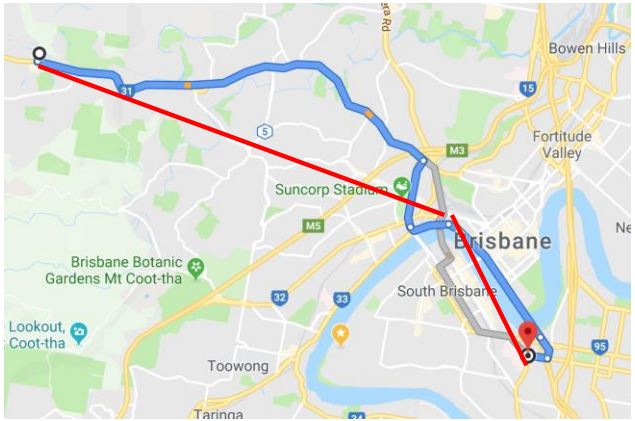
1. Search with Shortcuts
   - Contraction Hierarchy (CH)
     - Contract the more important City?
     - Binary Search:
       - Search towards City, shortcuts to City





2. Shortcut Table Lookup
   - 2 Hop Labeling
     - Toowong to City: 1 Hop, City to West End: 1 Hop
     - The Gap to City: 1 Hop, City to Woolloongaba: 1 Hop



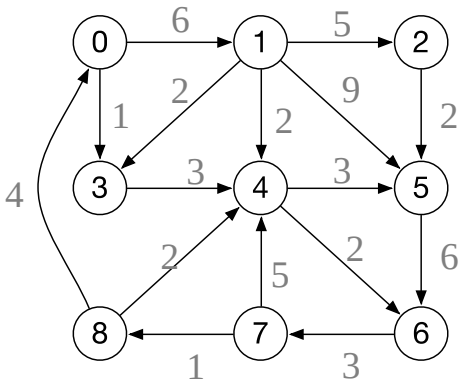


+ Preprocessing

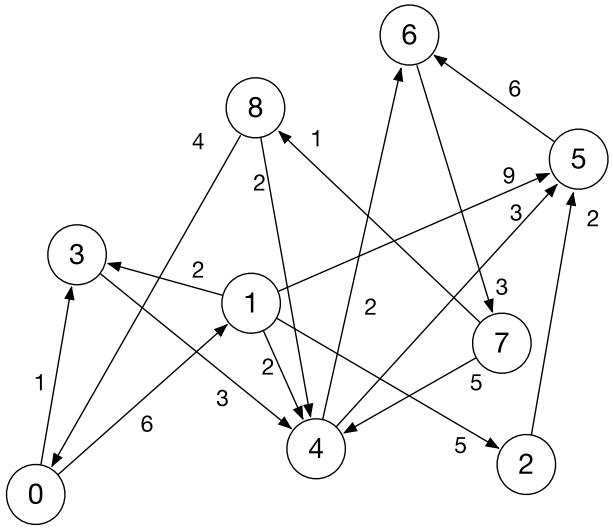
- Let’s come back to Brisbane map
  - Not just the City
    - Gateway Bridge
    - Albert Bridge connecting Indooroopilly and Chelmer
    - The crossing of Mains Rd and Kessels Rd
    - The T-Junction besides Toowong Village
    - The T-Junction besides UQ Logo
    - …
  - Some vertices are more important !

Contraction Hierarchy[1]

- Adding shortcuts to increase search speed
- Hierarchy
  - Visit vertices in an order
    - $v_0, v_2, v_4, v_7, v_1, v_3, v_5, v_8, v_6$



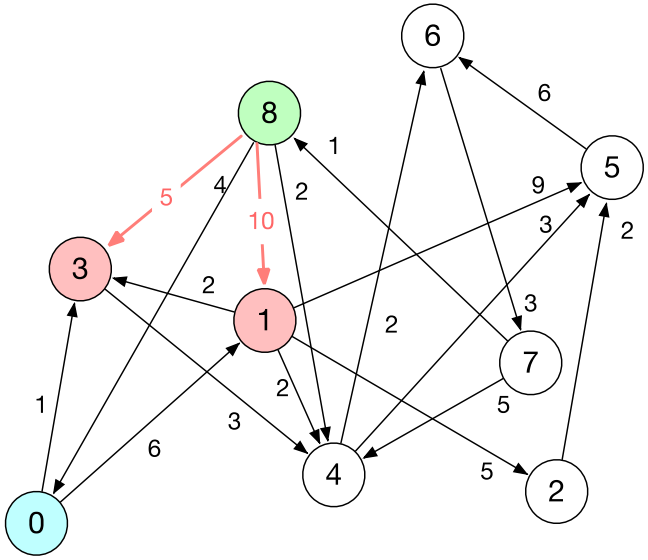
- Contraction
  - Add shortcuts when neighbors shortest paths pass through
    - $d(u \to v \to w) \leq d(u \to w)$ (ignoring v)
- Query
  - Forward Search Upward
  - Backward Search Upward

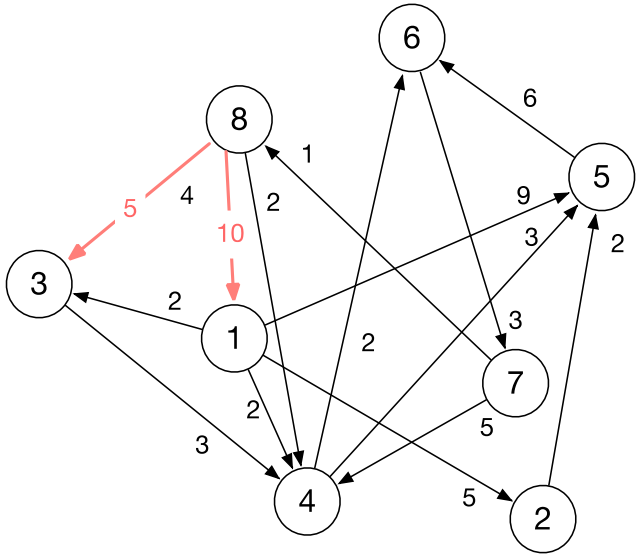


[1] Contraction Hierarchies Faster And Simpler Hierarchical Routing In Road Networks, International Workshop on Experimental and Efficient Algorithms, 2008

+ Contraction Hierarchy

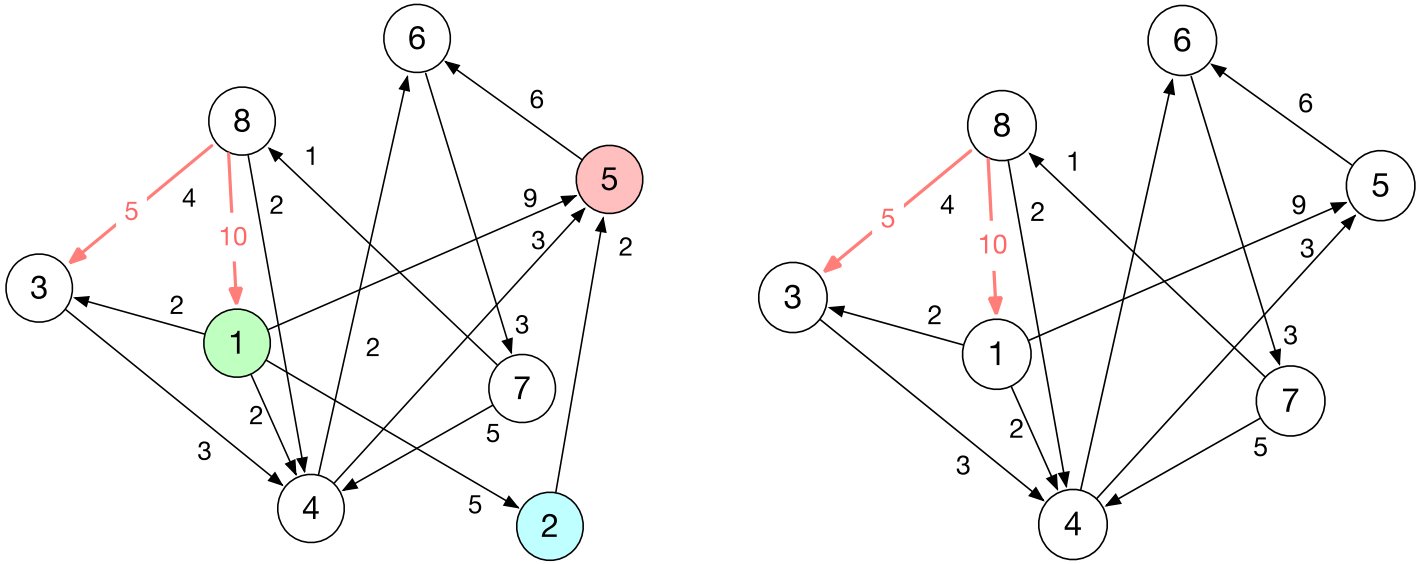
- Contract v_0
  - In-Neighbor: v_8
  - Out-Neighbor: v_1, v_3
  - Shortest Distance:
    - $d(v_8, v_1) = 10$, add shortcut
    - $d(v_8, v_3) = 5$, add shortcut





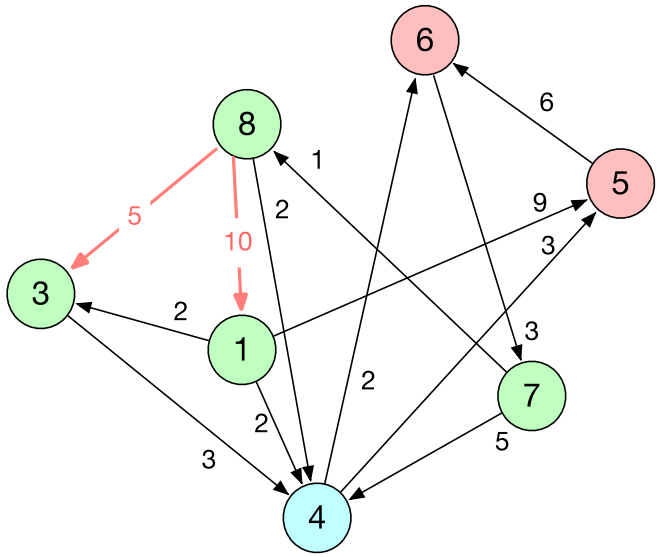
Contraction Hierarchy

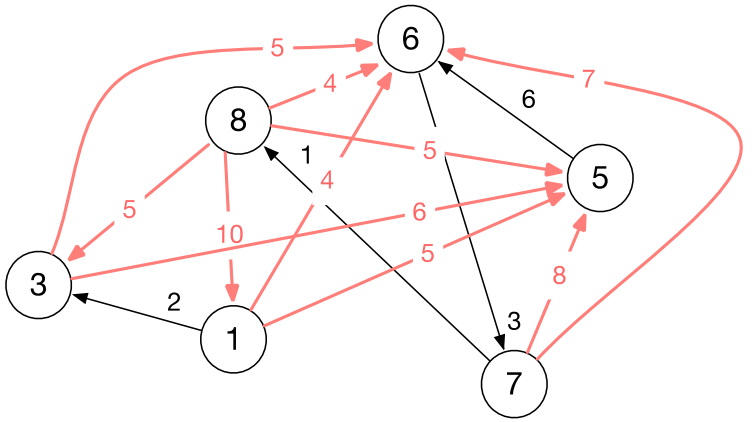
- Contract v_2
  - In-Neighbor: v_1
  - Out-Neighbor: v_5
  - Shortest Distance:
    - $d(v_1, v_5) = 5 < d(v_1, v_2) + d(v_2, v_5) = 7$



+ Contraction Hierarchy

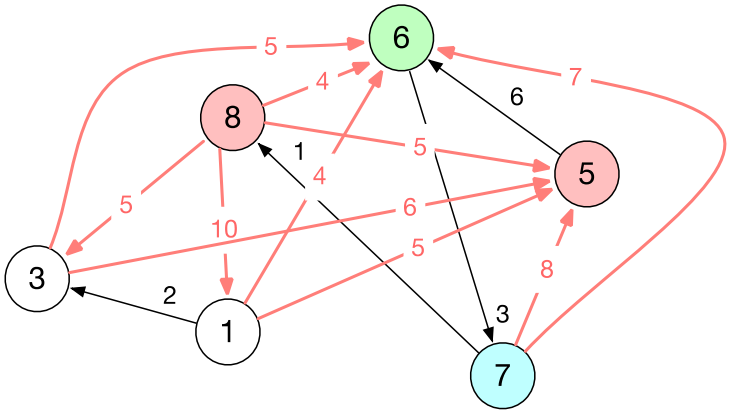
- Contract v_4
  - In-Neighbor: v_1, v_3, v_7, v_8
  - Out-Neighbor: v_5, v_6
  - Shortest Distance:
    - $d(v_1, v_5) = 5$, add shortcut
    - $d(v_1, v_6) = 4$, add shortcut
    - $d(v_3, v_5) = 6$, add shortcut
    - $d(v_3, v_6) = 4$, add shortcut
    - $d(v_7, v_5) = 8$, add shortcut
    - $d(v_7, v_6) = 7$, add shortcut
    - $d(v_8, v_5) = 5$, add shortcut
    - $d(v_8, v_6) = 4$, add shortcut

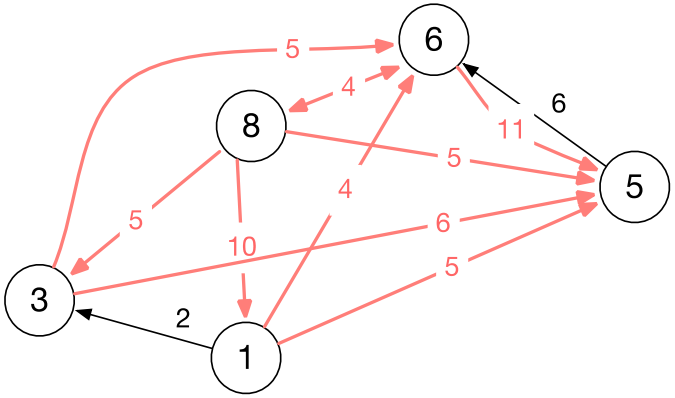




+ Contraction Hierarchy

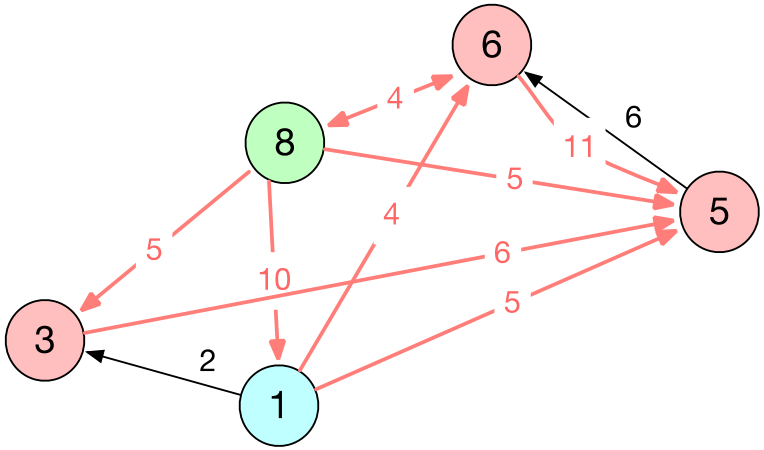
- Contract v_7
  - In-Neighbor: v_6
  - Out-Neighbor: v_5, v_8
  - Shortest Distance:
    - $d(v_6, v_5) = 11$, add shortcut
    - $d(v_6, v_8) = 4$, add shortcut

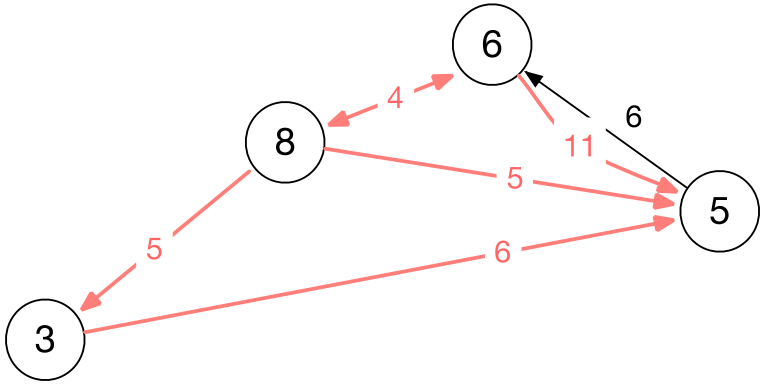




+ Contraction Hierarchy

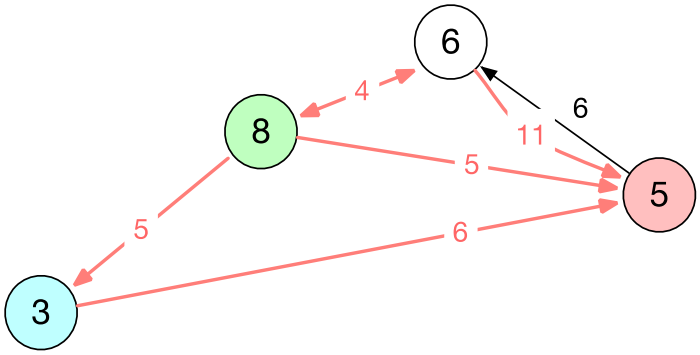
- Contract v_1
  - In-Neighbor: v_8
  - Out-Neighbor: v_3, v_5, v_6
  - Shortest Distance:
    - $d(v_8, v_3) = 5 < 12$
    - $d(v_8, v_5) = 5 < 15$
    - $d(v_8, v_6) = 4 < 14$

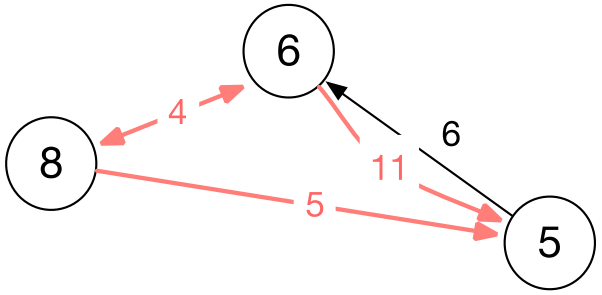




+ Contraction Hierarchy

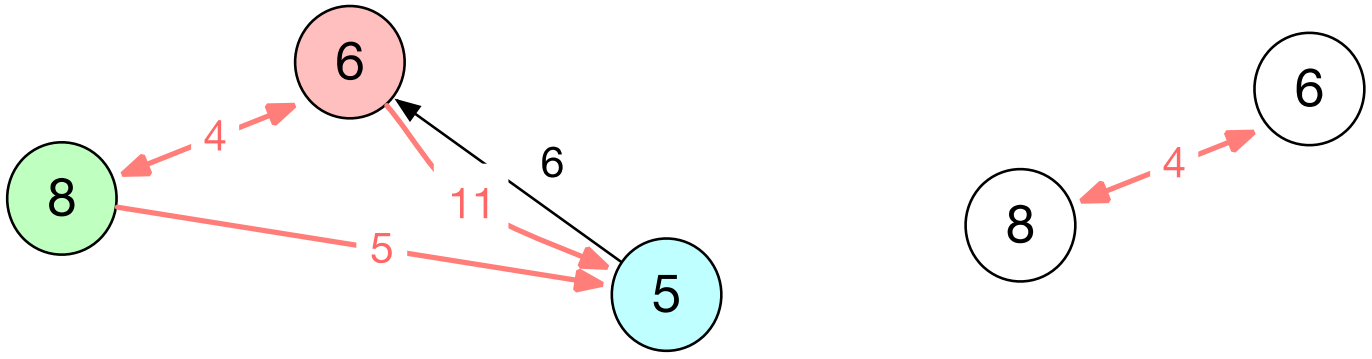
- Contract v_3
  - In-Neighbor: v_8
  - Out-Neighbor: v_3
  - Shortest Distance:
    - $d(v_8, v_5) = 5 < 11$



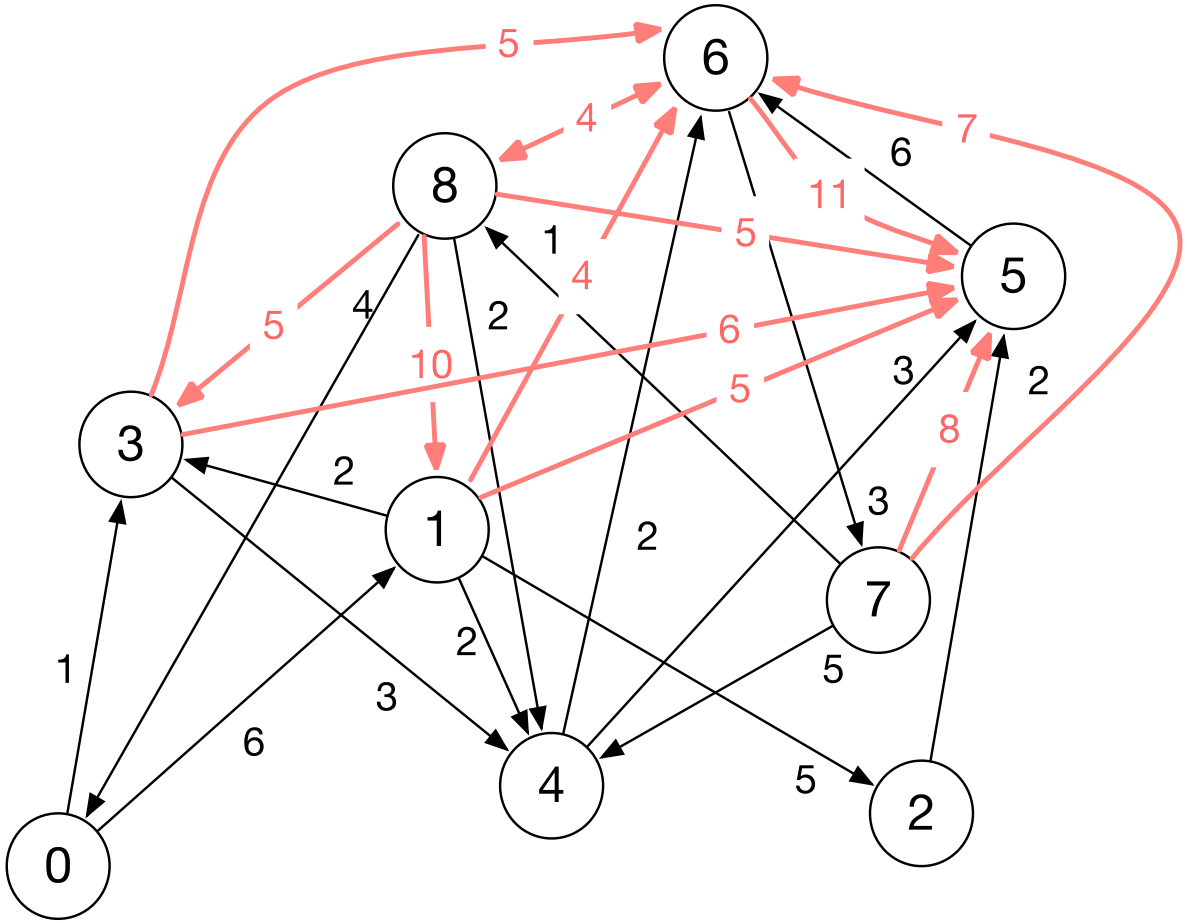


+ Contraction Hierarchy

- Contract v_5
  - In-Neighbor: v_6, v_8
  - Out-Neighbor: v_6
  - Shortest Distance:
    - $d(v_8, v_6) = 4 < 11$

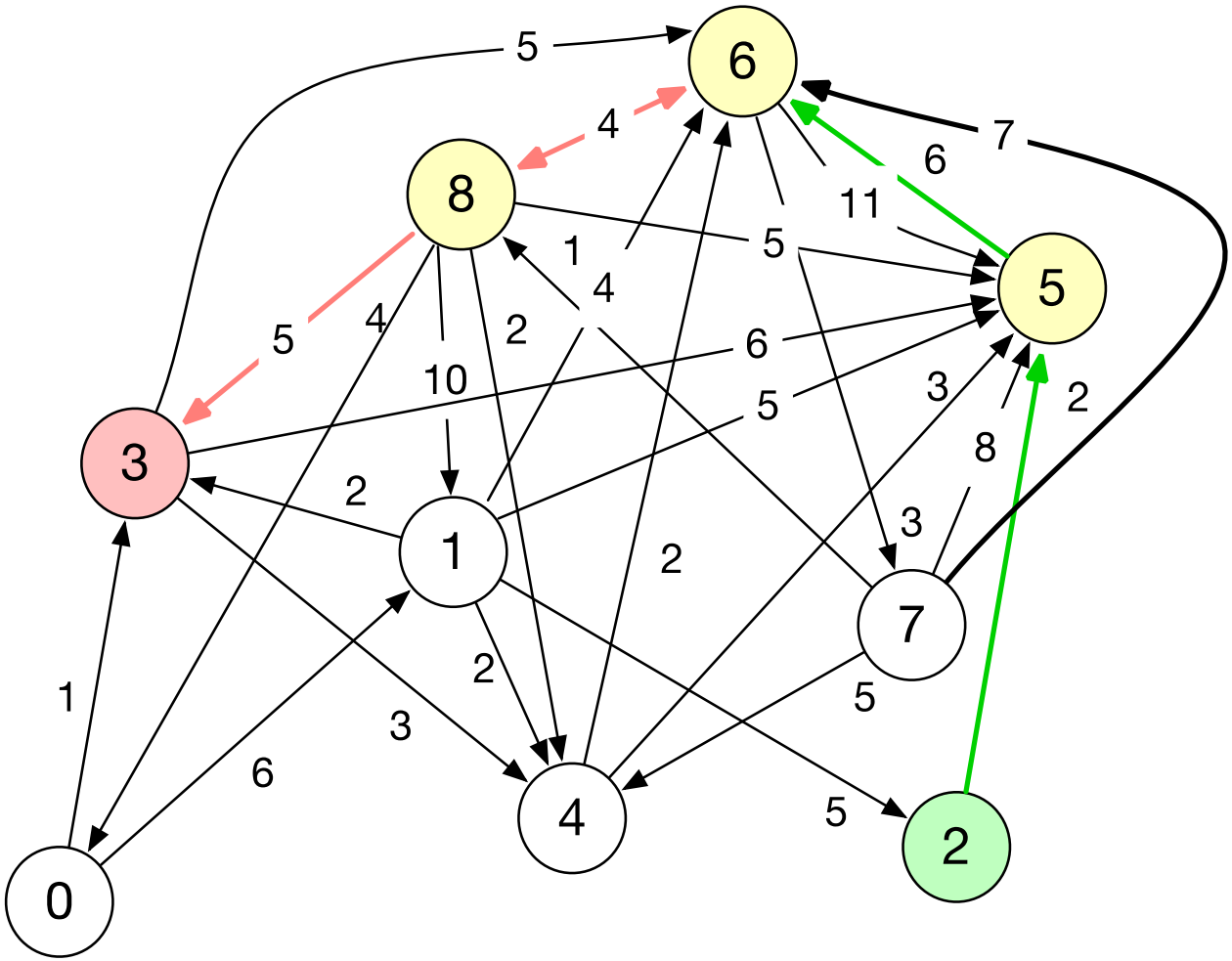


+ Contraction Hierarchy



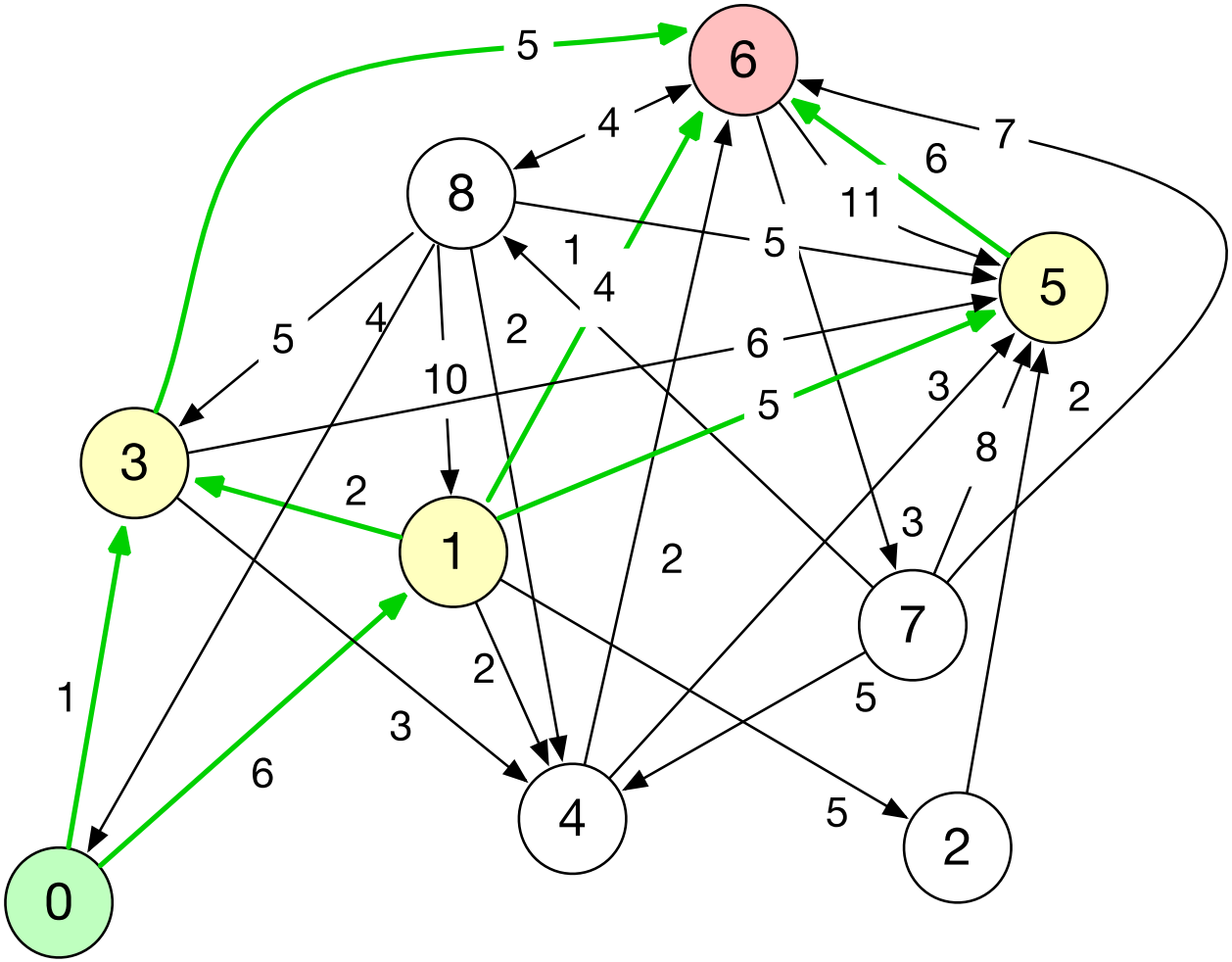
+ Contraction Hierarchy

- $v_2 \rightarrow v_3$



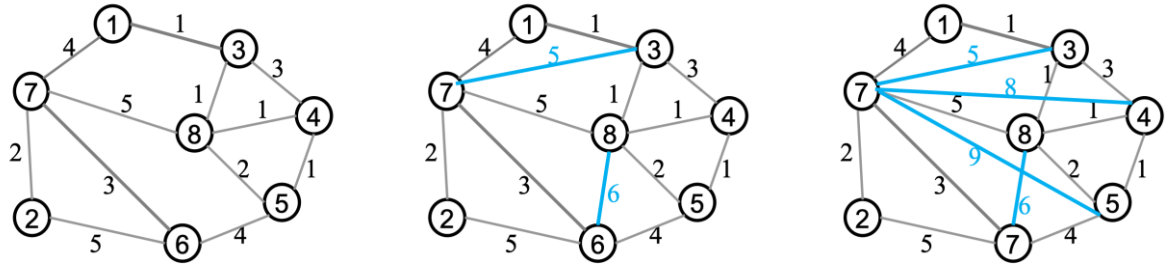
+ Contraction Hierarchy

- $v_0 \rightarrow v_6$



+ CH without Searching

- Slow because of many Dijkstra's search
  - Avoid searching?
  - For all $u, w \in v$'s neighbors
    - Add shortcut $d(u,v) + d(v,w)$
    - Or Update the existing one $\min(d(u,w), d(u,v) + d(v,w))$

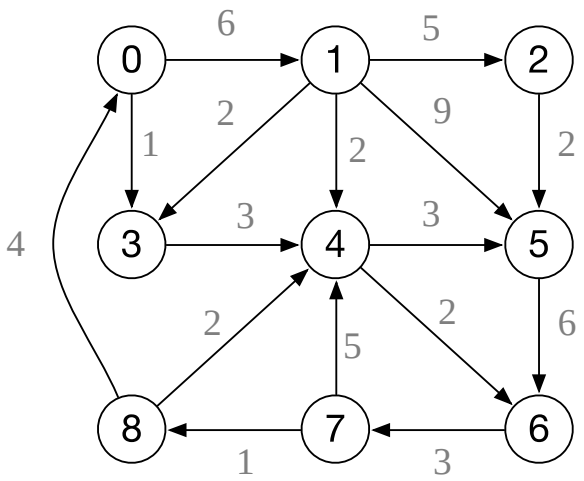


(a) Original Graph G (b) Contraction with pruning (c) Contraction without pruning

  - The densest possible case!
    - $O(|V|w^2)$
      - w is the "tree width", the largest number of neighbors during contraction
      - Not too big for road network
      - Explode when w is big

+ 2 Hop Labeling

- Hop Labels:
  - Out-Label: $L_{out}(u) = \{(h,\ \ d(u,h))\}$
  - In-Label: $\ L_{in}(u)\ \ = \{(h,\ \ d(h,u))\}$
  - Query: $d = \min\big(d(u,h) + d(h,v)\big)$
    - $h \in L_{out}(u) \cap L_{in}(v)$



Node	Out-Label	In-Label
0	(4,4) (1,6) (3,1)	(4,10) (5,14) (6,8)
1	(4,2)	(4,16)
2	(4,14) (1,22) (5,2)	(4,21) (1,5)
3	(4,3)	(4,11) (1,2) (5,15) (6,9)
4		
5	(4,12) (1,20)	(4,3)
6	(4,6) (1,14)	(4,2) (5,6)
7	(4,3) (1,11) (3,6) (0,5)	(4,5) (5,9) (6,3)
8	(4,2) (1,10) (3,5) (0,4)	(4,6) (5,10) (6,4) (7,1)

+ 2 Hop Labeling

- $v_0 \rightarrow v_6$

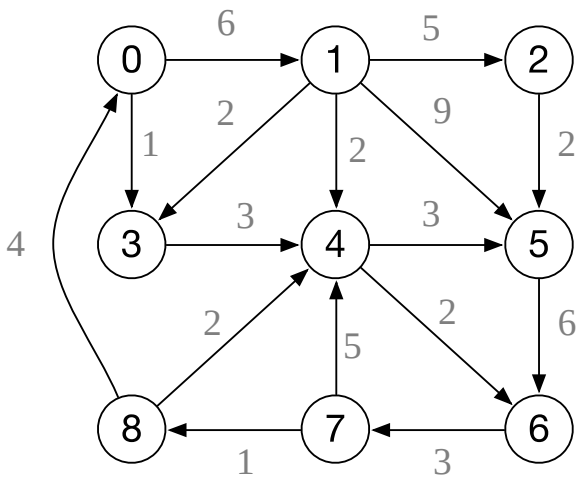
v_0 Out	v_6 In
$(v_4, 4)$	$(v_4, 2)$
$(v_1, 6)$	$(v_5, 6)$
$(v_3, 1)$	

  - d = 4 + 2 = 6

- $v_2 \rightarrow v_3$

v_2 Out	v_3 In
$(v_4, 14)$	$(v_4, 11)$
$(v_1, 22)$	$(v_1, 2)$
$(v_5, 2)$	$(v_5, 15)$
	$(v_6, 9)$

  - Hop 4: 14 + 11 = 25
  - Hop 1: 22 + 2 = 24
  - Hop 5: 2 + 15 = 17



Node	Out-Label	In-Label
0	(4,4) (1,6) (3,1)	(4,10) (5,14) (6,8)
1	(4,2)	(4,16)
2	(4,14) (1,22) (5,2)	(4,21) (1,5)
3	(4,3)	(4,11) (1,2) (5,15) (6,9)
4		
5	(4,12) (1,20)	(4,3)
6	(4,6) (1,14)	(4,2) (5,6)
7	(4,3) (1,11) (3,6) (0,5)	(4,5) (5,9) (6,3)
8	(4,2) (1,10) (3,5) (0,4)	(4,6) (5,10) (6,4) (7,1)

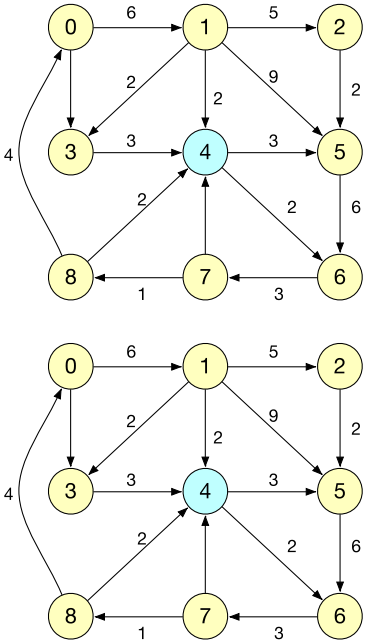
+ 2 Hop Labeling

- Construction
  - Optimal is NP-H[1]
  - Heuristic
    - Pruned Landmark Labeling [2]
    - Parallel Shortest path Labeling [7]
    - Independent Set [3]
    - Hop Doubling [4]
    - Hub Pushing [5]
    - Hierarchical 2-Hop [6]
    - …

[1] Reachability And Distance Queries Via 2-hop Labels, SIAM Journal on Computing 2002
[2] Fast Exact Shortest-path Distance Queries On Large Networks By Pruned Landmark Labeling, SIGMOD 2013
[3] IS-LABEL: An Independent-Set based Labeling Scheme for Point-to-Point Distance Querying, VLDB 2013
[4] Hop Doubling Label Indexing for Point-to-Point Distance Querying, VLDB 2014
[5] An Experimental Study on Hub Labeling based Shortest Path Algorithms, VLDB 2017
[6] When Hierarchy Meets 2-Hop-Labeling: Efficient Shortest Distance Queries on Road Networks, SIGMOD 2018
[7] Scaling Distance Labeling on Small-World Networks, SIGMOD2019

Pruned Landmark Labeling PLL

- Forward Search from v_i
  - Dijkstra's Search through the out-edges
  - Visit v_j, create **in-label** $(v_i, d(v_i, v_j))$of v_j
- Backward Search from v_i
  - Dijkstra's Search through the in-edges
  - Visit v_j, create **out-label** $(v_i, d(v_j, v_i))$ of v_j
- All Pair Result
  - Definitely correct but slow and huge

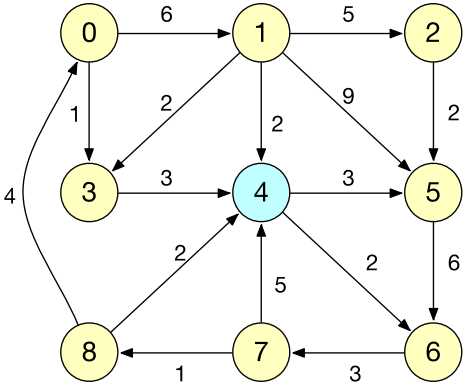


- Pruned Search
  - If $d(v_i, v_j)$ is not smaller than the query result of the existing labels
    - Stop searching from v_j
    - Do not visit v_j neighbors

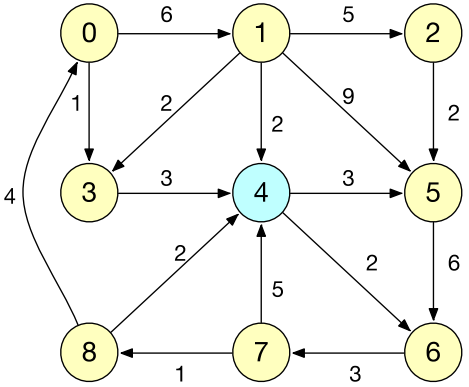
+ PLL Example

Source vertex Skipped vertex Visited vertex Unvisited vertex

Forward Search



Backward Search



Out Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,4	4,2	4,14	4,3		4,12	4,6	4,3	4,2

In Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,10	4,16	4,21	4,11		4,3	4,2	4,5	4,6

+ PLL Example

Source vertex Skipped vertex Visited vertex Unvisited vertex

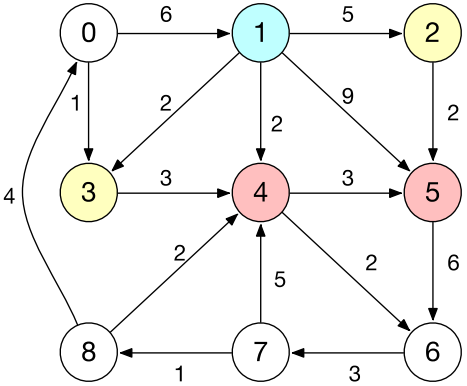
$v_1 \rightarrow v_4$

v_1 Out	v_4 In
$(v_4, 2)$	

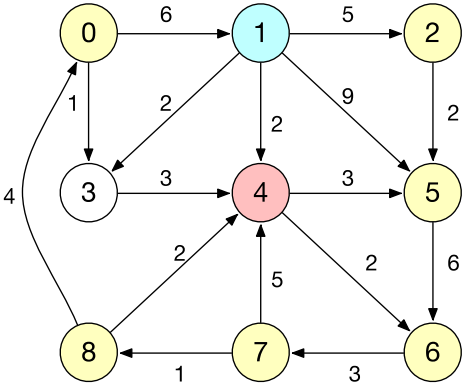
$v_1 \rightarrow v_5$

v_1 Out	v_5 In
$(v_4, 2)$	$(v_4, 3)$

Forward Search



Backward Search



$v_4 \rightarrow v_1$

v_4 Out	v_1 In
	$(v_4, 10)$

Out Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,4 1,6	4,2	4,14 1,22	4,3		4,12 1,20	4,6 1,14	4,3 1,11	4,2 1,10

In Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,10	4,16	4,21 1,5	4,11 1,2		4,3	4,2	4,5	4,6

+ PLL Example

Source vertex Skipped vertex Visited vertex Unvisited vertex

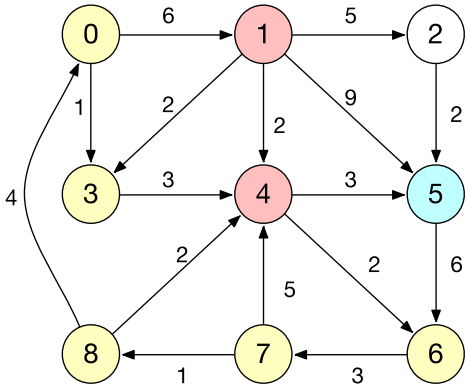
$v_5 \rightarrow v_1$

v_5 Out	v_1 In
$(v_4, 12)$	$(v_4, 16)$
$(v_1, 20)$	

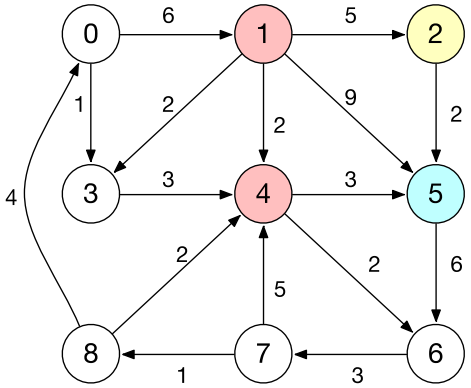
$v_5 \rightarrow v_4$

v_5 Out	v_4 In
$(v_4, 12)$	

Forward Search



Backward Search



$v_1 \rightarrow v_5$

v_1 Out	v_5 In
$(v_4, 2)$	$(v_4, 3)$

$v_4 \rightarrow v_5$

v_4 Out	v_5 In
	$(v_4, 3)$

Out Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,4 1,6	4,2	4,14 1,22 5,2	4,3		4,12 1,20	4,6 1,14	4,3 1,11	4,2 1,10

In Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,10 5,14	4,16	4,21 1,5	4,11 1,2 5,15		4,3	4,2 5,6	4,5 5,9	4,6 5,10

+ PLL Example

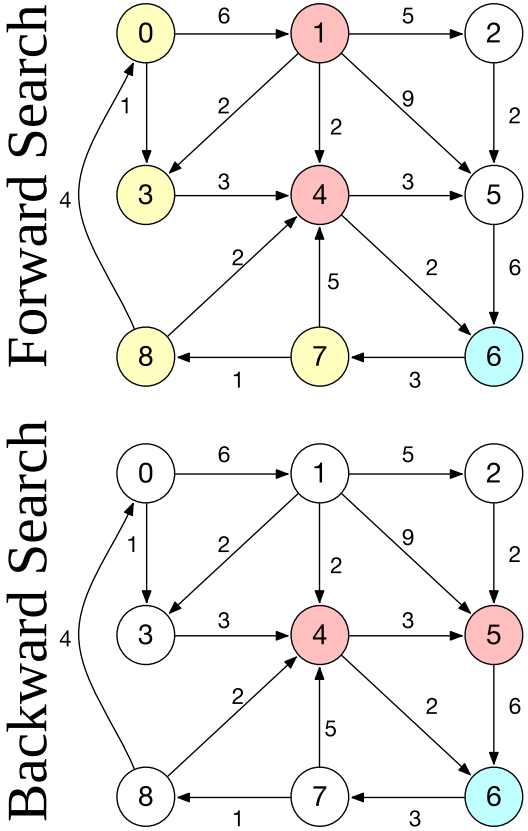
Source vertex Skipped vertex Visited vertex Unvisited vertex

$v_6 \rightarrow v_1$

v_6 Out	v_1 In
$(v_4, 6)$	$(v_4, 16)$
$(v_1, 14)$	

$v_6 \rightarrow v_4$

v_6 Out	v_4 In
$(v_4, 6)$	
$(v_1, 14)$	



$v_5 \rightarrow v_6$

v_5 Out	v_6 In
$(v_4, 12)$	$(v_4, 2)$
$(v_1, 20)$	$(v_5, 6)$

$v_4 \rightarrow v_6$

v_4 Out	v_6 In
	$(v_4, 2)$
	$(v_5, 6)$

Out Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,4 1,6	4,2	4,14 1,22 5,2	4,3		4,12 1,20	4,6 1,14	4,3 1,11	4,2 1,10

In Labels

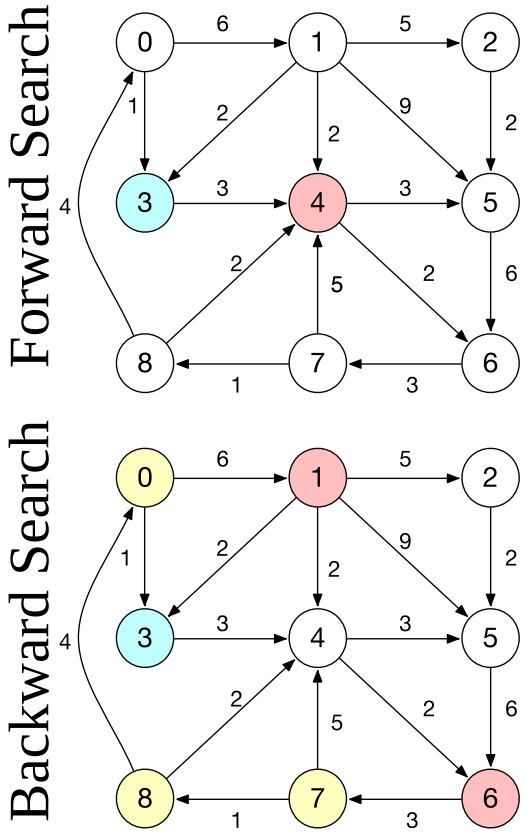
v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,10 5,14 6,8	4,16	4,21 1,5	4,11 1,2 5,15 6,9		4,3	4,2 5,6	4,5 5,9 6,3	4,6 5,10 6,4

+ PLL Example

Source vertex Skipped vertex Visited vertex Unvisited vertex

$v_3 \rightarrow v_4$

v_3 Out	v_1 In
$(v_4, 3)$	



$v_1 \rightarrow v_3$

v_1 Out	v_3 In
$(v_4, 2)$	$(v_4, 11)$
	$(v_1, 2)$
	$(v_5, 15)$
	$(v_6, 9)$

$v_6 \rightarrow v_3$

v_6 Out	v_3 In
$(v_4, 2)$	$(v_4, 11)$
$(v_5, 6)$	$(v_1, 2)$
	$(v_5, 15)$
	$(v_6, 9)$

Out Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,4 1,6 3,1	4,2	4,14 1,22 5,2	4,3		4,12 1,20	4,6 1,14	4,3 1,11 3,6	4,2 1,10 3,5

In Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,10 5,14 6,8	4,16	4,21 1,5	4,11 1,2 5,15 6,9		4,3	4,2 5,6	4,5 5,9 6,3	4,6 5,10 6,4

+ PLL Example

Source vertex Skipped vertex Visited vertex Unvisited vertex

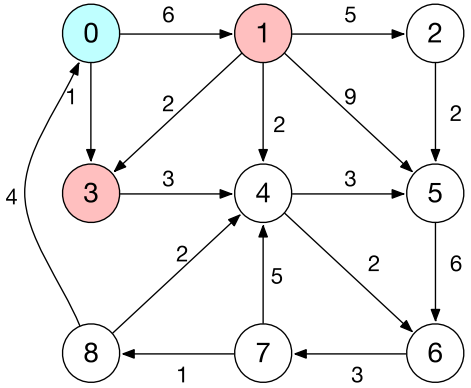
$v_0 \rightarrow v_1$

v_0 Out	v_1 In
$(v_4, 4)$	$(v_4, 16)$
$(v_1, 6)$	
$(v_3, 1)$	

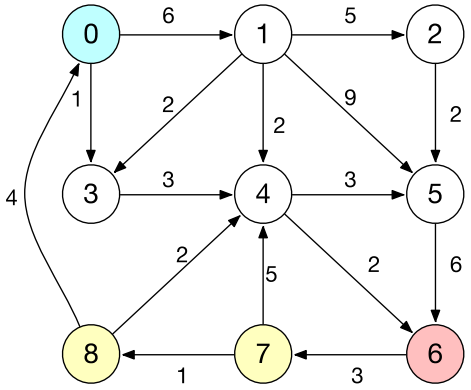
$v_0 \rightarrow v_3$

v_0 Out	v_3 In
$(v_4, 4)$	$(v_4, 11)$
$(v_1, 6)$	$(v_1, 2)$
$(v_3, 1)$	$(v_5, 15)$
	$(v_6, 9)$

Forward Search



Backward Search



$v_6 \rightarrow v_0$

v_6 Out	v_0 In
$(v_4, 6)$	$(v_4, 10)$
$(v_1, 14)$	$(v_5, 14)$
	$(v_6, 8)$

Out Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,4 1,6 3,1	4,2	4,14 1,22 5,2	4,3		4,12 1,20	4,6 1,14	4,3 1,11 3,6 0,5	4,2 1,10 3,5 0,4

In Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,10 5,14 6,8	4,16	4,21 1,5	4,11 1,2 5,15 6,9		4,3	4,2 5,6	4,5 5,9 6,3	4,6 5,10 6,4

+ PLL Example

Source vertex Skipped vertex Visited vertex Unvisited vertex

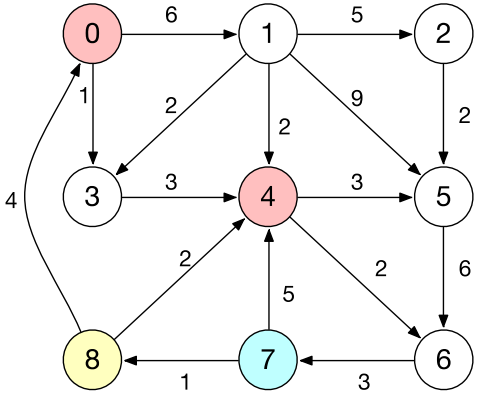
$v_7 \rightarrow v_0$

v_7 Out	v_0 In
$(v_4, 3)$	$(v_4, 10)$
$(v_1, 11)$	$(v_5, 14)$
$(v_3, 6)$	$(v_6, 8)$
$(v_0, 5)$	

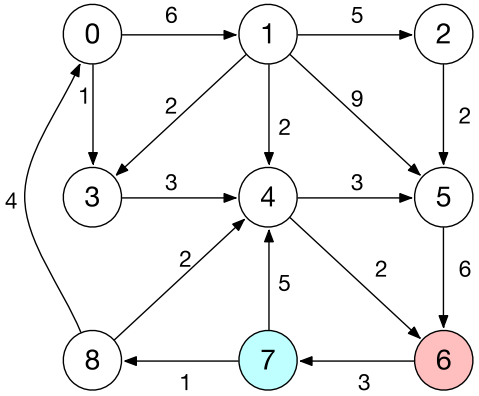
$v_0 \rightarrow v_3$

v_0 Out	v_4 In
$(v_4, 3)$	
$(v_1, 11)$	
$(v_3, 6)$	
$(v_0, 5)$	

Forward Search



Backward Search



$v_6 \rightarrow v_7$

v_6 Out	v_7 In
$(v_4, 6)$	$(v_4, 5)$
$(v_1, 14)$	$(v_5, 9)$
	$(v_6, 3)$

Out Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,4 1,6 3,1	4,2	4,14 1,22 5,2	4,3		4,12 1,20	4,6 1,14	4,3 1,11 3,6 0,5	4,2 1,10 3,5 0,4

In Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,10 5,14 6,8	4,16	4,21 1,5	4,11 1,2 5,15 6,9		4,3	4,2 5,6	4,5 5,9 6,3	4,6 5,10 6,4 7,1

+ PLL Example

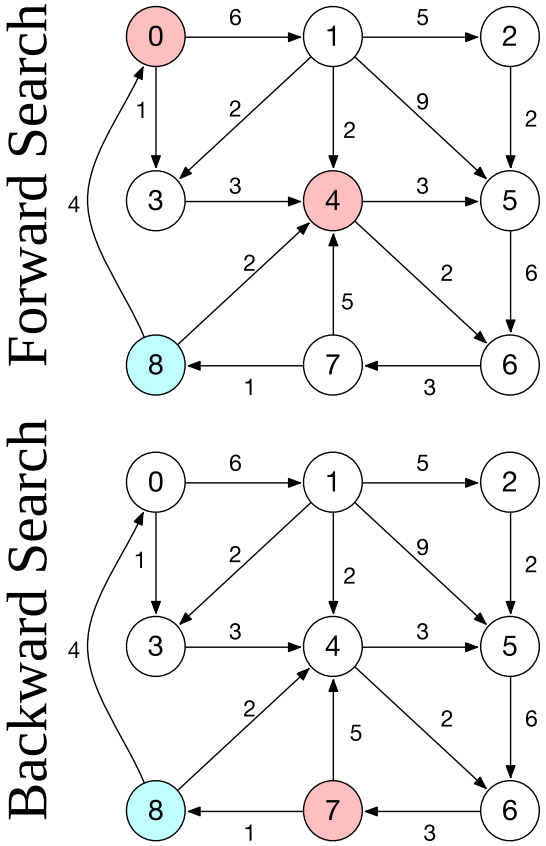
Source vertex Skipped vertex Visited vertex Unvisited vertex

$v_8 \rightarrow v_0$

v_8 Out	v_0 In
$(v_4, 2)$	$(v_4, 10)$
$(v_1, 10)$	$(v_5, 14)$
$(v_3, 5)$	$(v_6, 8)$
$(v_0, 4)$	

$v_0 \rightarrow v_4$

v_4 Out	v_4 In
$(v_4, 2)$	
$(v_1, 10)$	
$(v_3, 5)$	
$(v_0, 4)$	



$v_7 \rightarrow v_8$

v_7 Out	v_8 In
$(v_4, 3)$	$(v_4, 6)$
$(v_1, 11)$	$(v_5, 10)$
$(v_3, 6)$	$(v_6, 4)$
$(v_0, 5)$	$(v_7, 1)$

Out Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,4 1,6 3,1	4,2	4,14 1,22 5,2	4,3		4,12 1,20	4,6 1,14	4,3 1,11 3,6 0,5	4,2 1,10 3,5 0,4

In Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,10 5,14 6,8	4,16	4,21 1,5	4,11 1,2 5,15 6,9		4,3	4,2 5,6	4,5 5,9 6,3	4,6 5,10 6,4 7,1

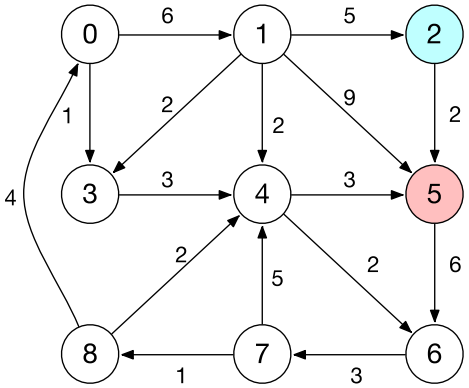
+ PLL Example

Source vertex Skipped vertex Visited vertex Unvisited vertex

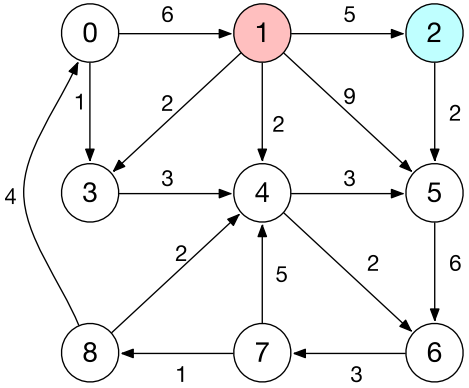
$v_8 \rightarrow v_0$

v_2 Out	v_5 In
$(v_4, 14)$	$(v_4, 3)$
$(v_1, 22)$	
$(v_5, 2)$	

Forward Search



Backward Search



$v_1 \rightarrow v_2$

v_1 Out	v_2 In
$(v_4, 2)$	$(v_4, 14)$
	$(v_1, 22)$
	$(v_5, 2)$

Out Labels

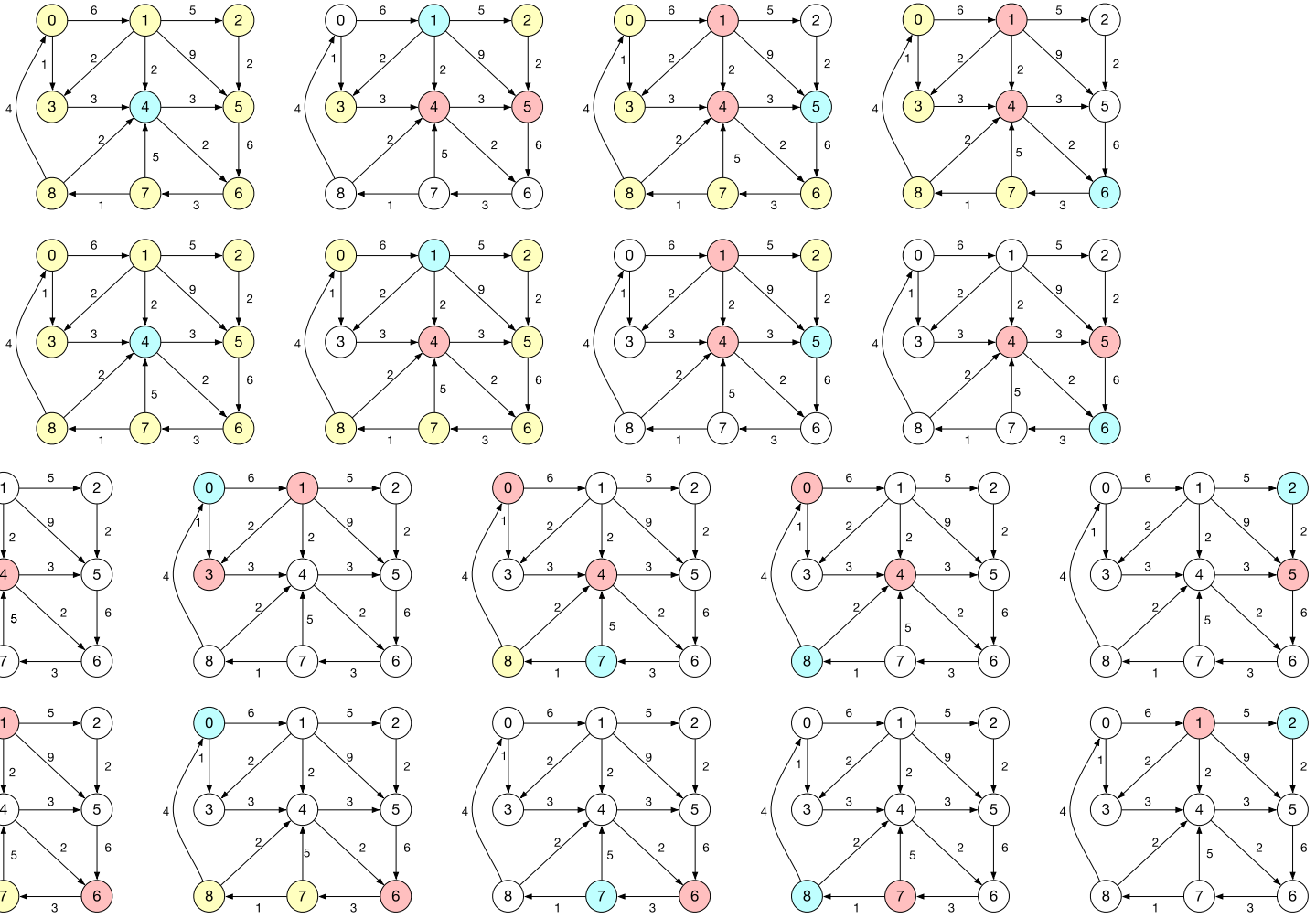
v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,4 1,6 3,1	4,2	4,14 1,22 5,2	4,3		4,12 1,20	4,6 1,14	4,3 1,11 3,6 0,5	4,2 1,10 3,5 0,4

In Labels

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
4,10 5,14 6,8	4,16	4,21 1,5	4,11 1,2 5,15 6,9		4,3	4,2 5,6	4,5 5,9 6,3	4,6 5,10 6,4 7,1

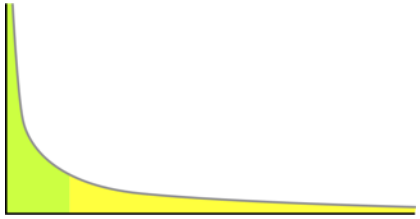
+ PLL Example

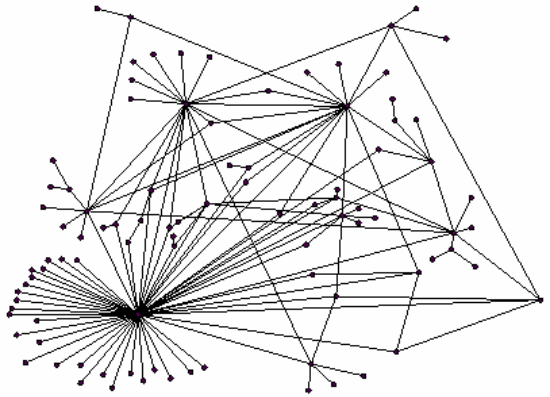
Source vertex Skipped vertex Visited vertex Unvisited vertex



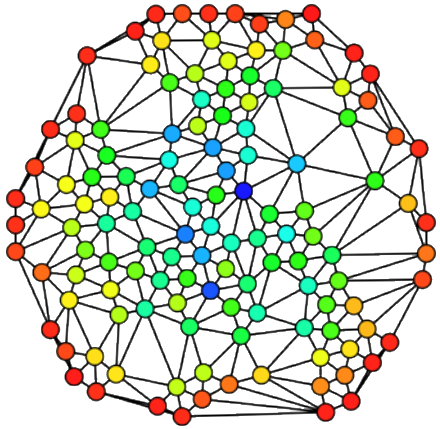
+ Vertex Importance

- How to determine the importance?
  - Scale-free graph
    - Degree distribution is power-law
    - Use degree as order
      - Social Network
      - WWW
      - Airline Network



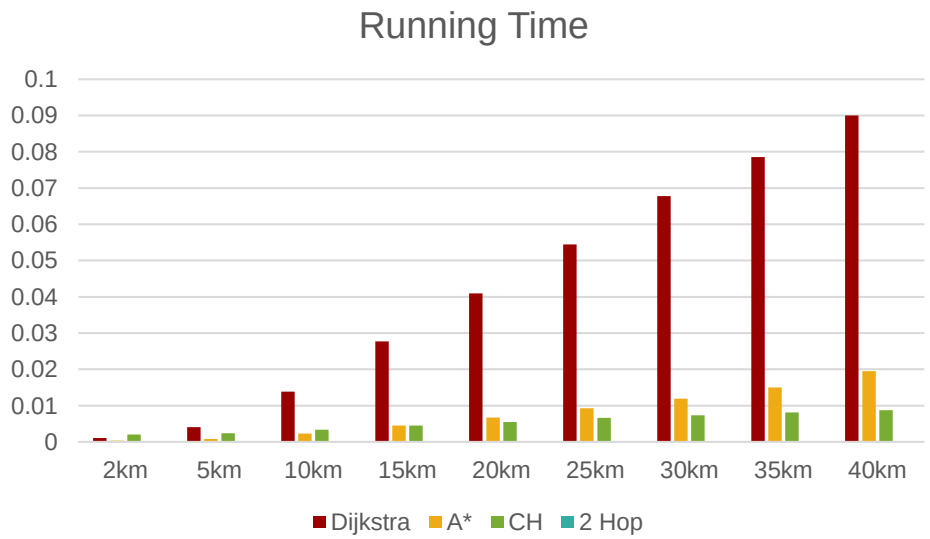


  - Road network
    - Usually degree < 5
    - Betweenness Centrality
      - The number of shortest paths passing through v
      - City has higher Betweenness Centrality
    - Minimum Degree Elimination



+ Running Time Example

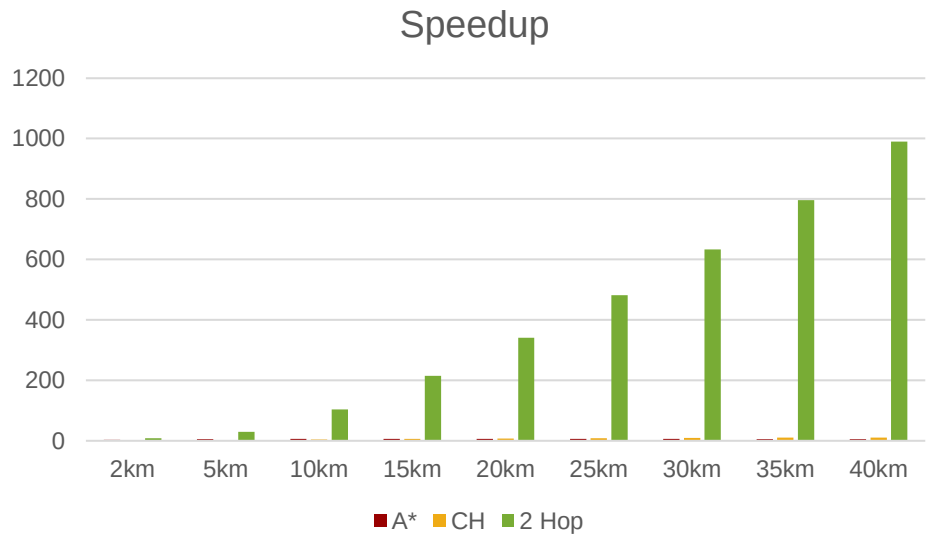
	Dijkstra	A*	CH	2 Hop
2km	0.001096	0.00033454	0.002035	0.0001362
5km	0.00402889	0.00077822	0.00236742	0.000137655
10km	0.0138615	0.00228061	0.00333867	0.00013421
15km	0.0276688	0.00450654	0.00449913	0.000129071
20km	0.0409668	0.00666603	0.00545312	0.000120408
25km	0.0544285	0.00929229	0.0065875	0.000113151
30km	0.0677643	0.0119041	0.007357	0.000107071
35km	0.078577	0.014954	0.0080694	0.000098671
40km	0.0900467	0.0194614	0.00869406	0.000091

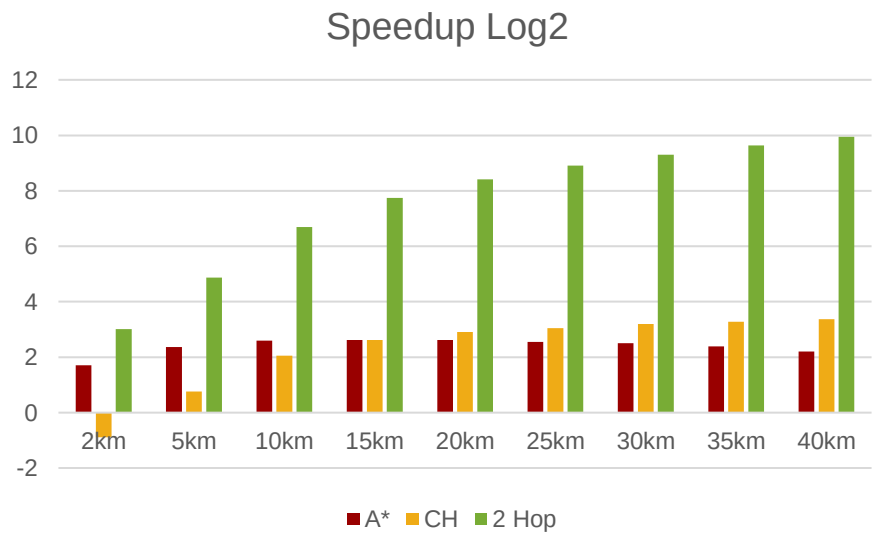


- Dataset
  - Beijing Road Network
  - 302,364 Vertices
  - 387,588 Edges

+ Speedup Compared with Dijkstra's

	A*	CH	2 Hop
2km	3.27618934	0.53857494	8.04698972
5km	5.17705125	1.70180619	29.2680251
10km	6.07797914	4.15180296	103.28217
15km	6.13969919	6.14981119	214.368836
20km	6.14560691	7.51254052	340.233207
25km	5.85738284	8.26239089	481.025355
30km	5.6925177	9.21086041	632.891259
35km	5.25458071	9.73765088	796.353539
40km	4.62693845	10.3572669	989.524176





+ Comparisons on Road Network

	Search Space	Intuition	Query	Index	Preprocessing
Dijkstra's	Circle around s	Distance Increasing	Slowest	0	0
Bi-Dijkstra's	2 Circles around s and t	Bi-directional search	2x	0	0
A*	Ellipse	Towards target	5x	0	0
Bi-A*	2 Ellipses	Bi-directional search	Slightly < A*	0	0
CH	Search with shortcuts	Shortcut	$10\times$	Small	Minutes
PLL	Cut Set	Pruned Search	$10^3 \sim 10^4\times$	Big	Hours

+ Summary

- Index-Free
  - Dijkstra's, Bi-Dijkstra's
  - A*, Bi-A*
- Index-Based
  - CH
  - 2 Hop: PLL