Week 11 Tutorial
Route Planning in Road Network

Fengmei Jin

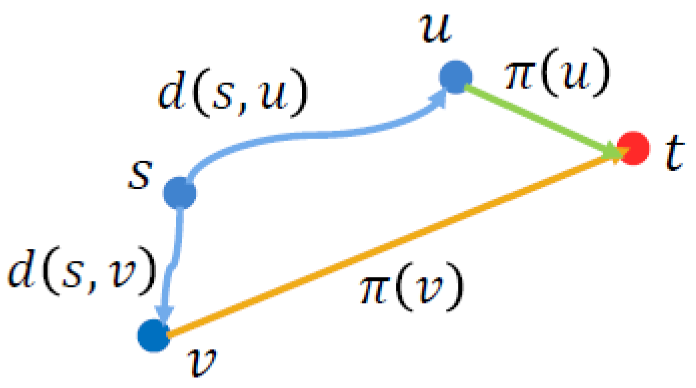
Email: fengmei.jin@uq.edu.au

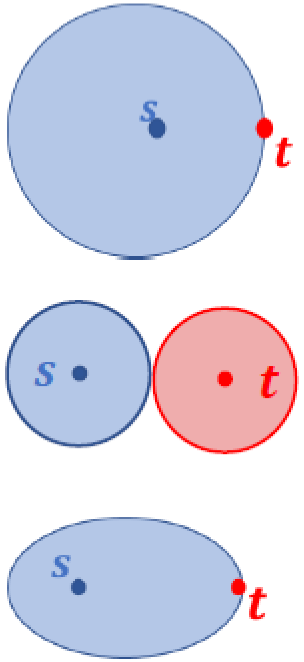
Question 1

- The performance of the distance-estimation based heuristic search algorithms ($A*$ and Landmark) are highly affected by the quality of the estimation.
- Please discuss how the estimated distance influences the algorithm performance.

Question 1 – answers

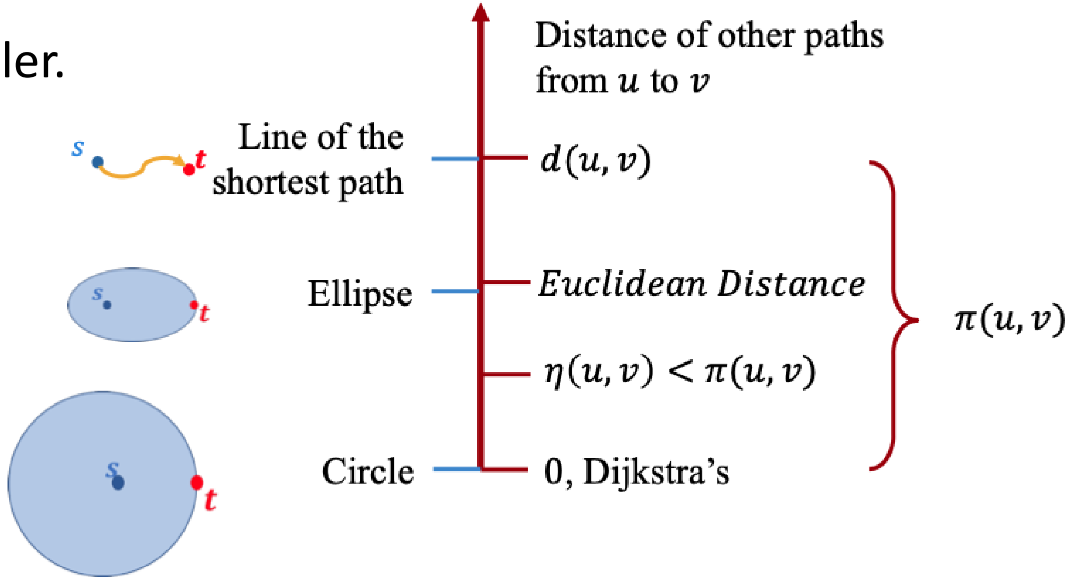
- In the distance-estimation based heuristic search, the key in the priority queue is $d(s,u) + \pi(u)$ instead of $d(s,u)$.
  - $\pi(u)$: Estimated distance from u to t
  - as long as $\pi(u) \leq d(u,t)$, the correctness can be guaranteed.





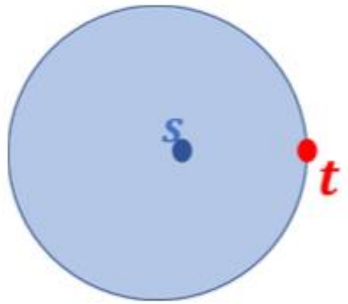
Question 1 – answers

- As for the influence of $\pi(u,t)$, the closer $\pi(u,v)$ is to $d(u,v)$, the smaller search space:
  - When it is 0, the search space reduces to the same as the Dijkstra's (a circle);
  - When $\pi(u,t) = d(u,t)$, the search space is just the shortest path (a line);
  - When $0 \leq \pi(u,t) \leq d(u,t)$, the search space is in between, like an ellipse.
    - when $\pi(u,t)$ is closer to 0, the ellipse is larger;
    - when $\pi(u,t)$ is closer to $d(u,t)$, the ellipse is smaller.

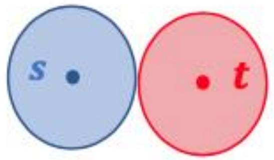


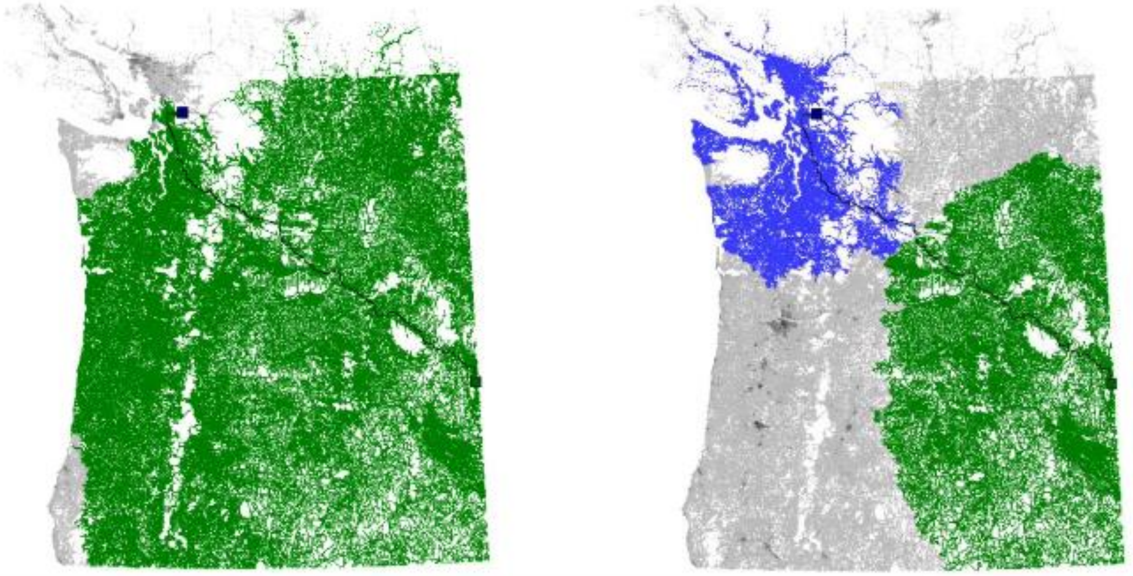
+ Shortest Path: Bi-Dijkstra's

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

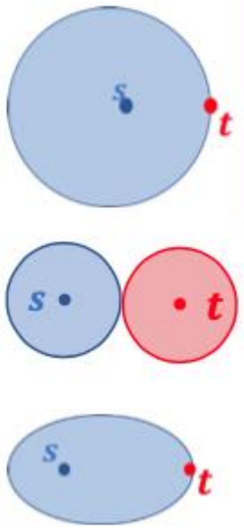


- Bi-Directional Dijkstra's
  - Replace a large circle with 2 smaller ones

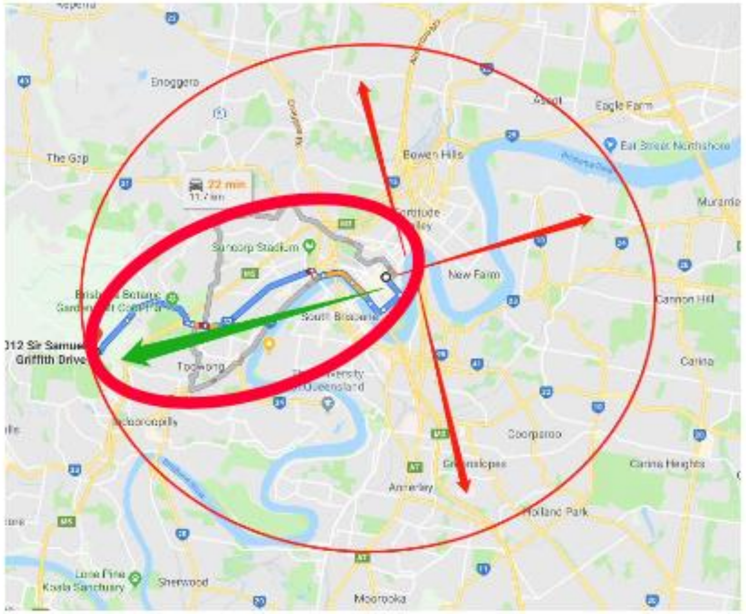




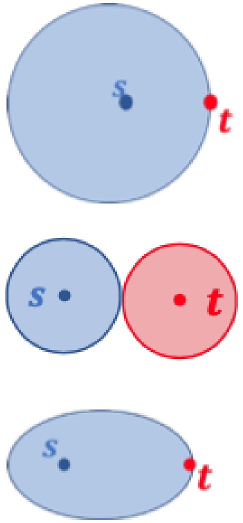
+ Goal Directed: A* Algorithm



- 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 >…
      - $\pi(Toowong) < \pi(Milton) <$ $\pi(\text{Fortitude Valley}) < \cdots$



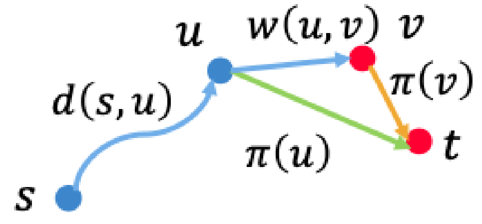
+ Goal Directed: A* Algorithm



- 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

- Dijkstra's-like Search
  - Different key
    - $d_\pi(u) = d(s,u) + \pi(u)$ as key in Q
  - Update top vertex u's neighbor v
    - $d(s,v) = \min(d(s,v)\ ,\ d(s,u) + w(u,v))$
    - Update with $d_\pi(v) = d(s,v) + \pi(v)$
    - Distance information is also preserved
  - Terminate
    - When t is the top of Q

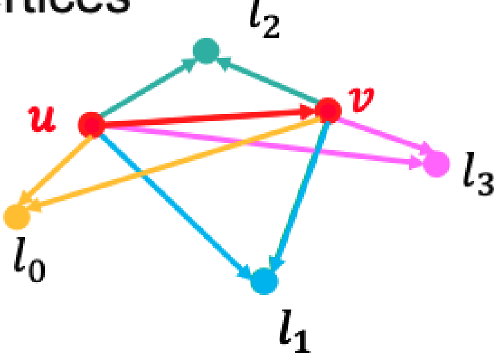


v	d
v_0	$d(s,v_0)$
v_1	$d(s,v_1)$
v_2	$d(s,v_2)$
v_3	$d(s,v_3)$
v_4	$d(s,v_4)$
v_5	$d(s,v_5)$

$d_\pi(s,v_0)$
$d_\pi(s,v_1)$
$d_\pi(s,v_2)$
$d_\pi(s,v_3)$

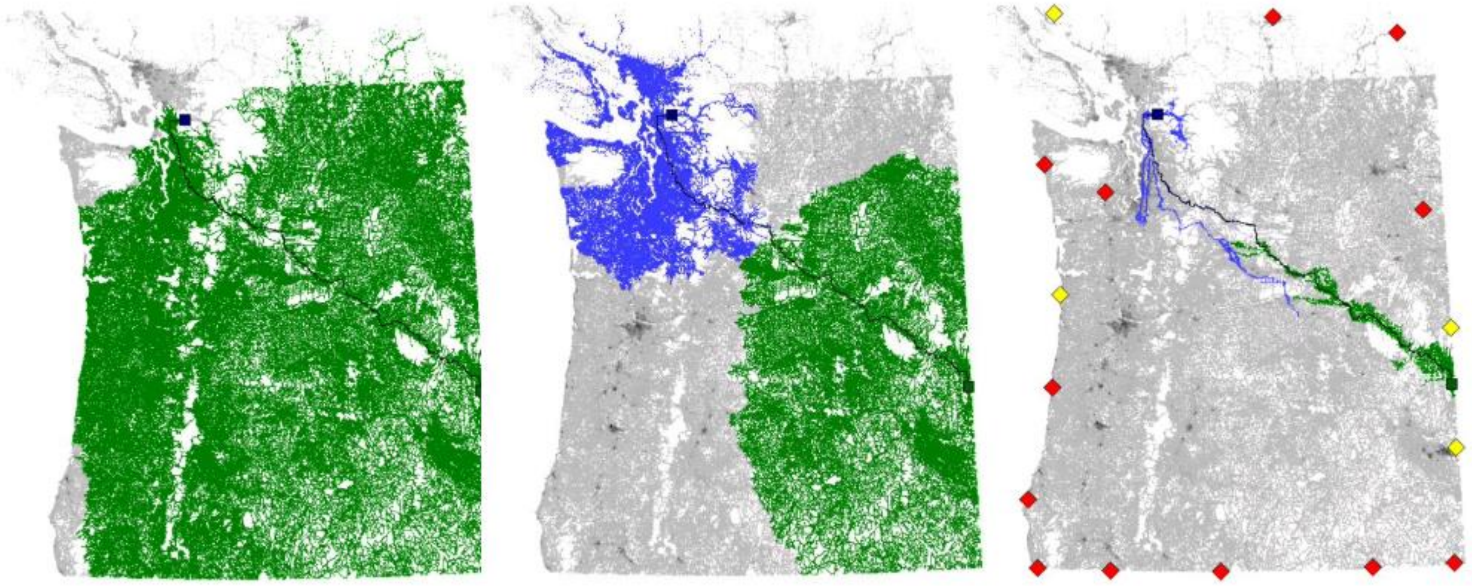
+ Goal Directed: Landmark

- Select a set of vertices $L = \{l_0, l_1, \ldots, l_k\}$
  - Precompute distance from l_i to every other vertices



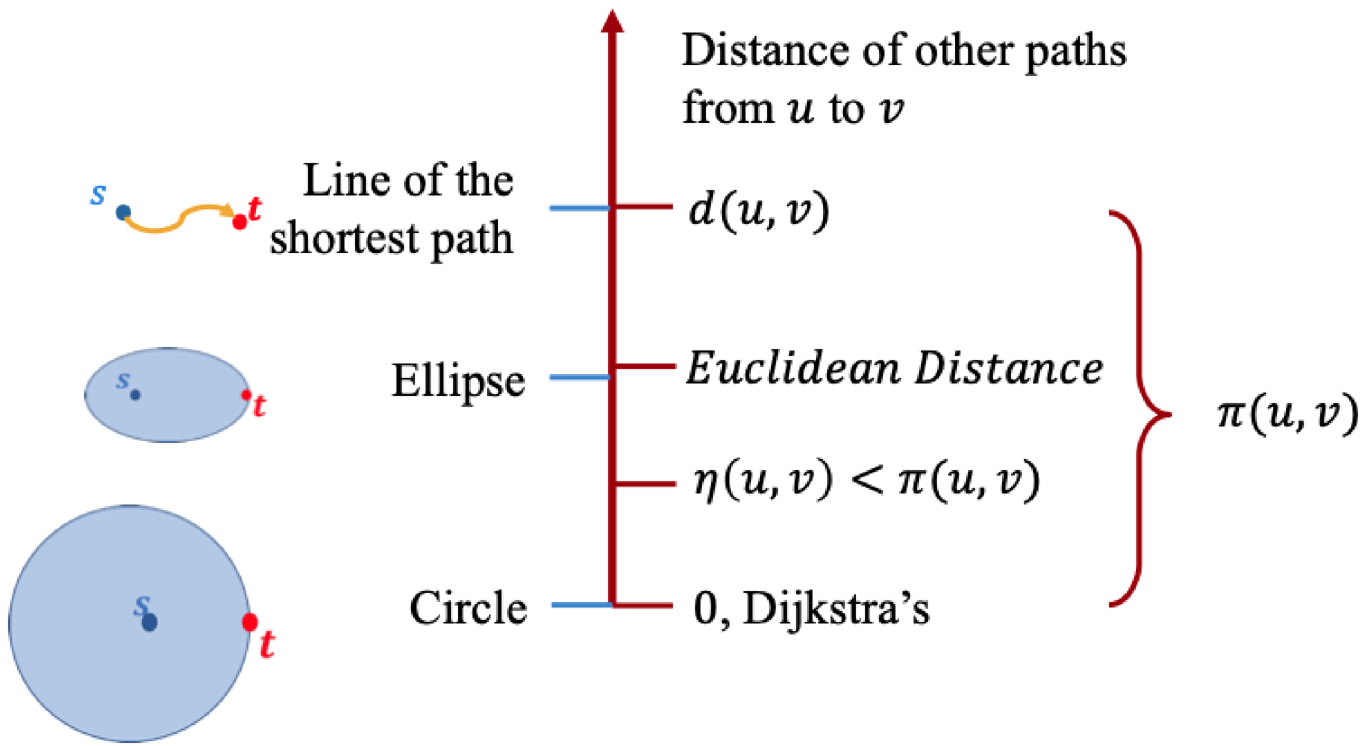
- Lower-bounds
  - $d(u,v) \geq |d(l_i,u) - d(l_i,v)|$
  - Use the maximum as the estimation
    - $\pi(u,v) = max(|d(l_i,u) - d(l_i,v)|)$

- A good landmark
  - Before u, like l_0
  - After v, like l_3



+ Goal Directed

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



Question 2

- The Dijkstra's algorithm is essentially a single-source shortest path algorithm that can be used to answer the point-to-point path query.
- By contrast, the $A*$ algorithm is essentially a point-to-point shortest path algorithm, as its searching heuristic is towards a destination.
  - The single-source shortest path: to find shortest paths from a single source vertex v to all other vertices in the graph
  - The point-to-point shortest path: from a source to a single destination
- Can $A*$ also be used as a single-source algorithm? If so, how does it work?

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

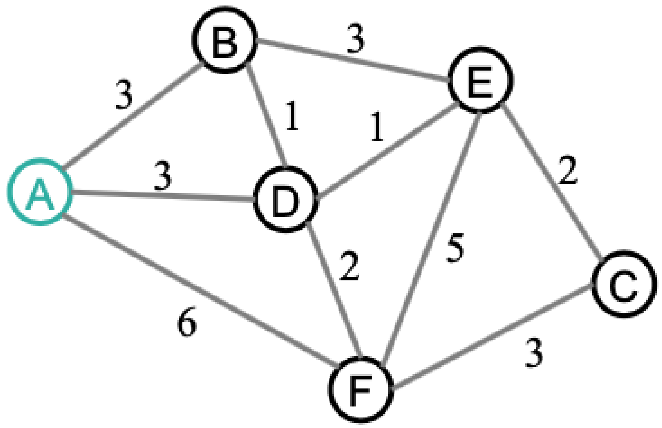
V Top Vertex: Minimum in Q

V Visited Vertex: Not in Q, Found Shortest

V Unvisited Vertex: Not in Q, still ∞

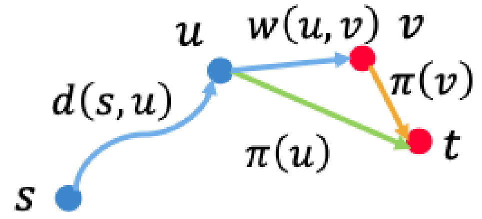
v	d
A	0
B	∞
C	∞
D	∞
E	∞
F	∞

$A(0)$



+ Goal Directed: A* Algorithm

- Dijkstra's-like Search
  - Different key
    - $d_\pi(u) = d(s,u) + \pi(u)$ as key in Q
  - Update top vertex u's neighbor v
    - $d(s,v) = \min(d(s,v)\ ,\ d(s,u) + w(u,v))$
    - Update with $d_\pi(v) = d(s,v) + \pi(v)$
    - Distance information is also preserved
  - Terminate
    - When t is the top of Q



v	d
v_0	$d(s,v_0)$
v_1	$d(s,v_1)$
v_2	$d(s,v_2)$
v_3	$d(s,v_3)$
v_4	$d(s,v_4)$
v_5	$d(s,v_5)$

$d_\pi(s,v_0)$
$d_\pi(s,v_1)$
$d_\pi(s,v_2)$
$d_\pi(s,v_3)$

Question 2 – answers

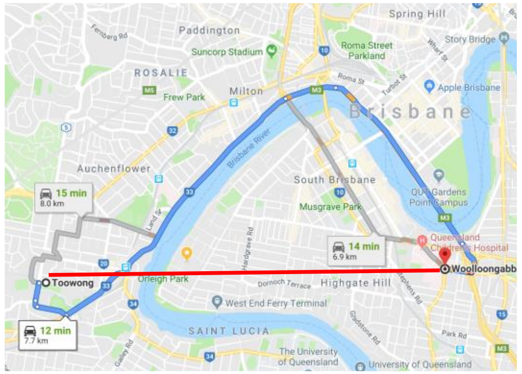
- Yes, $A*$ can be used to find the distances from the source to all the other vertices in the graph.
- The procedure is the same as the Dijkstra's:
  - When a point pops out of the priority queue with $d(s, u) + \pi(u, t)$, this $d(s, u)$ is the shortest distance from s to u.
  - We can run this procedure on and on, even passes t.
  - The algorithm terminates when the priority queue becomes empty. In this way, we can find the distance from s to all the other vertices.

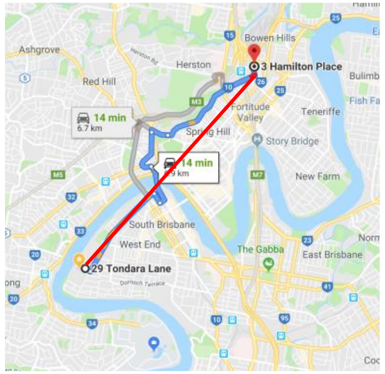
******The proof is not required by this course.*

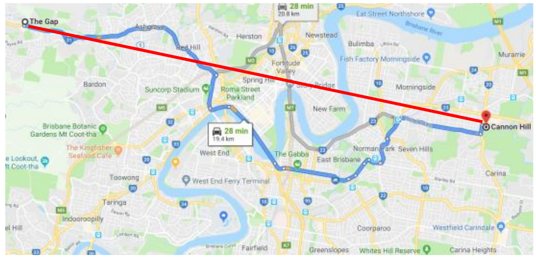
+ Shortest Path Query

- All the previous mentioned methods
  - Search only use the original graph
  - Nearly no pre-processing
  - Building brick for the ground truth
  - But slow…
- Query
  - Faster query answering
  - With index
    - Add shortcuts?
    - All pair distance?

+ Preprocessing





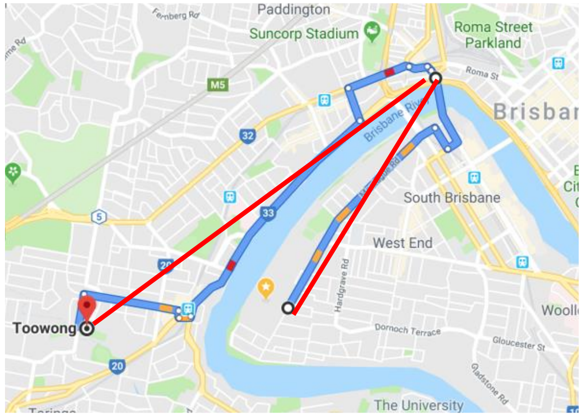


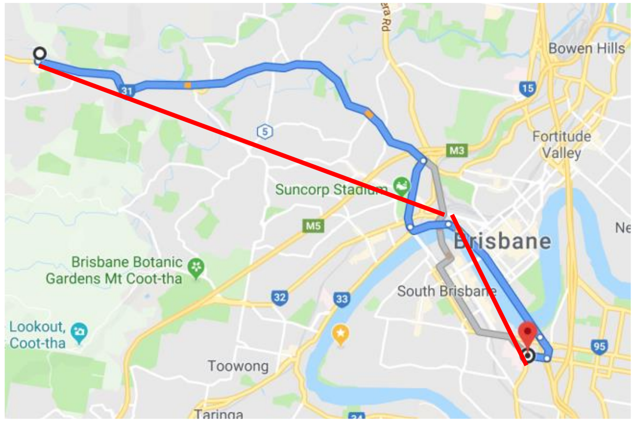
■ The first shortcut approach

- Contraction Hierarchy (CH)
  - Contract the more important City…

■ The second cut-and-paste approach

- 2 Hop Labelling
  - Toowong to City: 1 Hop, City to West End: 1 Hop
  - The Gap to City: 1 Hop, City to Woolloongaba: 1 Hop





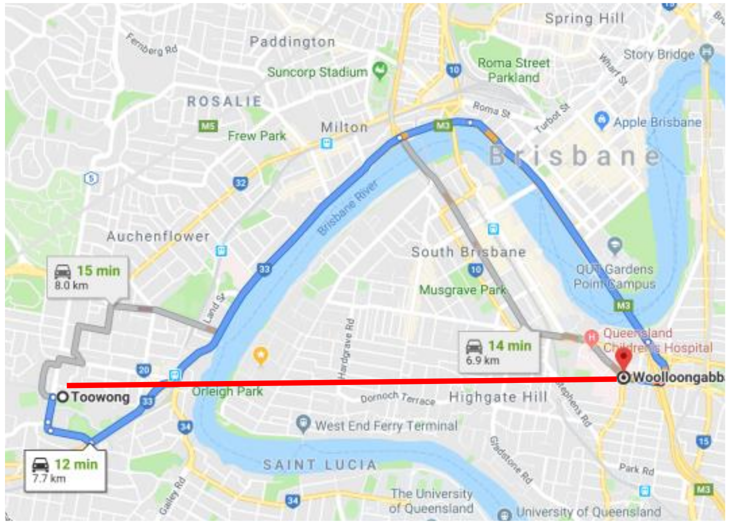
Question 3 – Contraction Hierarchy

- During the construction of CH, the shortcuts we add have the actual shortest distances between the vertices. However, this requires a large amount of Dijkstra's search to guarantee this property.

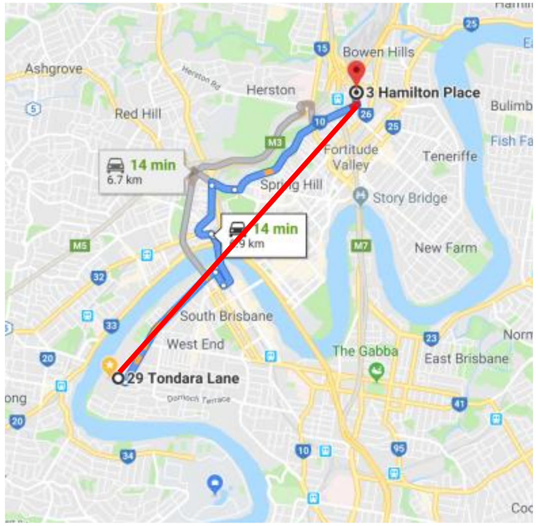
Question 3 – Contraction Hierarchy

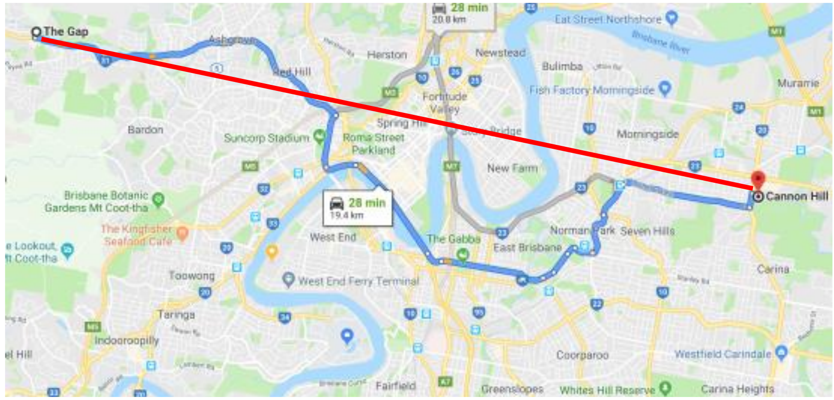
- During the construction of CH, the shortcuts we add have the actual shortest distances between the vertices. However, this requires a large amount of Dijkstra's search to guarantee this property.
- Instead, if for any neighbour pair (v,w) of u, where v and w are contracted later than u, we add a shortcut (v,w) with distance $\min\left(d(v,u)+d(u,w),d(v,w)\right)$, then the contraction process will become much faster.
  - $d(v,w)=\infty$ if there is no edge or shortcut between v and w
- Can this method also answer the query correctly? What are the advantages and the disadvantages of this method compared with the classic CH?

+ Preprocessing



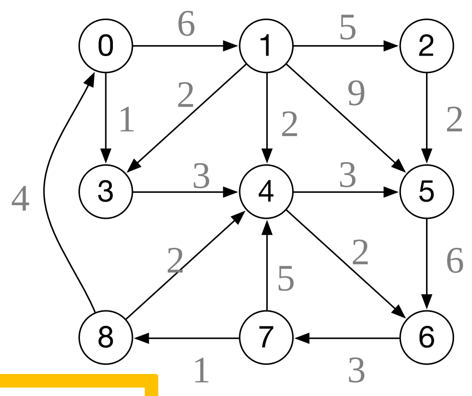
- Let's come back to Brisbane map
  - Brisbane City
    - City has several bridges
    - City is connected with M3, M5 and M7
    - City is in the center
    - City plays a more important role in the shortest paths
    - …
  - Add shortcuts for all the shortest paths that travel via City
  - When searching for a shortest path, we can use these helpful shortcuts without searching the City
    - Search space is reduced!
  - In a way, City is "contracted"

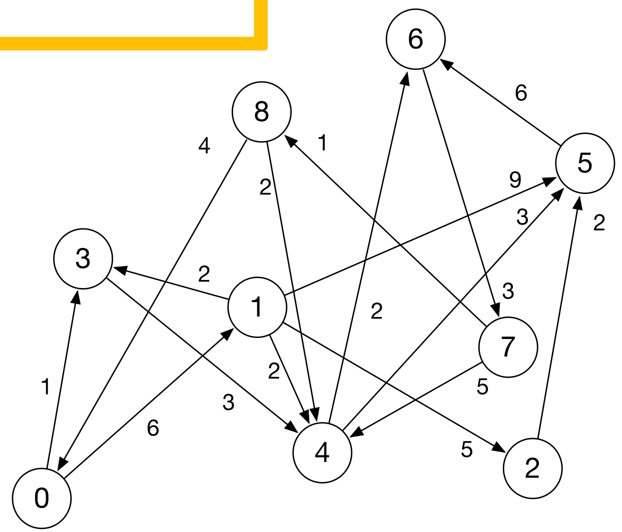




+ Contraction Hierarchy[1]

- 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

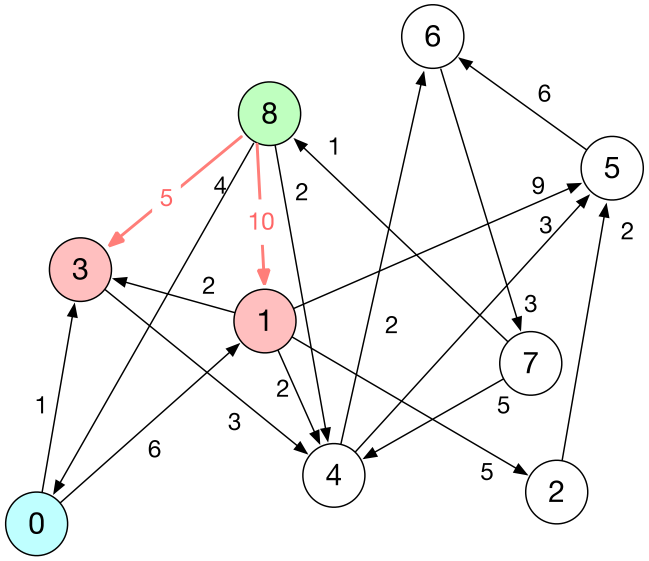


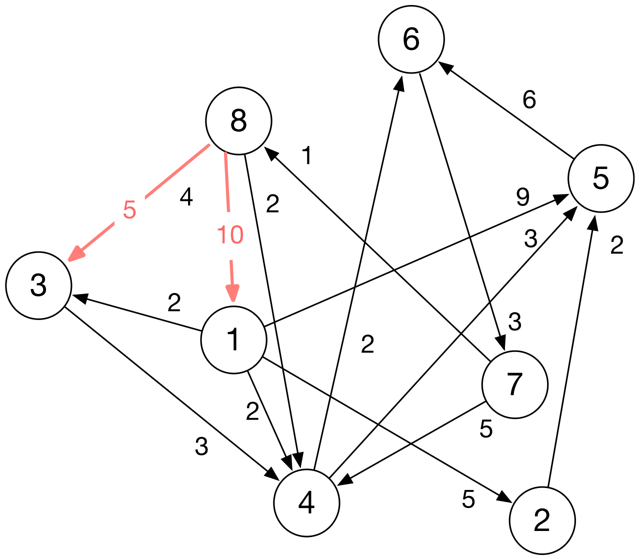


[1] Contraction Hierarchies Faster And Simpler Hierarchical Routing In Road Networks, International Workshop on Experimental and Efficient Algorithms, 2018

+ Contraction Hierarchy

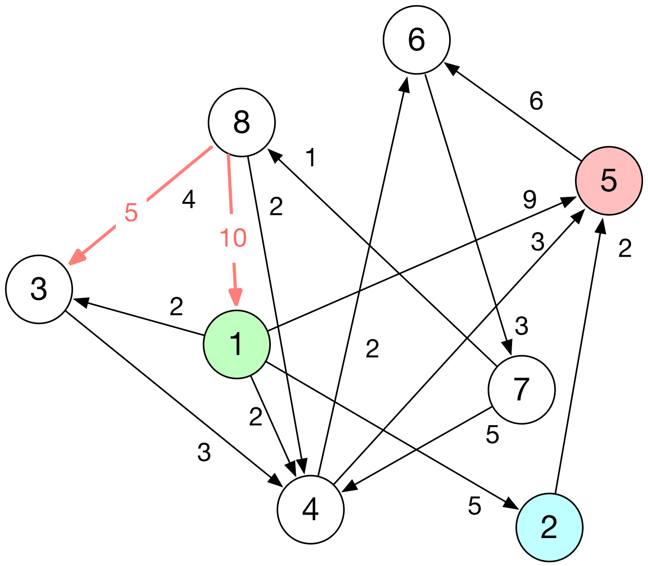
- Contract v_0
  - In-Neighbor: v_8
  - Out-Neighbor: v_1, v_3
  - Shortest Distance:
    - $d(v_8, v_1) = 10$, add shortcut
    - $d(v_8, v_3) = 5$, add shortcut

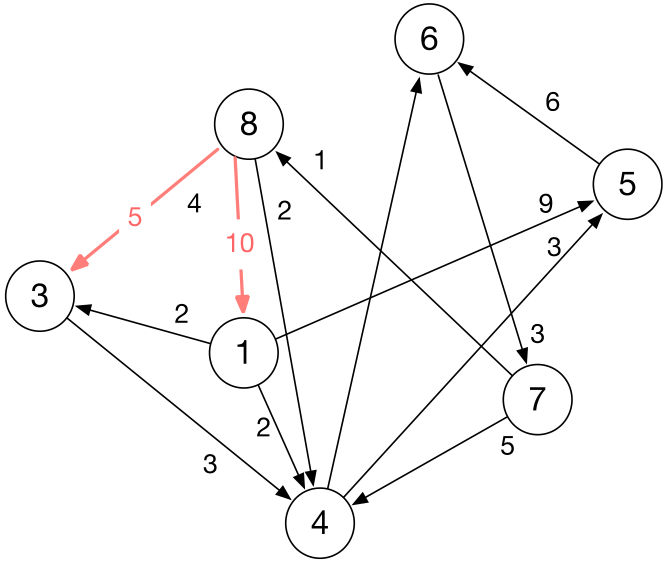




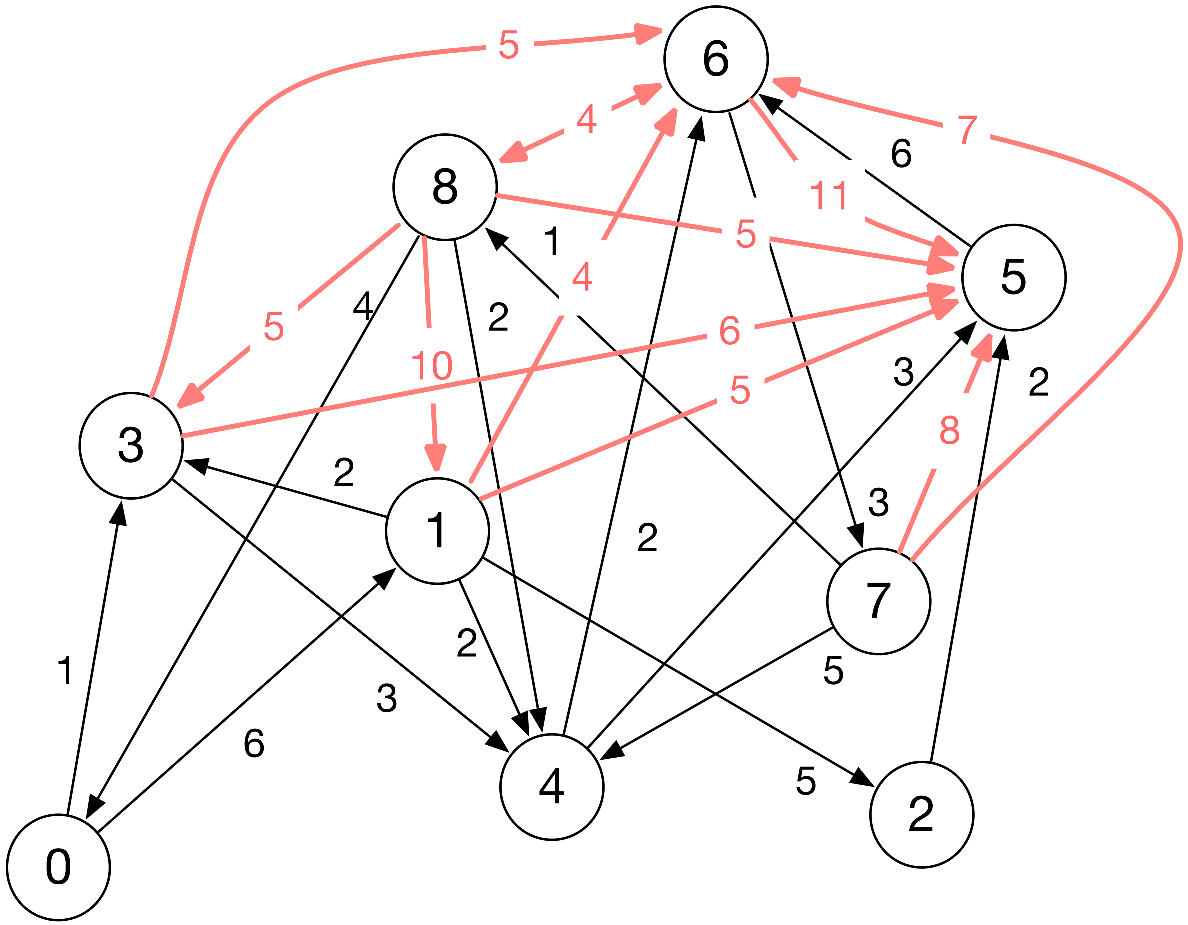
+ Contraction Hierarchy

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



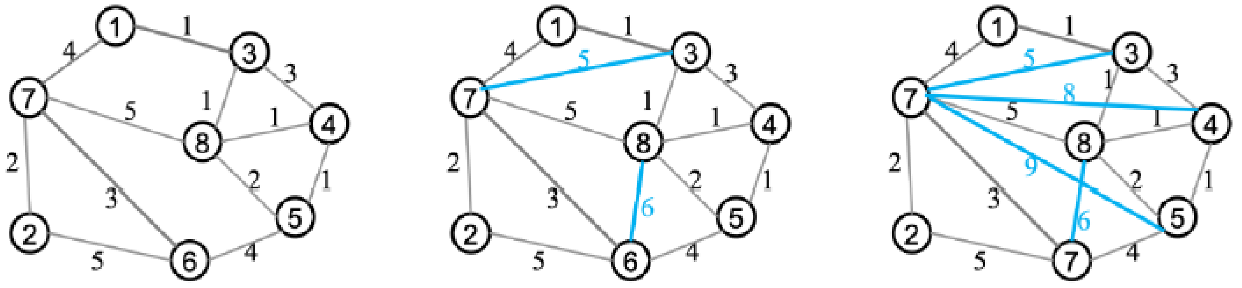


+ Contraction Hierarchy



+ CH without searching

- CH construction is slow, because of Dijkstra's search
  - How to avoid searching?
  - For all $u, w \in v$'s neighbors
    - CH: add shortcut $d(u,v) + d(v,w)$ only when $u \rightarrow v \rightarrow w$ is the shortest path from u to w
    - Or: Update the existing one by $\min(d(u,w), d(u,v) + d(v,w))$ → much looser



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

Question 3 – correctness

- Yes, the correctness is guaranteed.
- Because we still need to run the Bi-directional upwardly search, the Dijkstra's search is in a way postponed to the query answering stage.
- When we contract a vertex v, the standard CH adds the shortcut only when $u \to v \to w$ is the shortest path from u to w; while the new method is much looser and creates more shortcuts even when $u \to v \to w$ is not the shortest path.
- This redundant information guarantees the correctness of the latter contractions.

Question 3 – pros and cons

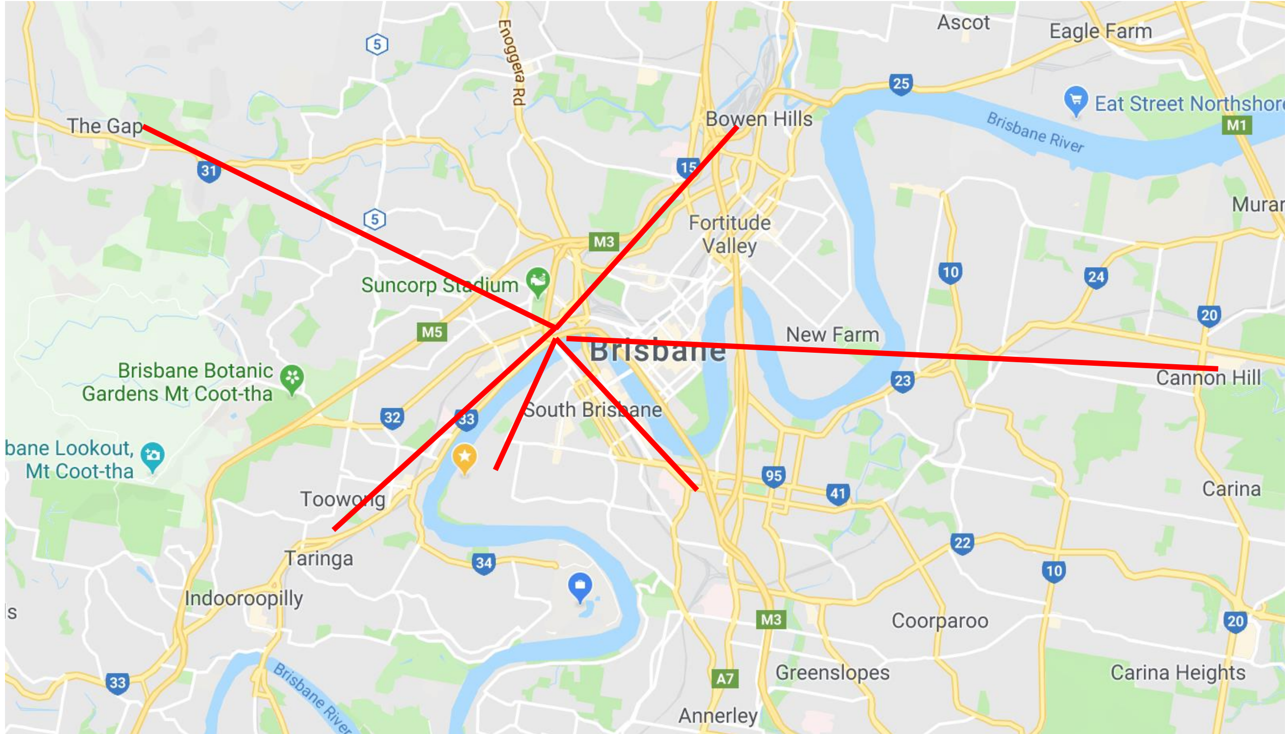
- This approach has the advantage of fast construction because no Dijkstra is needed.
- But it takes longer time to answer a query than the standard CH, because it has a much larger index size. *(Similar performance to $A*$)*

Question 4

- Pruned Landmark Labelling is an easy way to construct the 2 Hop labels. However, it is more suitable for the $small\ world$ graph than the road network.
- The small world graph is a kind of graph that is composed of several densely connected communities, and the connections between the communities are very limited.
- Please discuss how the pruned landmark labelling can benefit from this property.

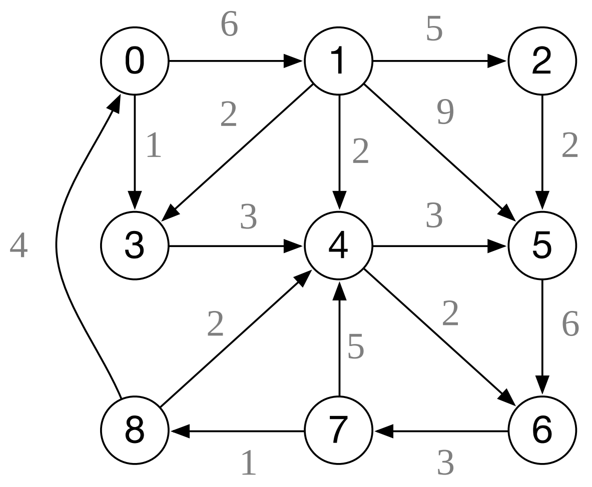
+ Preprocessing

- 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 !



+ 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

- $v_0 \rightarrow v_6$

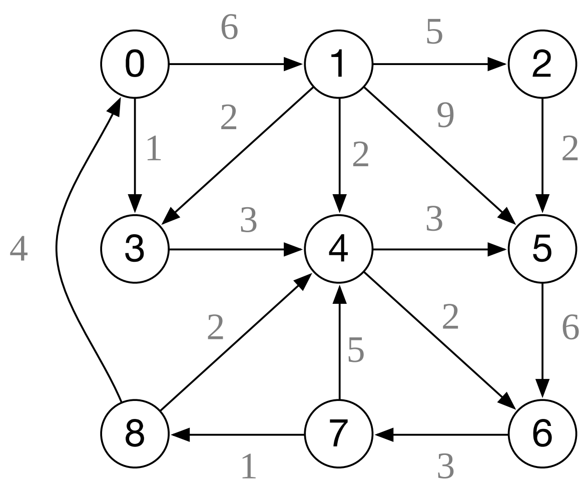
v_0 Out	v_6 In
$(v_4, 4)$	$(v_4, 2)$
$(v_1, 6)$	$(v_5, 6)$
$(v_3, 1)$	

  - Hop 4: $d = 4 + 2 = 6$

- $v_2 \rightarrow v_3$

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

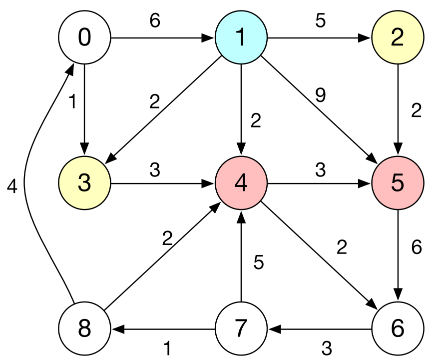
  - Hop 4: 14 + 11 = 25
  - Hop 1: 22 + 2 = 24
  - Hop 5: 2 + 15 = 17



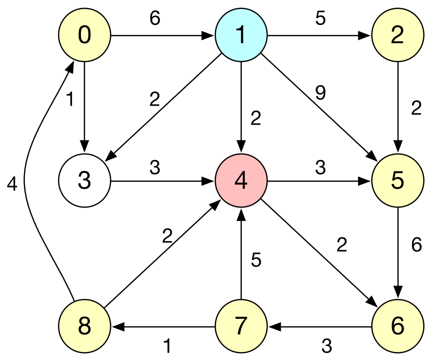
Node	Out-Label	In-Label
0	(4,4) (1,6) (3,1)	(4,10) (5,14) (6,8)
1	(4,2)	(4,16)
2	(4,14) (1,22) (5,2)	(4,21) (1,5)
3	(4,3)	(4,11) (1,2) (5,15) (6,9)
4		
5	(4,12) (1,20)	(4,3)
6	(4,6) (1,14)	(4,2) (5,6)
7	(4,3) (1,11) (3,6) (0,5)	(4,5) (5,9) (6,3)
8	(4,2) (1,10) (3,5) (0,4)	(4,6) (5,10) (6,4) (7,1)

+ Pruned Landmark Labelling (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
  - Correct but slow and huge

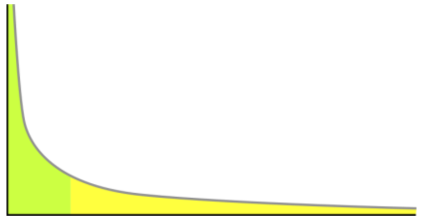
- Pruned Search
  - If $d(v_i, v_j)$ is not smaller than the query result of the existing labels
    - Stop searching from v_j
    - Do not visit v_j neighbors

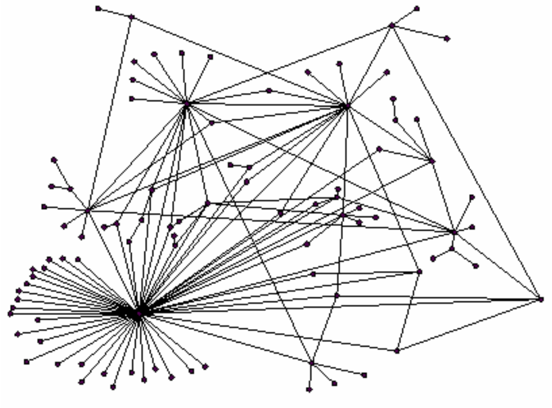
Question 4 – answers

- The small world graph has a property that small number of nodes have a very large number of degrees, such that most of the remaining nodes connect to them. Therefore, these nodes are the natural hub of the graph. The shortest paths among other vertices have a very high chance to pass through them.
- Then if we run the pruned landmark labelling in the degree decreasing order, these important nodes would become labels in other nodes earlier, and they can help prune the search space a lot.

+ 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