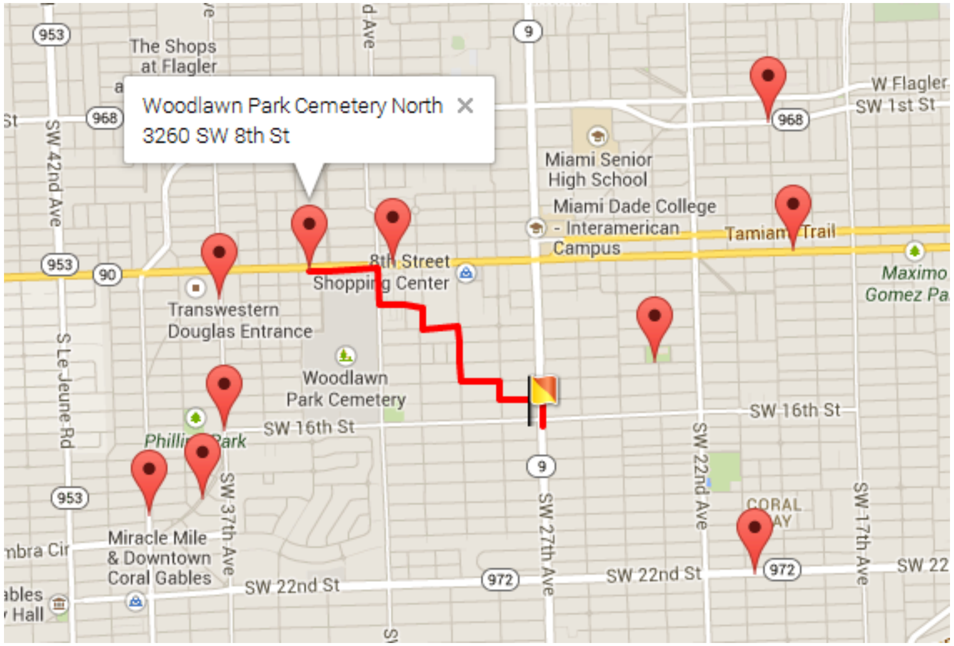
Big Data Summer School

Graph Algorithms

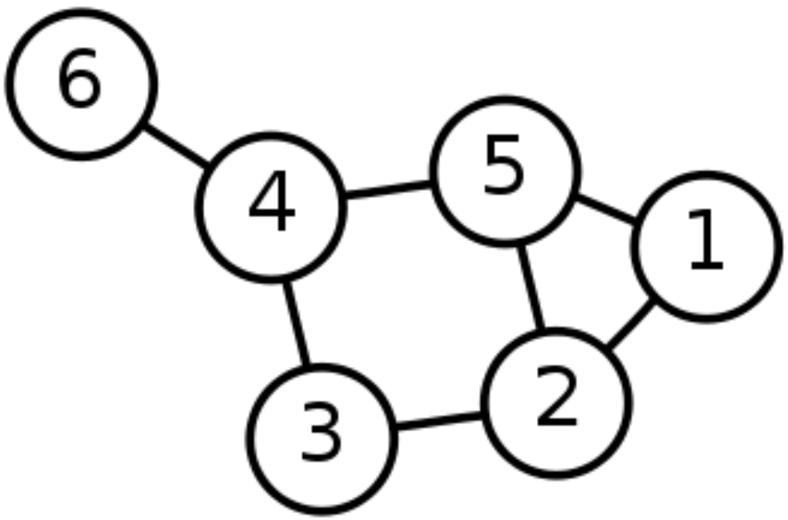
KNN Search on Road Network

- Given **[a query location]**, **[a set of candidate objects]**, KNN query returns the **[K]**-nearest objects ranked by shortest path distance.



Single-Source Shortest Path Problem

Single-Source Shortest Path Problem - The problem of finding shortest paths from a source vertex *v* to all other vertices in the graph.



Dijkstra's algorithm

Dijkstra's algorithm - is a solution to the single-source shortest path problem in graph theory.

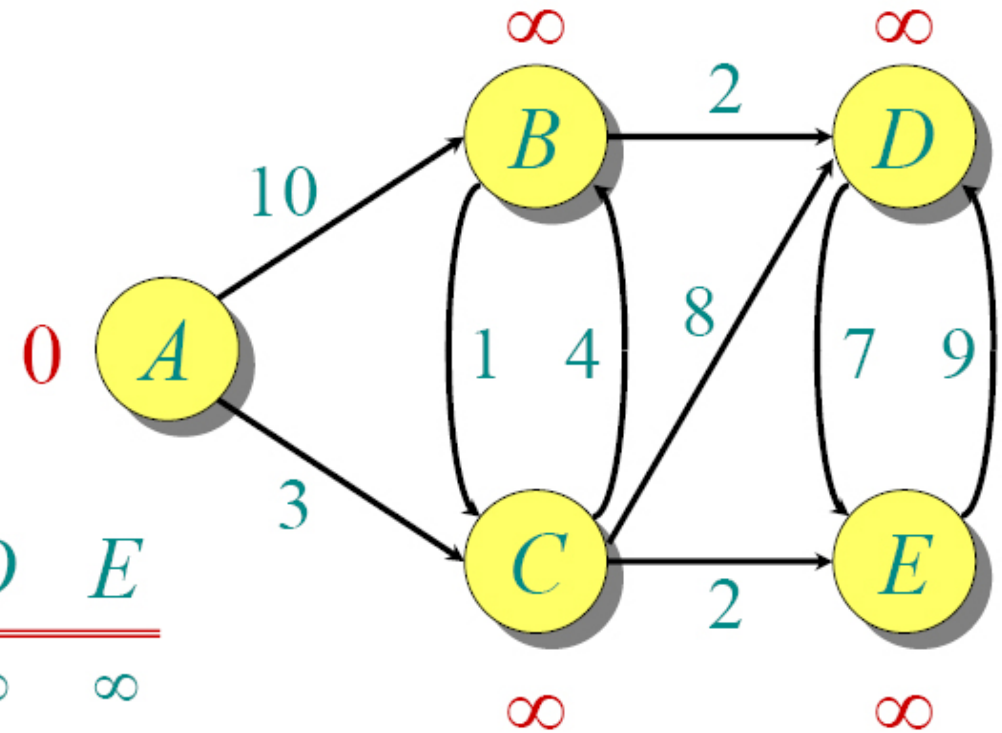
Approach: Greedy

Input: Weighted graph G={E,V} and source vertex $v \in V$, such that all edge weights are nonnegative

Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex $v \in V$ to all other vertices

Dijkstra Animated Example

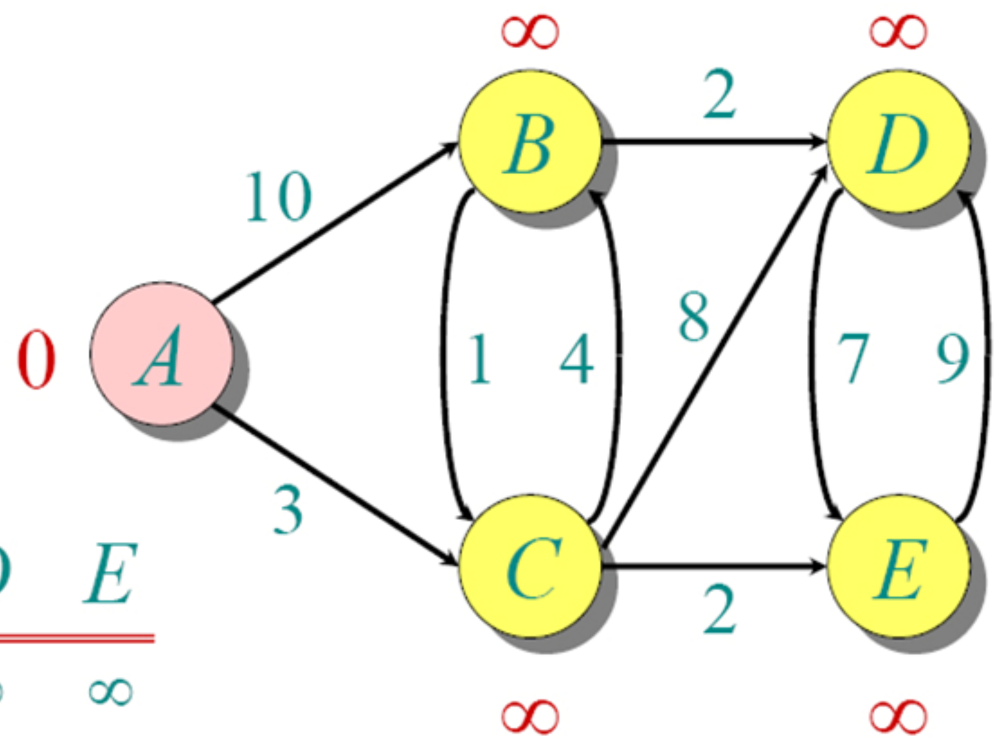
Initialize:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

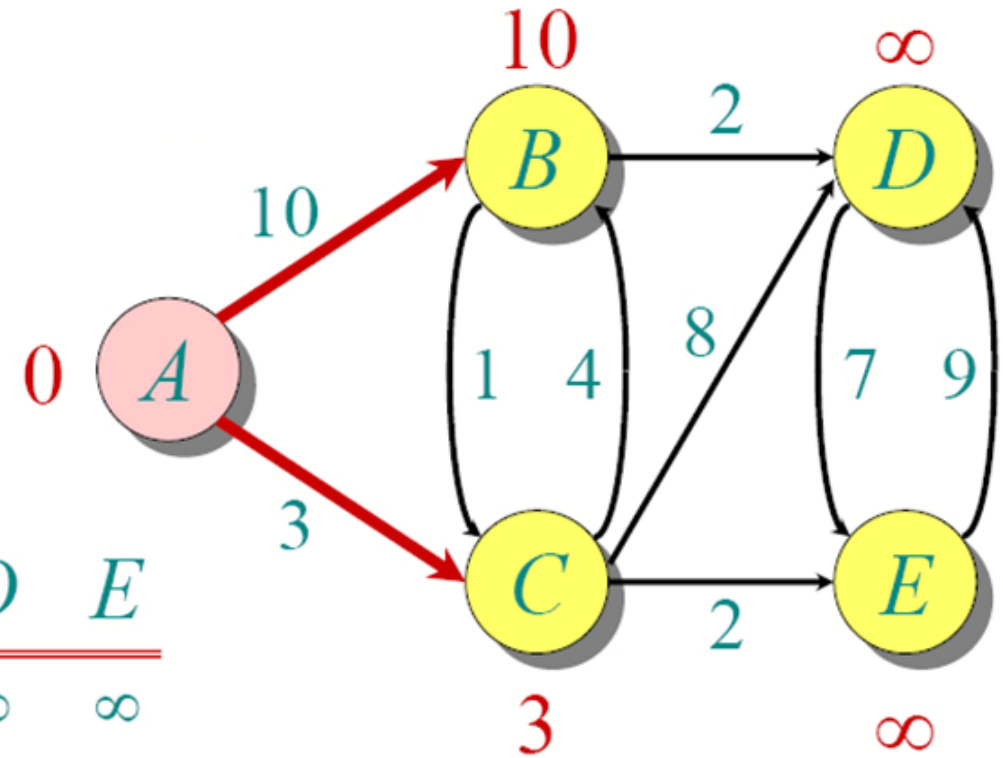
S: {}

Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

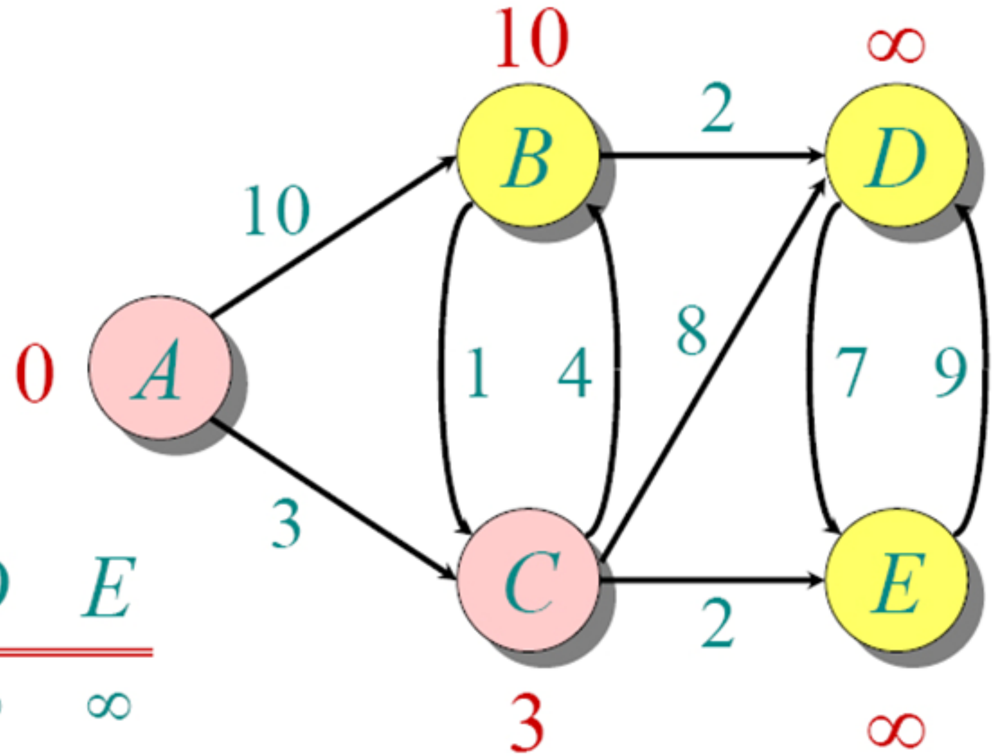
Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

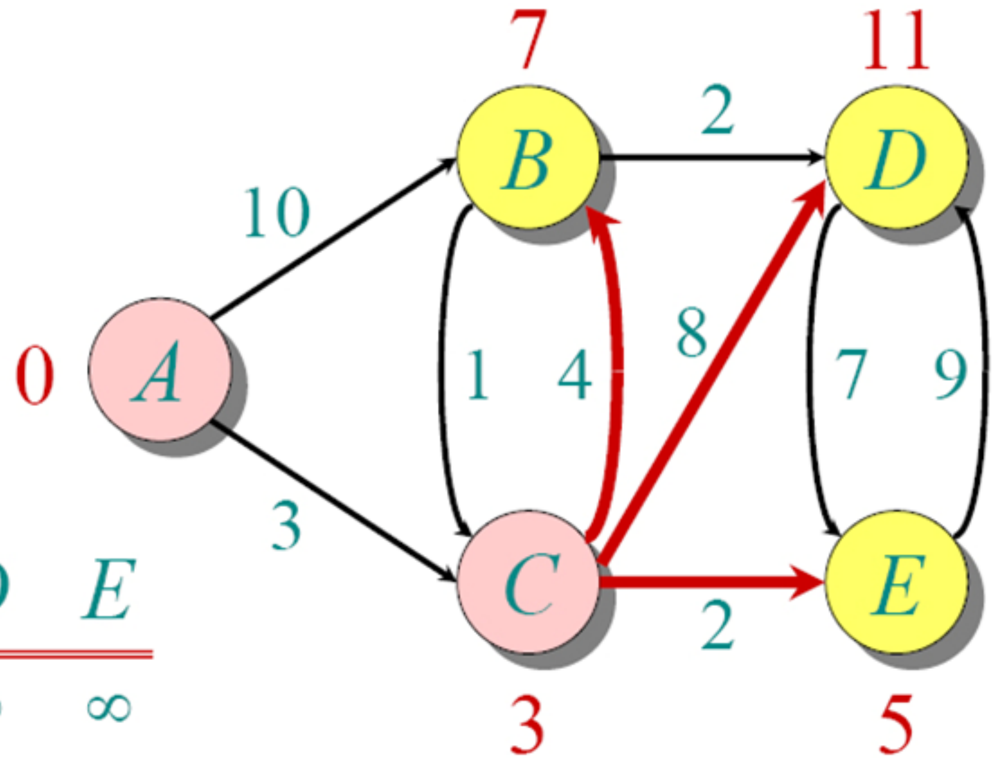
Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

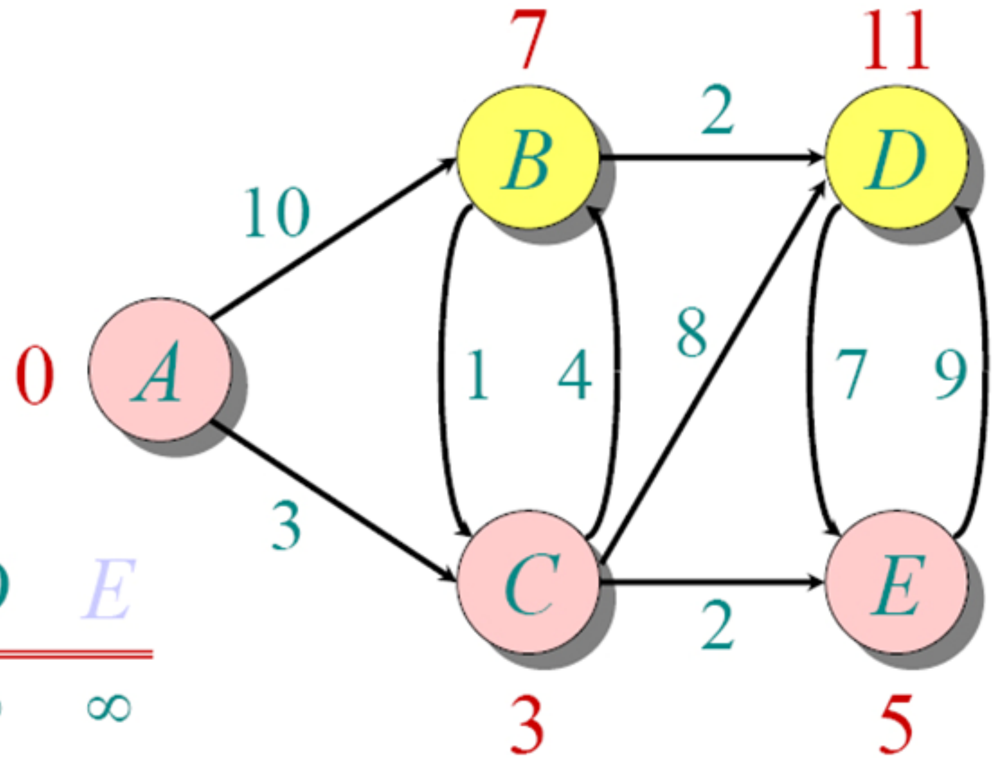
Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C }

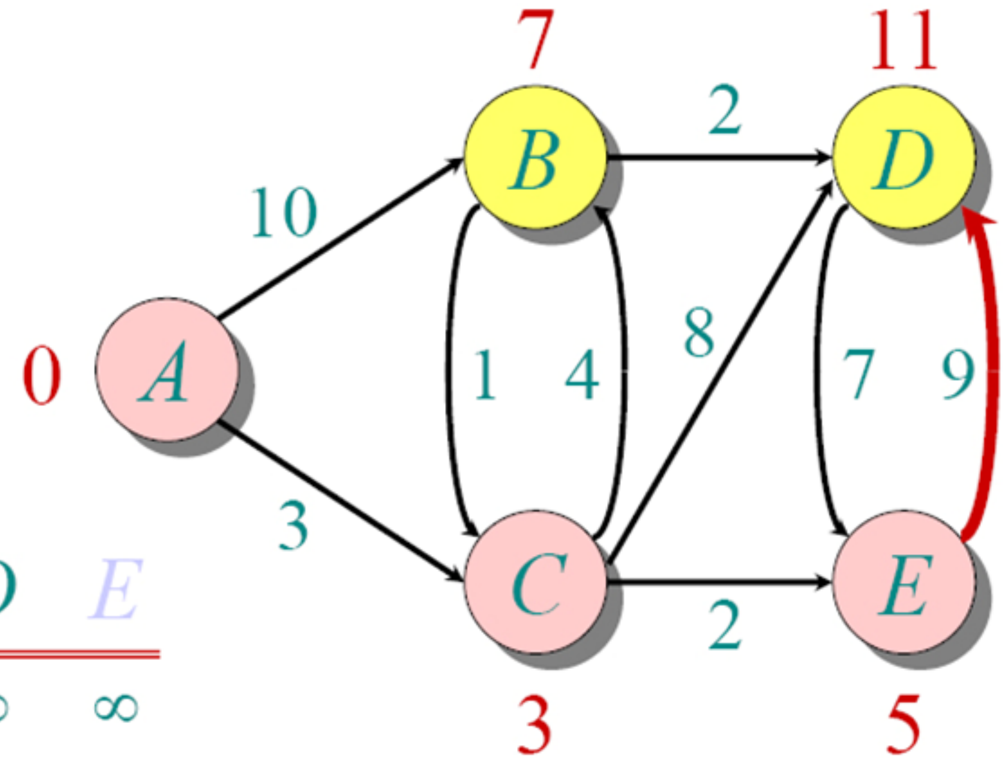
Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C, E }

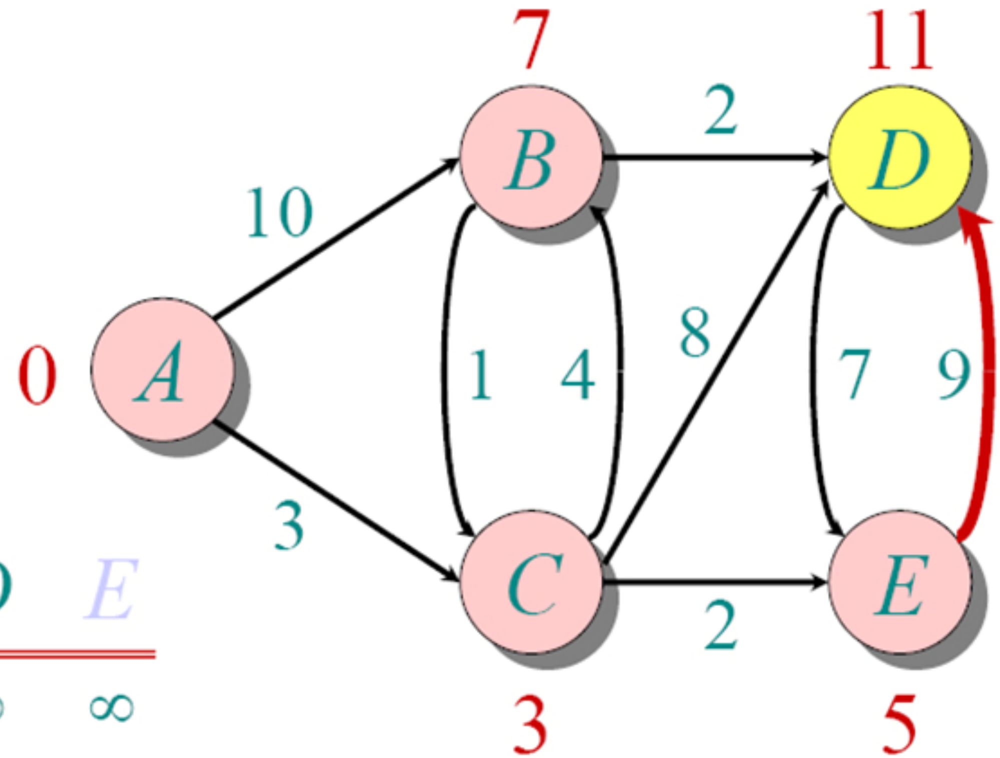
Dijkstra Animated Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }

Dijkstra Animated Example

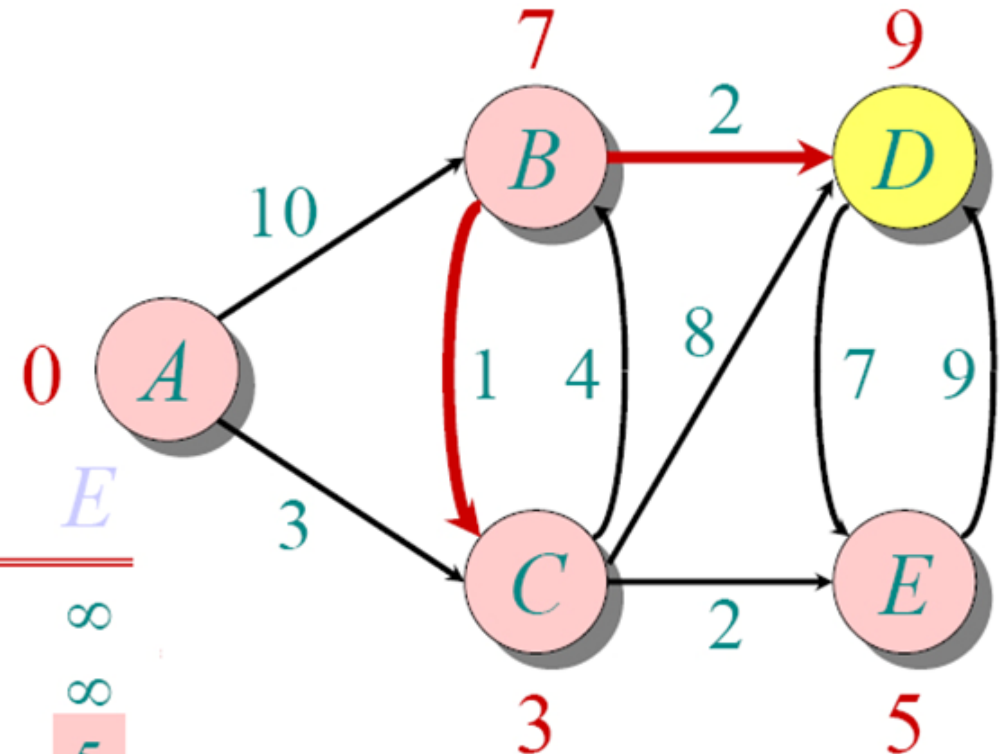


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E, B }

Dijkstra Animated Example

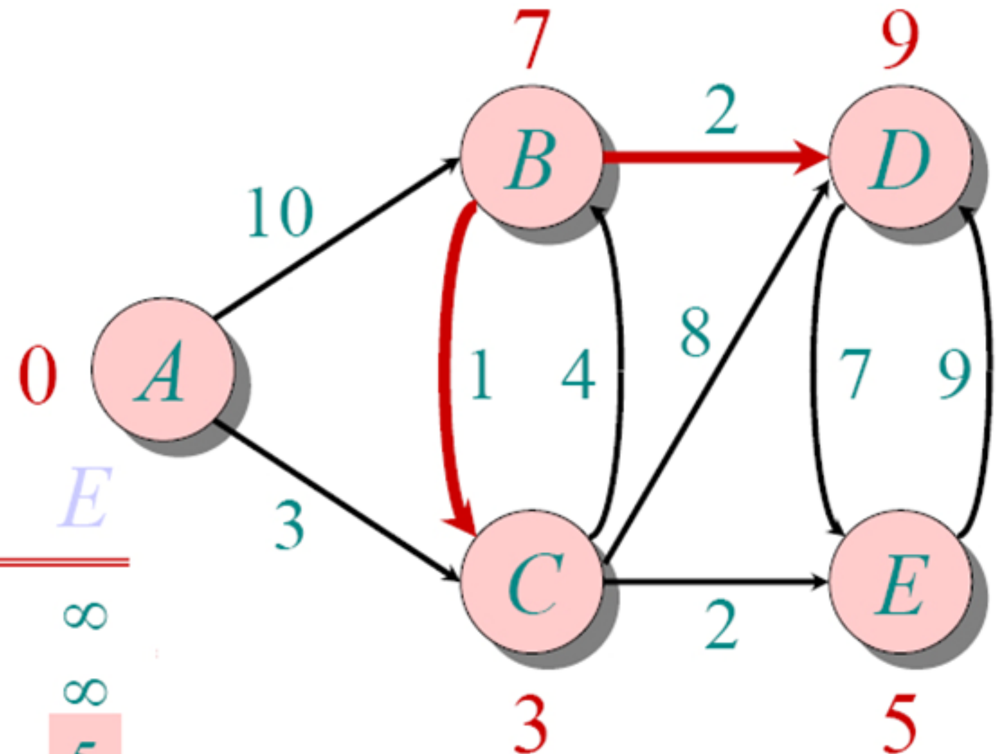
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	



S: { A, C, E, B }

Dijkstra Animated Example

Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	$\boxed{5}$
		$\boxed{7}$		11	
				$\boxed{9}$	



S: { A, C, E, B, D }

Dijkstra's algorithm - Pseudocode

```
dist[s] ←0                                  (distance to source vertex is zero)
for  all v ∈ V–{s}
     do  dist[v] ←∞                         (set all other distances to infinity)
S←∅                                         (S, the set of visited vertices is initially empty)
Q←V                                         (Q, the queue initially contains all vertices)
while Q ≠∅                                  (while the queue is not empty)
do   u ← mindistance(Q,dist)                (select the element of Q with the min. distance)
    S←S ∪ {u}                               (add u to list of visited vertices)
    for all v ∈ neighbors[u]
         do  if   dist[v] > dist[u] + w(u, v)              (if new shortest path found)
                  then     d[v] ←d[u] + w(u, v)      (set new value of shortest path)
                        (if desired, add traceback code)
return dist
```

Implementations and Running Times

The simplest implementation is to store vertices in an array or linked list. This will produce a running time of $O(|V|^2 + |E|)$.

For sparse graphs, or graphs with very few edges and many nodes, it can be implemented more efficiently storing the graph in an adjacency list using a binary heap or priority queue. This will produce a running time of $O((|E|+|V|) \log |V|)$

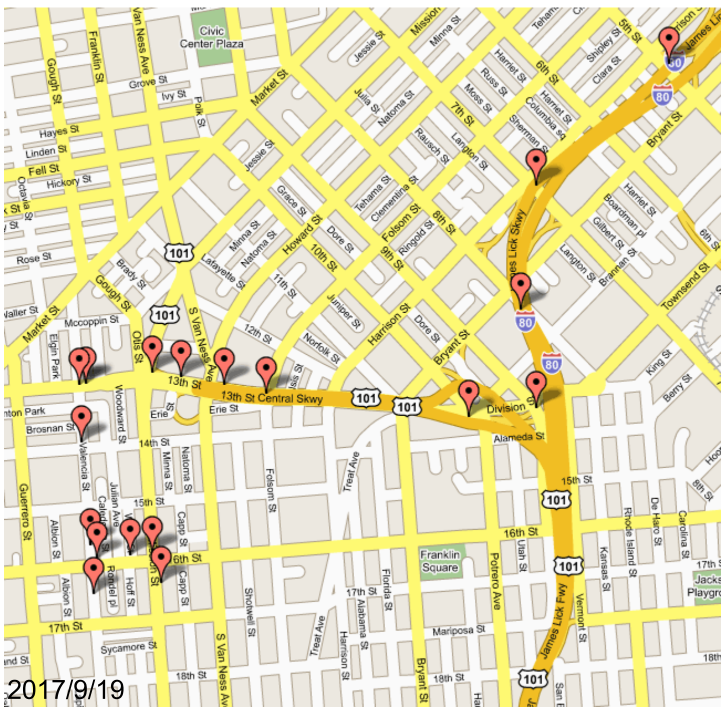
Dijkstra - Why It Works

- **Lemma 1**: Triangle inequality
  If $\delta(u,v)$ is the shortest path length between u and v,
  $\delta(u,v) \le \delta(u,x) + \delta(x,v)$
- **Lemma 2**:
  The subpath of any shortest path is itself a shortest path.

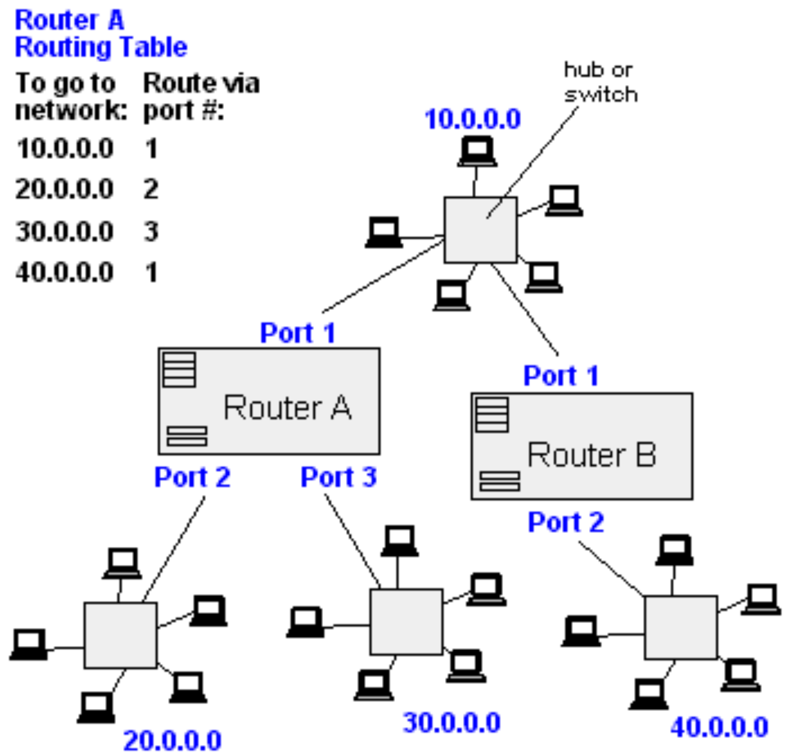
- The key is to understand why we can claim that anytime we put a new vertex in S, we can say that we already know the shortest path to it.

Applications of Dijkstra's Algorithm

- Traffic Information Systems are most prominent use
- Mapping (Map Quest, Google Maps)
- Routing Systems

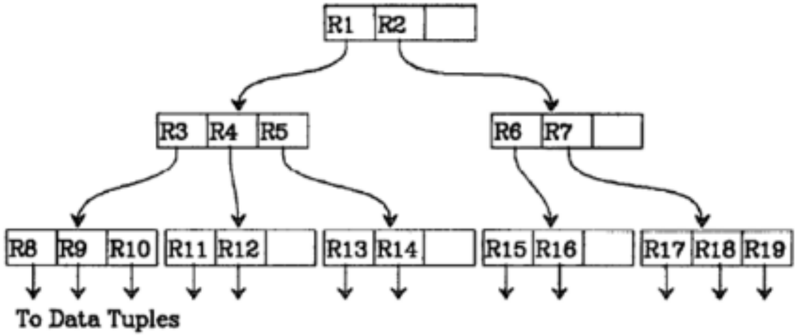


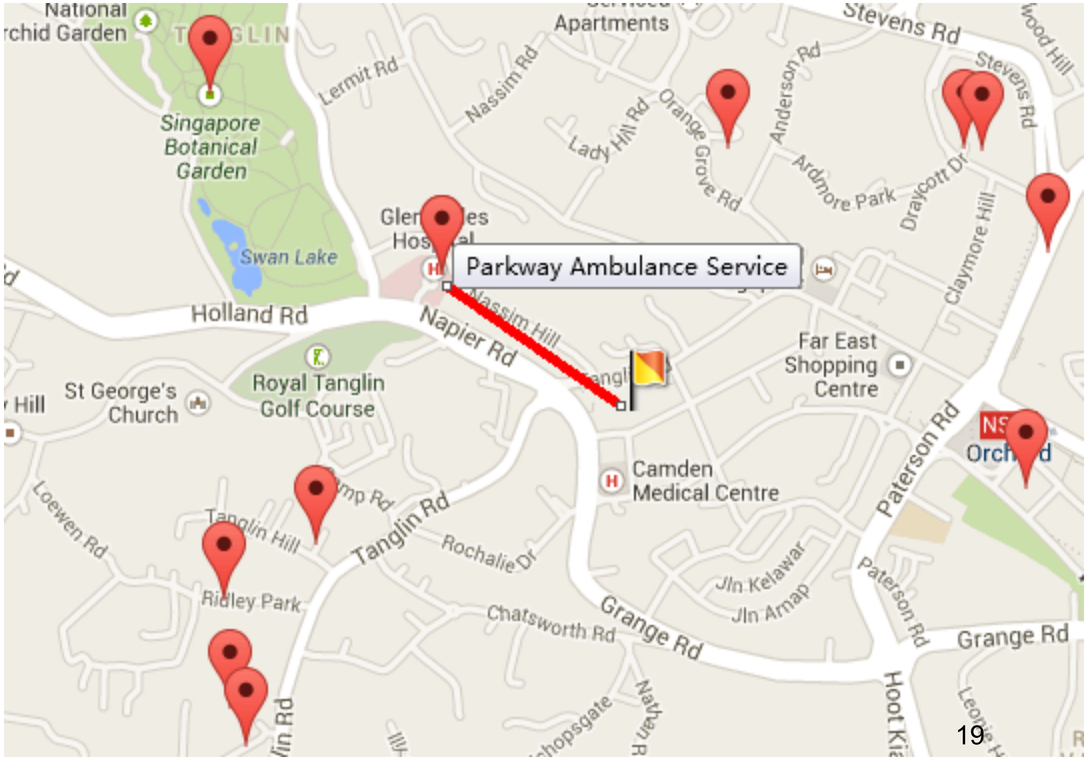
From Computer Desktop Encyclopedia



KNN Search on Metric Space

- As known, on metric space, R-tree is elegant and efficient in finding top-k answers.





Challenge

- However, it is not easy to extend KNN search on metric space to road networks.
- Bottleneck 1: Calculating distance between two nodes
  - On metric space: O(1) time, O(1) space
  - On road network(graph):

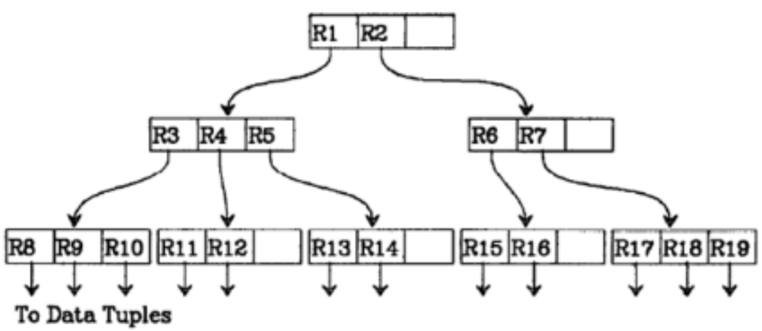
	Time	Space
Pre-compute all pairs	O(1)	$O(n^2)$
Improved Dijkstra	O(nlogn)	O(1)

Challenge

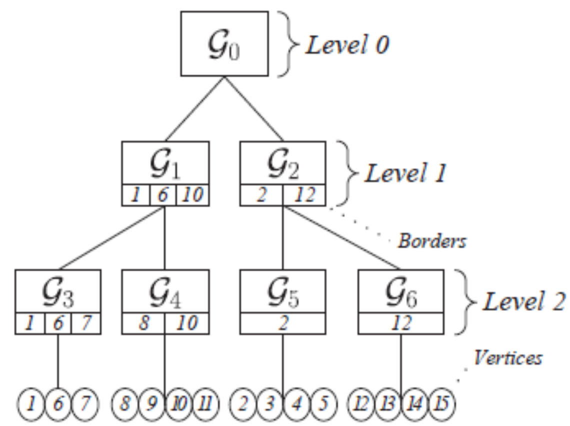
- Bottleneck 2: Pruning algorithm
  - On metric space:
    - By exploiting minimal bounding rectangle(MBR), R-tree can efficiently support top-k pruning with the help of best-first-search algorithm.
  - On road network:
    - However, NO SUCH index exists.
    - Best-first-search algorithm cannot be used.

G-tree

- What we want?
  - An elegant index similar to R-tree on road network.
    - 1. Balance tree structure;
    - 2. Exploit best-first-search to find top-k answers;
    - 3. Good extensibility (keyword search, etc.).



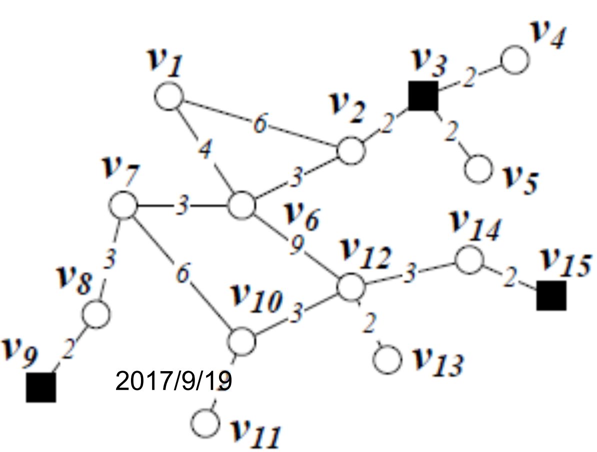
R-tree

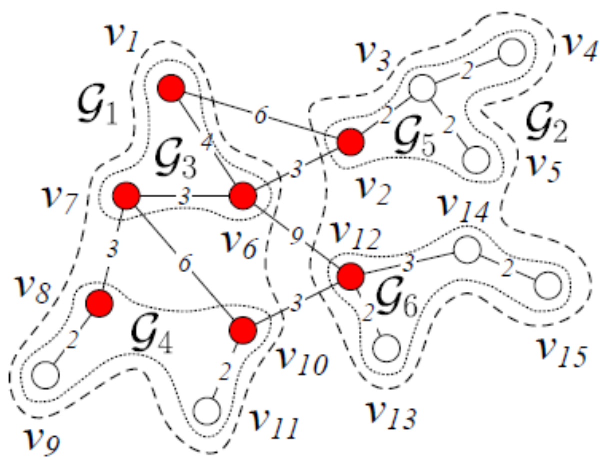


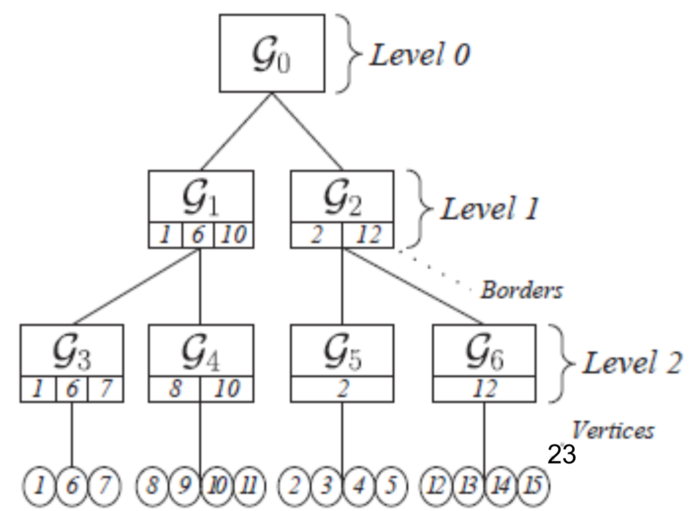
G-tree

G-tree

- The G-tree is constructed by recursively partitioning the road network into sub-networks and each G-tree node corresponds to a sub-network.
- Each leaf node contains no more than τ nodes.
- We employ METIS to produce balancing partition and minimize the cross-edge between sub-graphs.

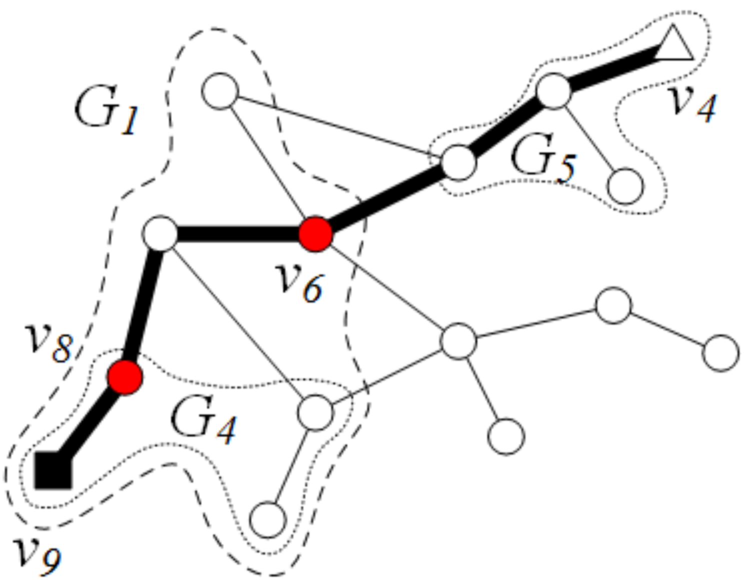






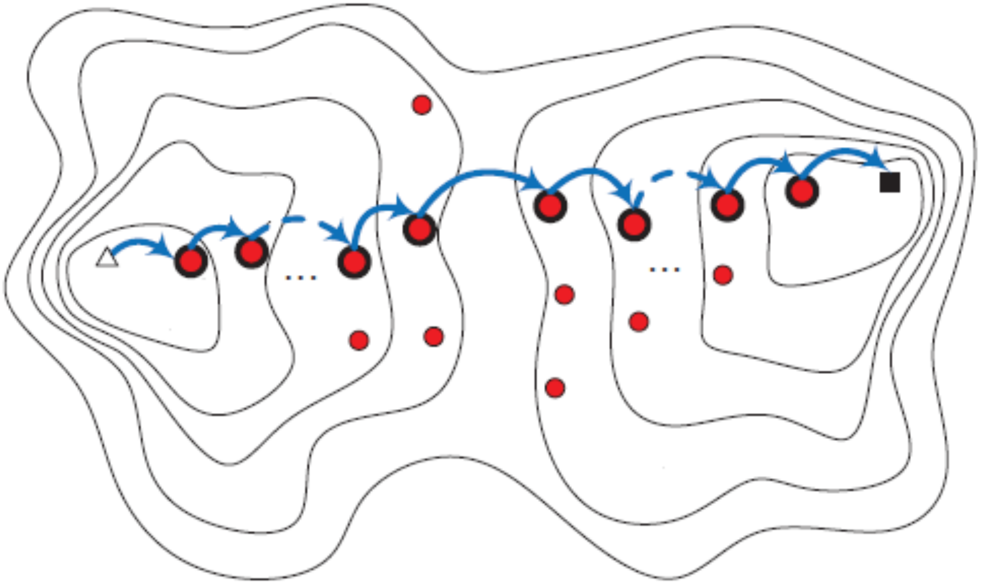
Border

- One type of nodes is important: BORDER.
- Border is the entry or exit of one sub-network, i.e. any path enters or exits one sub-network MUST bypass at least one border of such sub-network.



Border

- On a hierarchy graph, one path will bypass a series of borders which respectively belongs to one tree node on the G-tree.



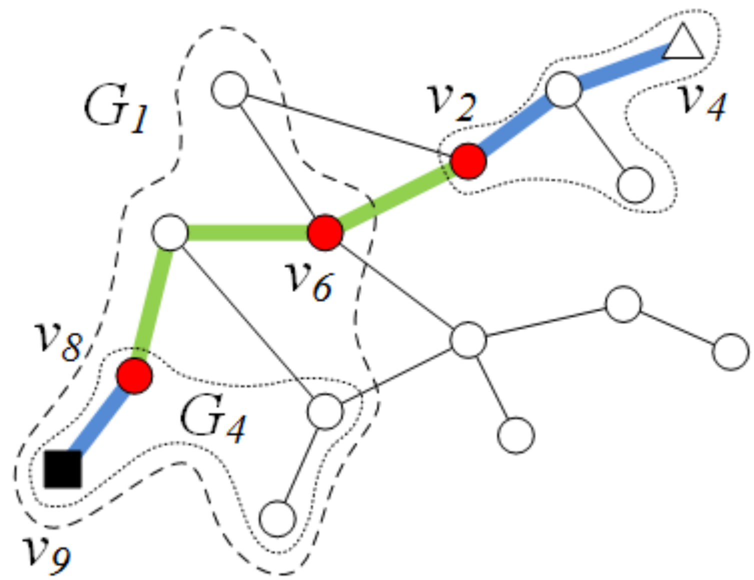
Distance Matrix

- A path on graph can be assembled segment by segment with only TWO types of distances:
  - **1. <u>Distances between two borders</u>(within the same level or adjacent level of G-tree)**

    $v_2 \leftrightarrow v_6$, $v_6 \leftrightarrow v_8$

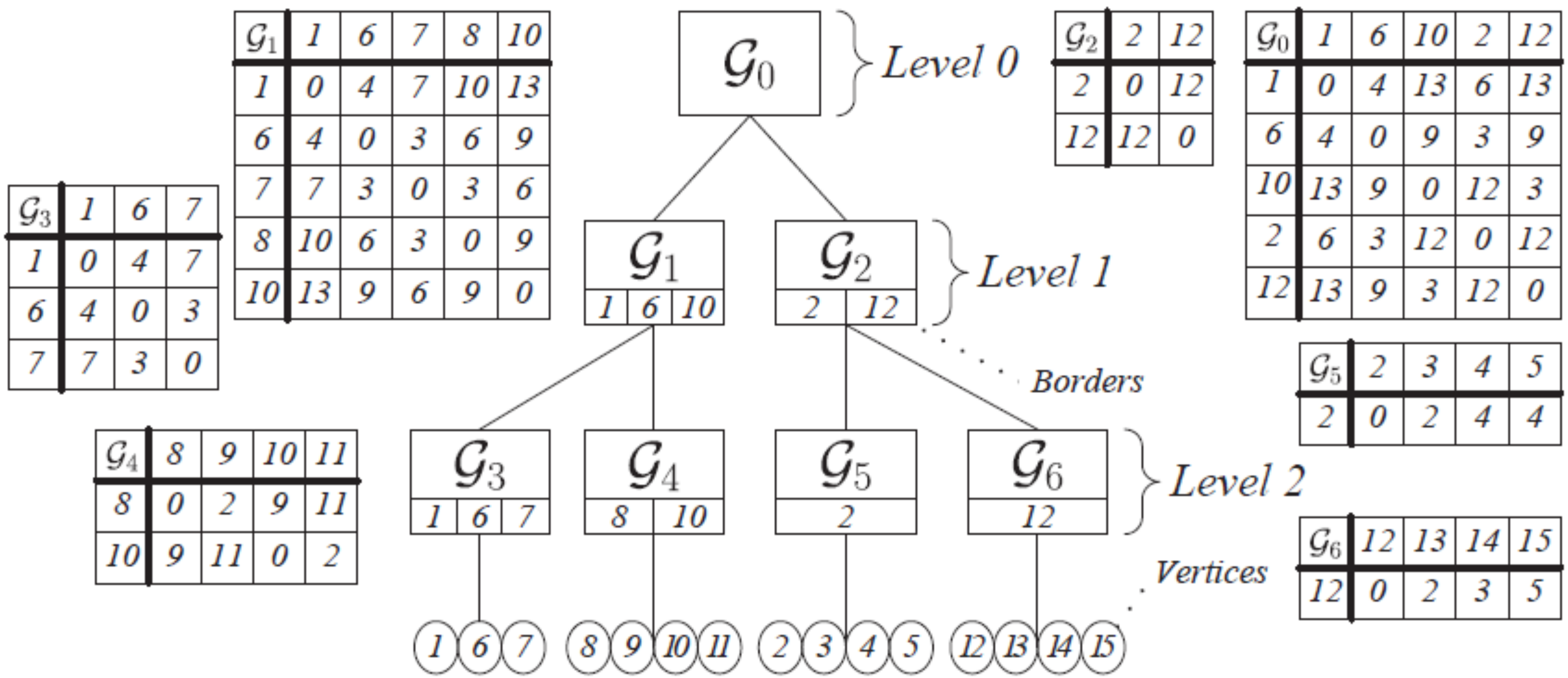
  - **2. <u>Distances between leaves' nodes and borders</u>(within leaf nodes)**

    $v_4 \leftrightarrow v_2$, $v_8 \leftrightarrow v_9$



Distance Matrix

- Thus, we store a distance matrix on each node of G-tree.



$\mathcal{G}_1$	1	6	7	8	10
1	0	4	7	10	13
6	4	0	3	6	9
7	7	3	0	3	6
8	10	6	3	0	9
10	13	9	6	9	0

$\mathcal{G}_2$	2	12
2	0	12
12	12	0

$\mathcal{G}_0$	1	6	10	2	12
1	0	4	13	6	13
6	4	0	9	3	9
10	13	9	0	12	3
2	6	3	12	0	12
12	13	9	3	12	0

$\mathcal{G}_3$	1	6	7
1	0	4	7
6	4	0	3
7	7	3	0

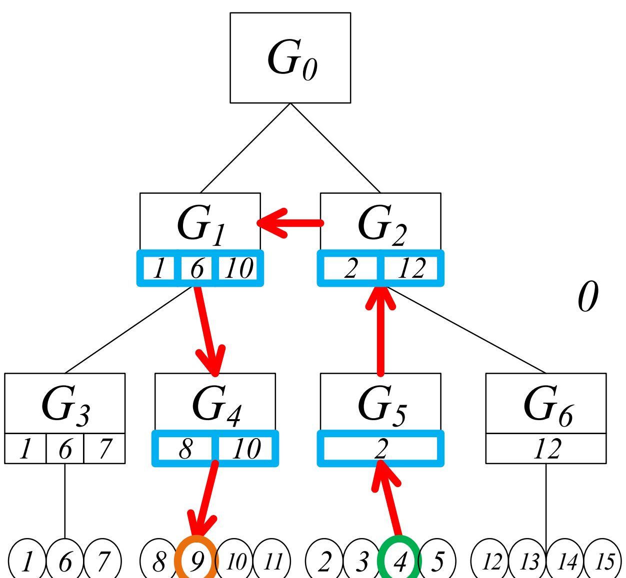
$\mathcal{G}_5$	2	3	4	5
2	0	2	4	4

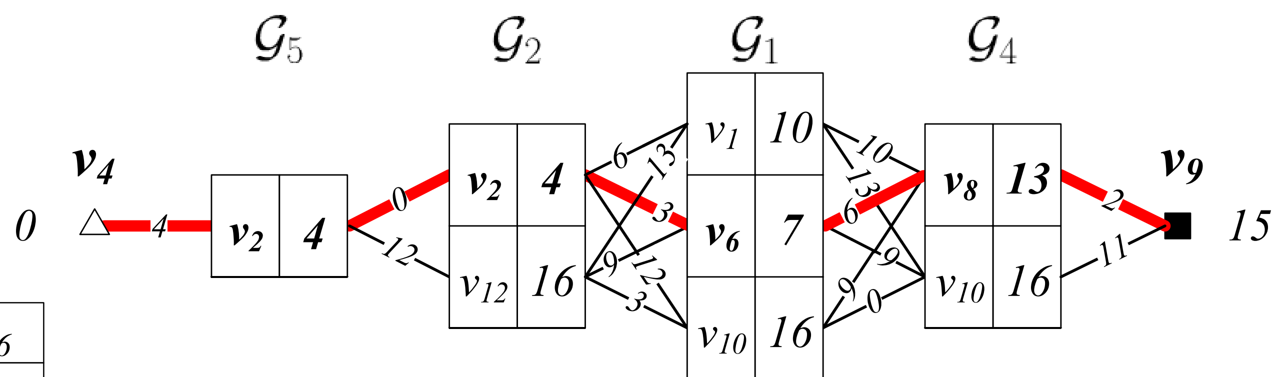
$\mathcal{G}_4$	8	9	10	11
8	0	2	9	11
10	9	11	0	2

$\mathcal{G}_6$	12	13	14	15
12	0	2	3	5

Single Pair Shortest Path

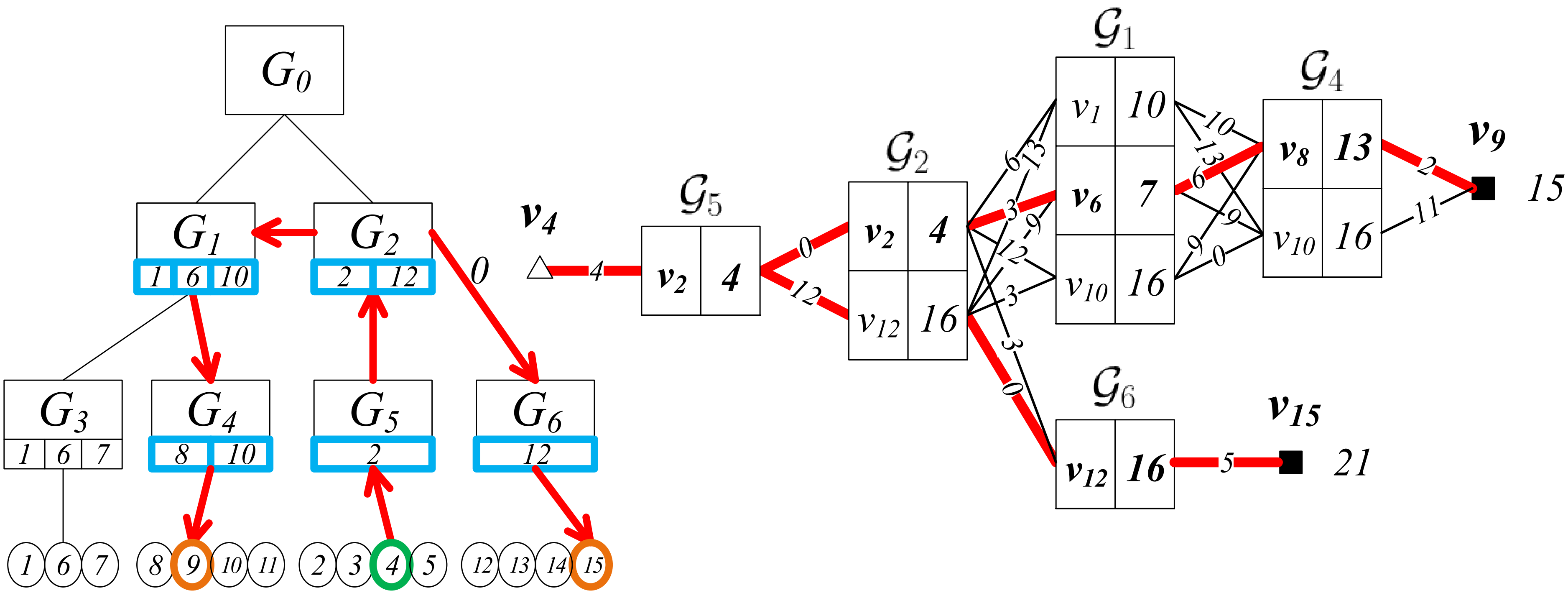
- Given two nodes *s* & *t*, the graph distance, denoted by *Mind(s,t)*, is calculated as follows:
  - 1. Find the path from *s*’ to *t*’ leaf node on G-tree.
  - 2. Use dynamic programming algorithm to find the shortest path distance.





Single Source Shortest Path

- Given two paths started from one node, e.g. v_4 to v_9/v_{15}, they might share a section of intermediate paths. Dynamic programming is applied as well.



KNN Search

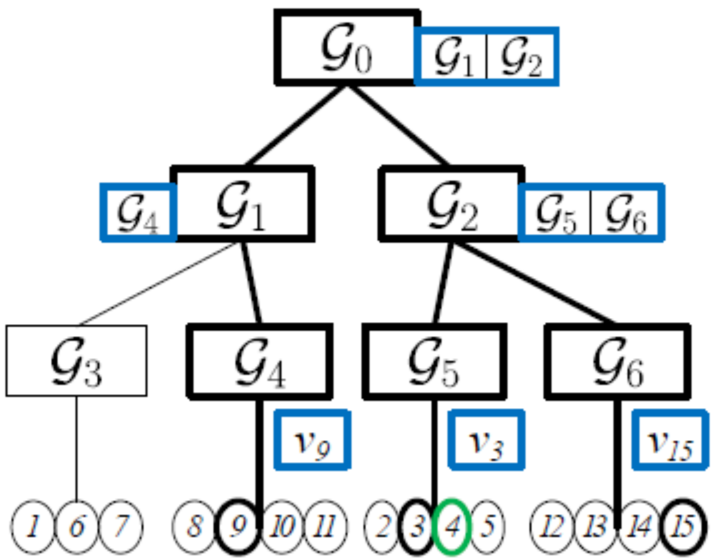
- We enable best-first-search algorithm on G-tree to retrieve top-k answers.
- **[Best-First-Search]**
  - Organize the promising nodes/objects in a priority-queue *Q* ranked by the minimal distance from query location to such node/object.
  - Iteratively dequeue the head element *e* of the priority-queue:
    - If *e* is a tree node, add *e*'s children into *Q*;
    - If e is an object, thus, *e* is the nearest object so far, then we add *e* to the result list.

It's Proven Good

KNN Search: Occurrence List

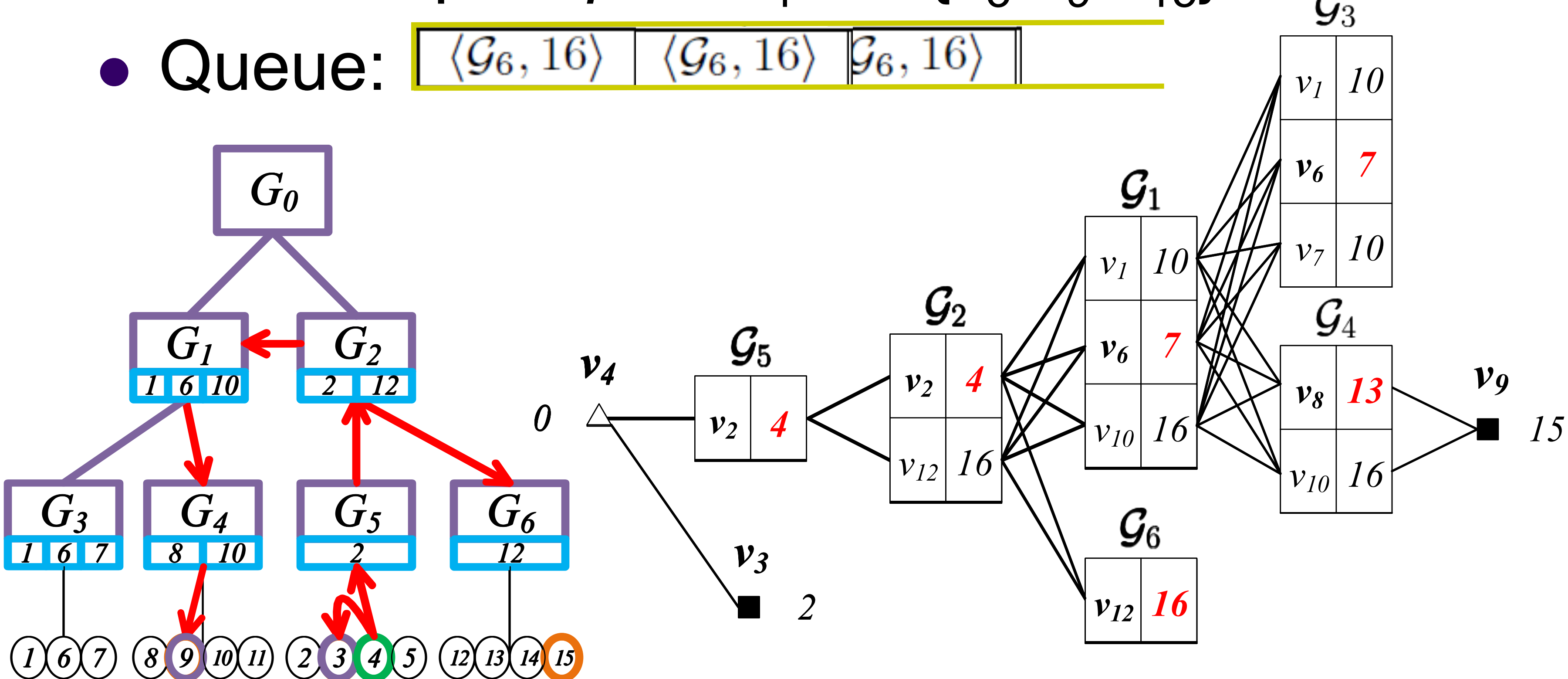
- Occurrence list (on each node) is an indicator for quickly locating promising nodes/objects:
  - For non-leaf node: composed of child nodes' ID of which descendants contain candidate objects;
  - For leaf node: cor

O={v_3,v_9,v_{15}}



KNN Search— An Example

- An example: *qloc*=v_4, O={v_3,v_9,v_{15}}, K=2
- Queue: $\langle \mathcal{G}_6, 16 \rangle$ $\langle \mathcal{G}_6, 16 \rangle$ $\mathcal{G}_6, 16 \rangle$



Top-2 Result: $\langle v_3, 2 \rangle$ $\langle v_9, 15 \rangle$

Superiority of G-tree

- We maintain the intermediate result *Mind(qloc,border$_i$)* on each node of G-tree, hence, each tree node is accessed only once.
- Our method significantly reduces the overhead for calculating minimal boundary, as we can obtain the minimal boundary from the intermediate result incidentally.
- By the use of best-first-search algorithm, G-tree has significant pruning power.