Graph Distances in the Streaming Model

Joan Feigenbaum
Sampath Kannan
Andrew McGregor
Siddharth Suri
& Jian Zhang

The Streaming Model

The Streaming Model

- Classic Problem: Median Finding

The Streaming Model

- Classic Problem: Median Finding
- Parameters of the Model:
  - How much *memory*?
  - How many *passes*?
  - How much *computation time* between data elements?

The Streaming Model

- Classic Problem: Median Finding
- Parameters of the Model:
  - How much *memory*?
  - How many *passes*?
  - How much *computation time* between data elements?
- Statistics, Norms and Histograms…

The Streaming Model

- Classic Problem: Median Finding
- Parameters of the Model:
  - How much *memory*?
  - How many *passes*?
  - How much *computation time* between data elements?
- Statistics, Norms and Histograms…
- What about graph problems?

Graph Streaming

Graph Streaming

- Consider an instance of a graph problem $G=(V,E)$

Graph Streaming

- Consider an instance of a graph problem $G=(V,E)$
- The edges are revealed to us in some arbitrary order $(e_1, e_2, e_3, \ldots)$

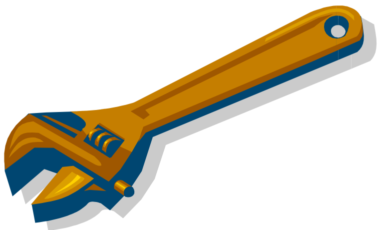
Graph Streaming

- Consider an instance of a graph problem $G=(V,E)$
- The edges are revealed to us in some arbitrary order (e_1, e_2, e_3, …)
- We focus on the memory limit O(n polylog n).

Overview

- Efficient Spanner Construction
- Lower Bounds for Approximation Factors
- Hardness of Constructing BFS

1. Spanner Construction



Spanner Construction

Spanner Construction

- *t*-Spanner of Graph *G*: Subgraph *H* where no distance is stretched by more than factor *t*.

Spanner Construction

- *t*-Spanner of Graph *G*: Subgraph *H* where no distance is stretched by more than factor *t*.

Spanner Construction

- *t*-Spanner of Graph *G*: Subgraph *H* where no distance is stretched by more than factor *t*.
- Main result: Randomized algorithm for log *n*-spanner using *O(n polylog n)* space and *O(polylog n)* time per edge.

Spanner Construction

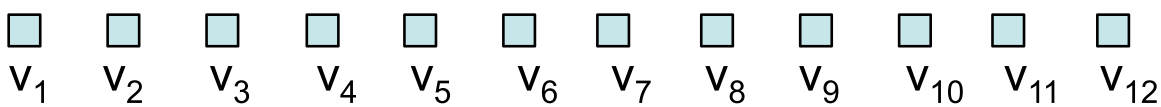
- *t*-Spanner of Graph *G*: Subgraph *H* where no distance is stretched by more than factor *t*.
- Main result: Randomized algorithm for log *n*-spanner using *O(n polylog n)* space and *O(polylog n)* time per edge.

Spanner Construction

- *t*-Spanner of Graph *G*: Subgraph *H* where no distance is stretched by more than factor *t*.
- Main result: Randomized algorithm for log *n*-spanner using *O(n polylog n)* space and *O(polylog n)* time per edge.
- Naive Algorithm: If an edge completes a short cycle, ignore it.

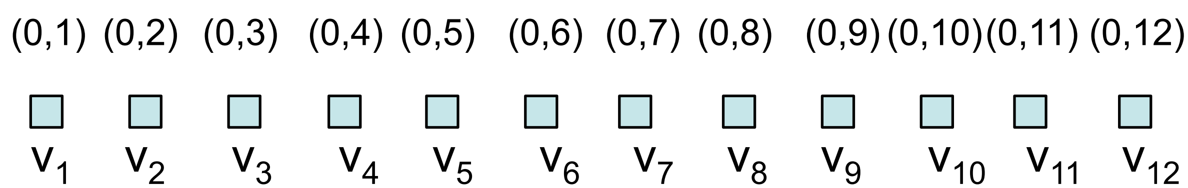
Our Algorithm for a 2*t*+1 Spanner

- Assign an initial set of labels L(*v*) to each vertex *v*:



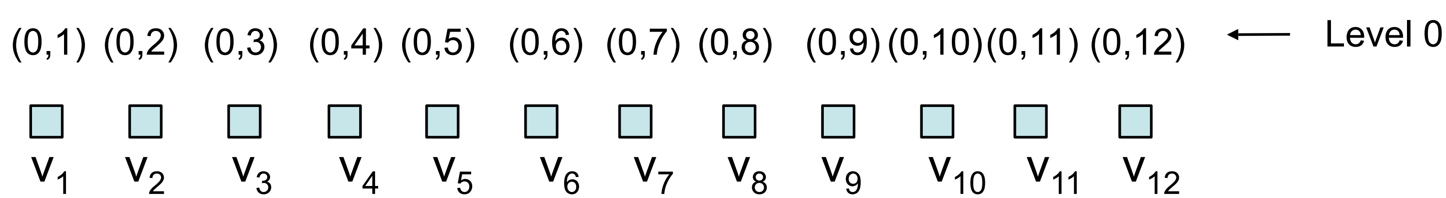
Our Algorithm for a 2*t*+1 Spanner

- Assign an initial set of labels L(*v*) to each vertex *v*:



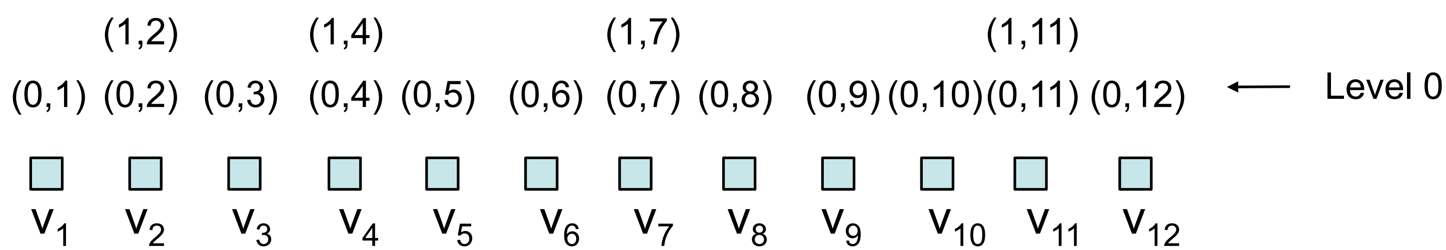
Our Algorithm for a 2*t*+1 Spanner

- Assign an initial set of labels L(*v*) to each vertex *v*:



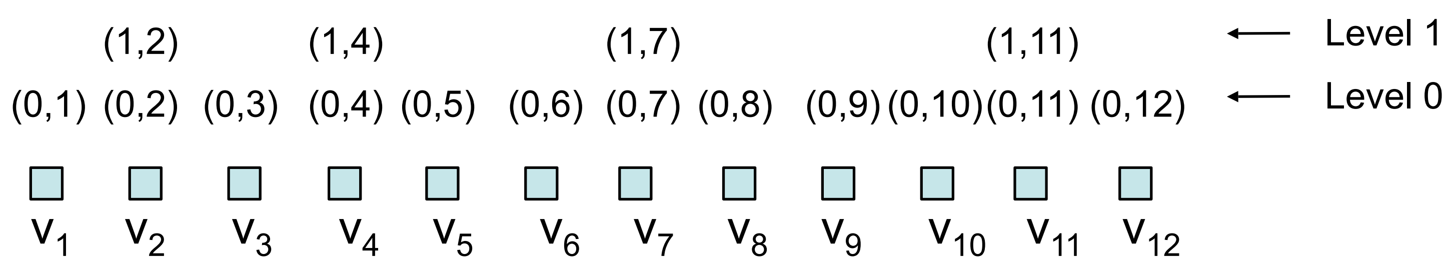
Our Algorithm for a 2*t*+1 Spanner

- Assign an initial set of labels L(*v*) to each vertex *v*:



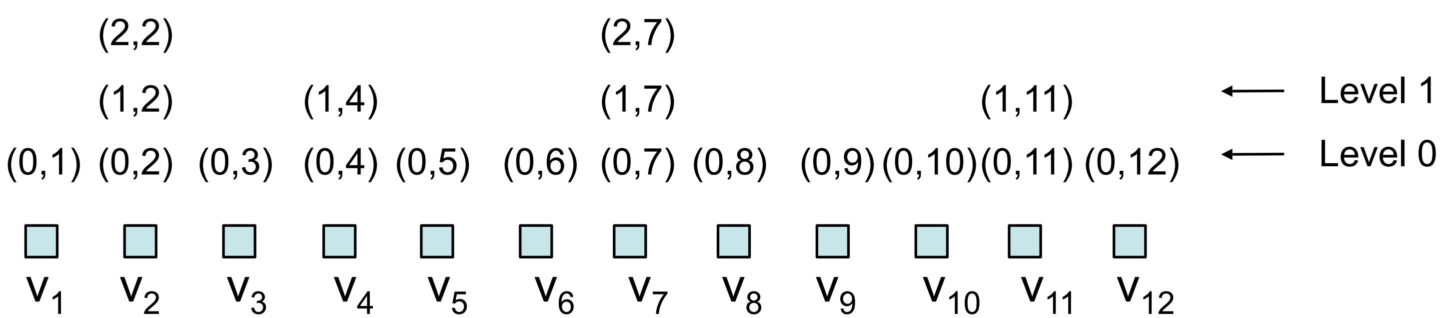
Our Algorithm for a 2*t*+1 Spanner

- Assign an initial set of labels L(*v*) to each vertex *v*:



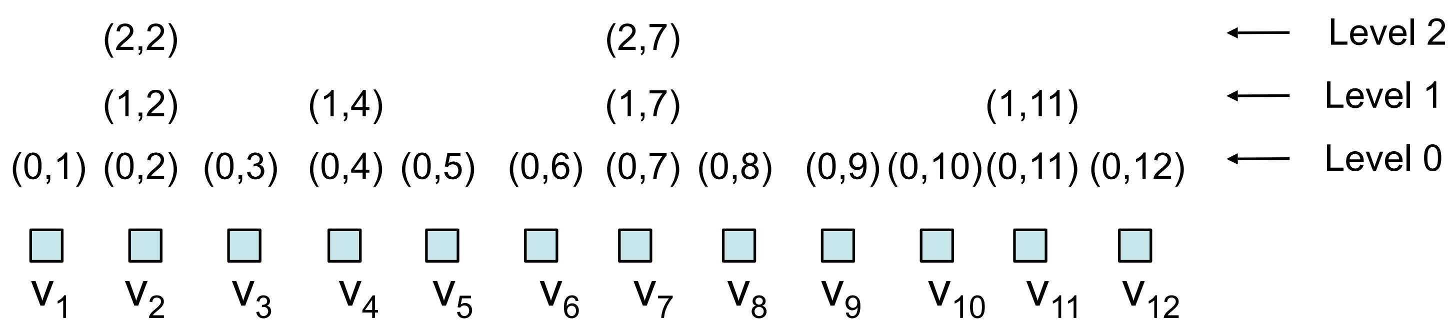
Our Algorithm for a 2*t*+1 Spanner

- Assign an initial set of labels L(*v*) to each vertex *v*:



Our Algorithm for a 2*t*+1 Spanner

- Assign an initial set of labels L(*v*) to each vertex *v*:



Our Algorithm for a $2t+1$ Spanner

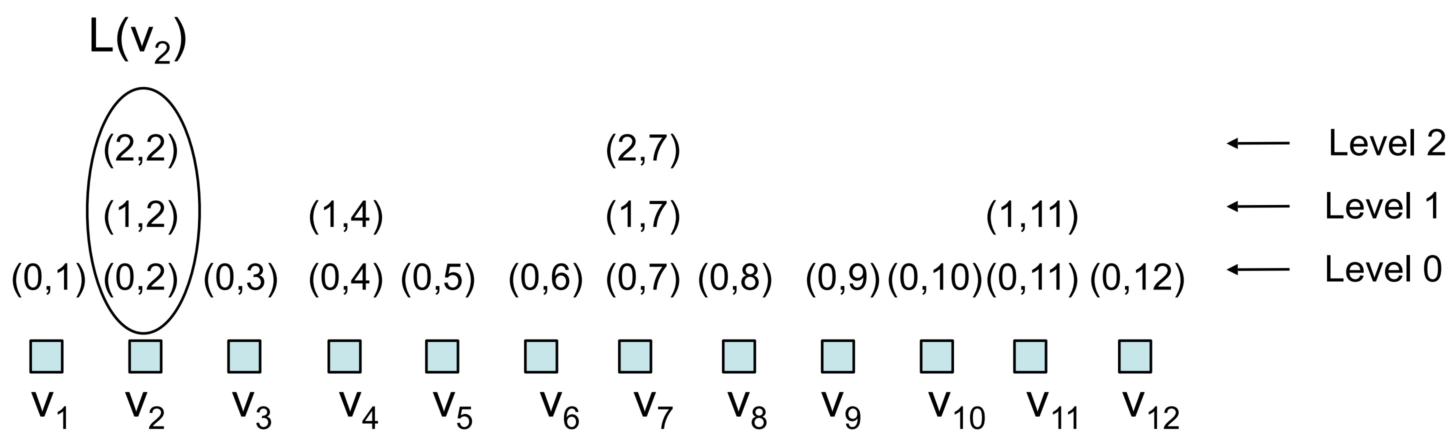
- Assign an initial set of labels L(*v*) to each vertex *v*:

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
Level 2		(2,2)					(2,7)					
Level 1		(1,2)		(1,4)			(1,7)				(1,11)	
Level 0	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)	(0,8)	(0,9)	(0,10)	(0,11)	(0,12)

- Stop at *top* level labels of height $t/2$

Our Algorithm for a 2*t*+1 Spanner

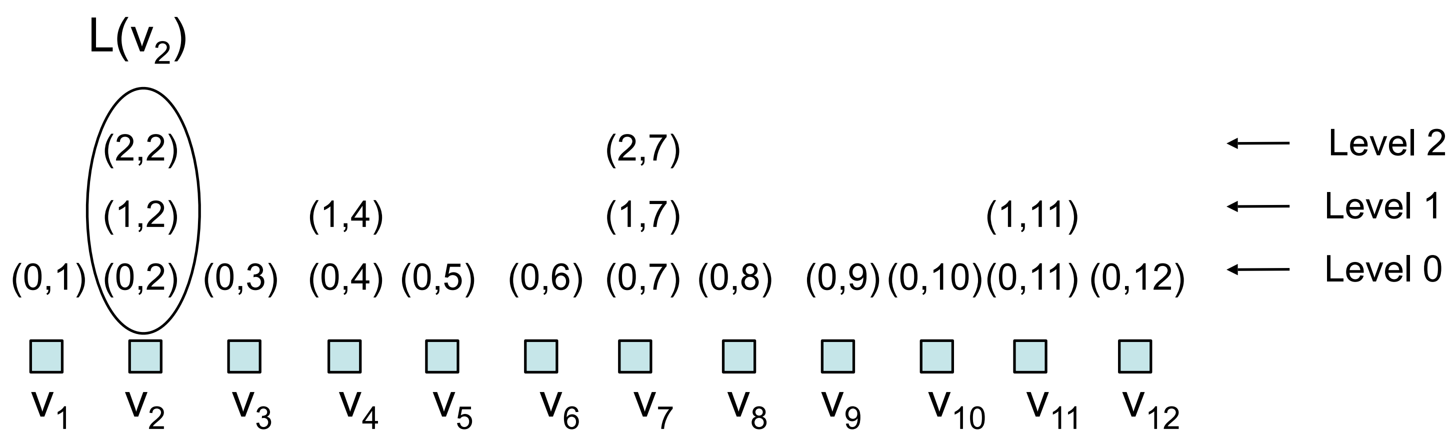
- Assign an initial set of labels L(*v*) to each vertex *v*:



- Stop at *top* level labels of height *t*/2

Our Algorithm for a 2*t*+1 Spanner

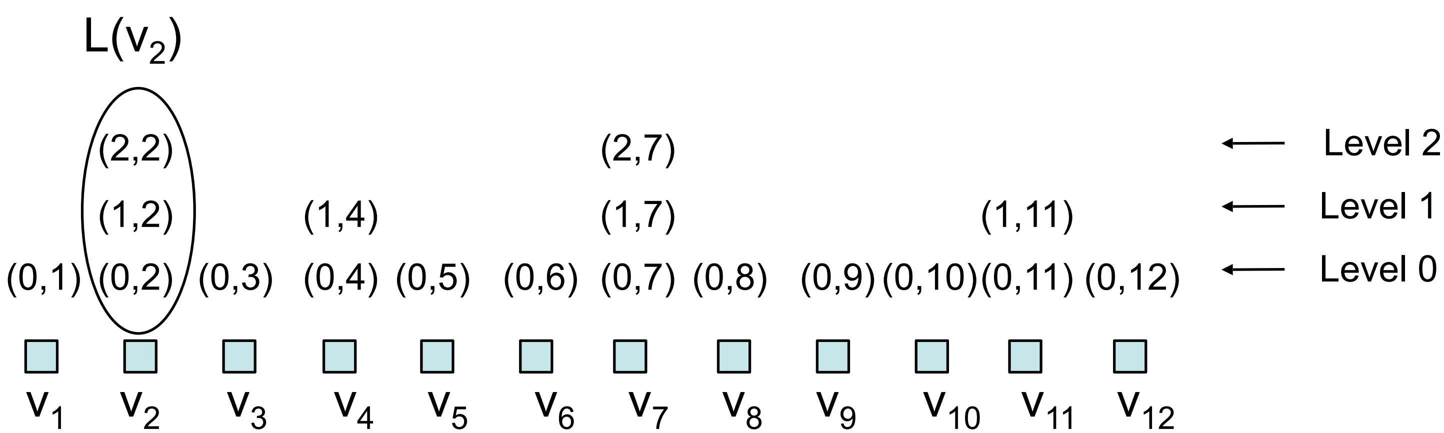
- Assign an initial set of labels L(*v*) to each vertex *v*:



- Stop at *top* level labels of height *t*/2
- C(*l*) = *cluster* of nodes with label *l*

Our Algorithm for a 2*t*+1 Spanner

- Assign an initial set of labels L(*v*) to each vertex *v*:



- Stop at *top* level labels of height *t*/2
- C(*l*) = *cluster* of nodes with label *l*
- As edges stream in we grow L(.) and C(.)

Types of edges in Spanner:

- Spanning trees for each C(*l*)
- Set T of edges connecting clusters at top level
- For each vertex v, set of edges M(*v*) to neighbours of *v*.

Invariants:

- Each vertex occurs in at most one cluster per level.
- If $l \in L(v)$ and $l' = Succ(l)$ then $l' \in L(v)$

On arrival of edge $e = (u, v)$…

On arrival of edge e =(u,v)…

- If u and v have a label in common then ignore.

On arrival of edge *e* =(*u*,*v*)…

- If *u* and *v* have a label in common then ignore.

On arrival of edge $e = (u,v)$…

- If u and v have a label in common then ignore.
- If u and v both have top layer labels then add to T (unless there's already an edge between these labels)

On arrival of edge *e* =(*u*,*v*)…

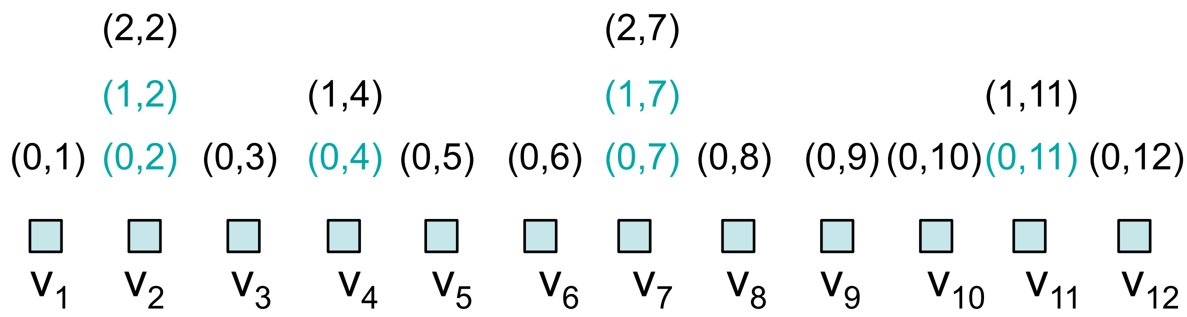
- If *u* and *v* have a label in common then ignore.
- If *u* and *v* both have top layer labels then add to *T* (unless there's already an edge between these labels)

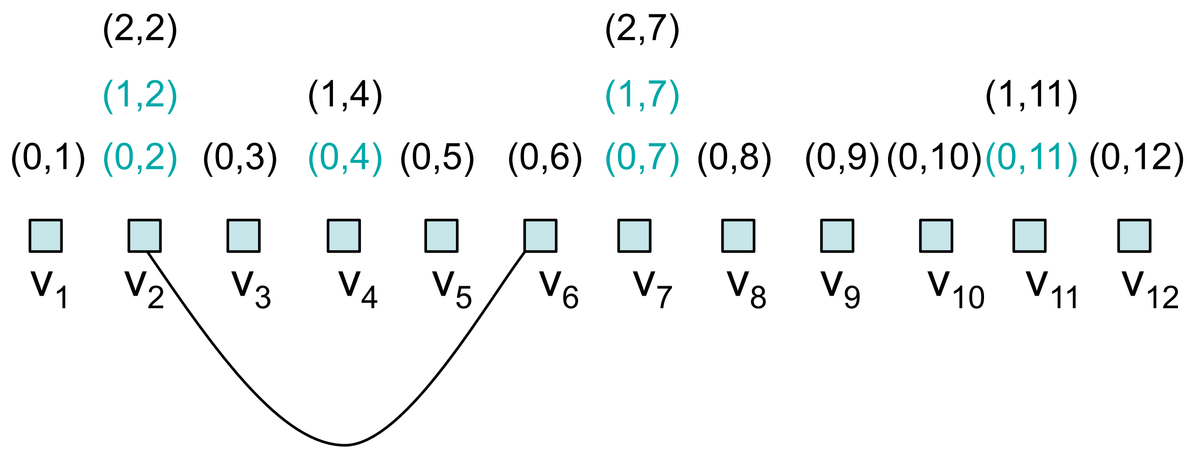
On arrival of edge $e = (u,v)$…

- If u and v have a label in common then ignore.
- If u and v both have top layer labels then add to T (unless there's already an edge between these labels)
- Let $L_u(v)$ = all labels in $L(u)$ higher than $L(v)$:

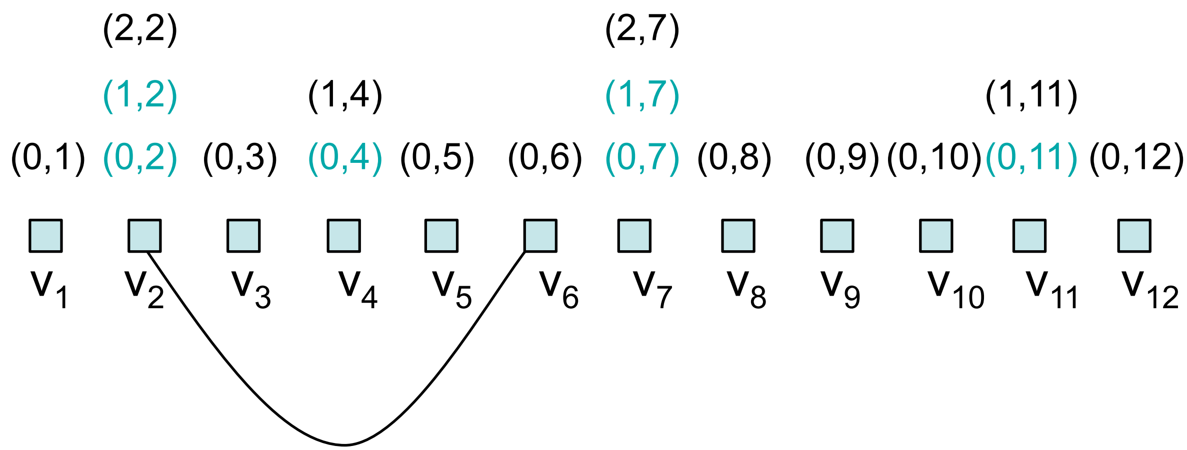
On arrival of edge *e* =(*u,v*)…

- If *u* and *v* have a label in common then ignore.
- If *u* and *v* both have top layer labels then add to *T* (unless there's already an edge between these labels)
- Let $L_u(v)$ = all labels in L(*u*) higher than L(*v*):
  - If $L_u(v)$ contains a selected label *l*, choose the smallest and add all the successors (*l'*, *l"…*) of *l* to *L(v).* Add *e* to C(*l'), C*(*l"*)...
  - If $L_u(v)$ doesn't contains a selected label, add *e* to M(*v*) if it doesn't contain a *u'* with the same label as *u.*


(2,2)
(2,7)
(1,2)
(1,4)
(1,7)
(1,11)
(0,1) (0,2) (0,3) (0,4) (0,5) (0,6) (0,7) (0,8) (0,9) (0,10) (0,11) (0,12)
v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12



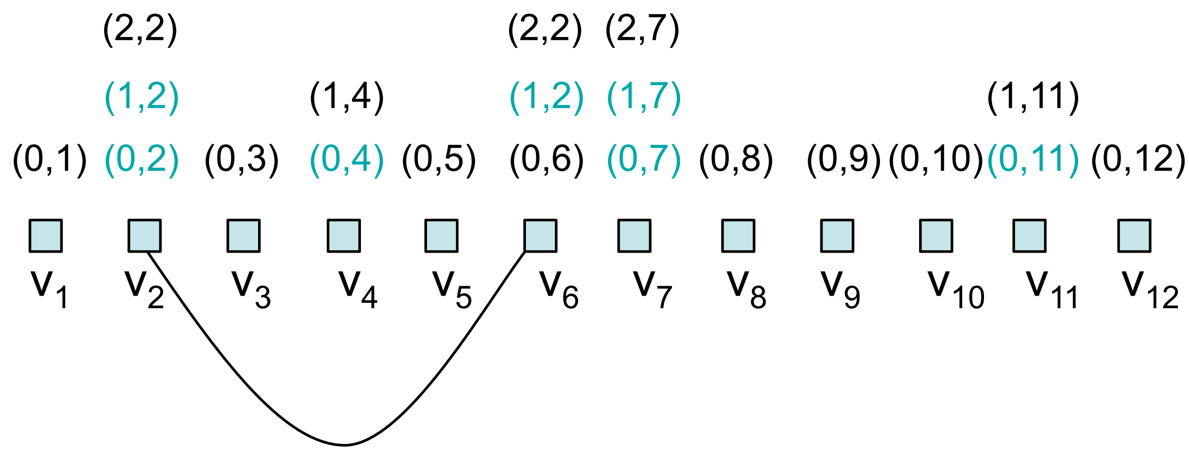
(v_2, v_6): $L_{v2}(v_6) = \{(0,2),(1,2),(2,2)\}$



(v_2,v_6): $L_{v2}(v_6) = \{(0,2),(1,2),(2,2)\}$

$L(v_6) = L(v_6) \cup \{(1,2),(2,2)\}$.

Add (v_2,v_6) to spanning trees for clusters C(1,2) and C(2,2).



(v_2,v_6): $L_{v2}(v_6) = \{(0,2),(1,2),(2,2)\}$

$L(v_6) = L(v_6) \cup \{(1,2),(2,2)\}$.

Add (v_2,v_6) to spanning trees for clusters C(1,2) and C(2,2).

Proofs: Graph Created is Sparse

Proofs: Graph Created is Sparse

- |T| = O(*n*) w.h.p:

Proofs: Graph Created is Sparse

- |T| = O(*n*) w.h.p:
  With high probability, # of clusters at top level is O(√n).

Proofs: Graph Created is Sparse

- |T| = O(*n*) w.h.p:
  With high probability, # of clusters at top level is $O(\sqrt{n})$.

Proofs: Graph Created is Sparse

- $|T| = O(n)$ w.h.p:
  With high probability, # of clusters at top level is $O(\sqrt{n})$.

- Total size of all cluster trees is $O(t\ n)$:

Proofs: Graph Created is Sparse

- $|T| = O(n)$ w.h.p:
  With high probability, # of clusters at top level is $O(\sqrt{n})$.

- Total size of all cluster trees is $O(t\ n)$:
  Each vertex appears at most in one cluster at each level.

Proofs: Graph Created is Sparse

- |T| = O(*n*) w.h.p:
  With high probability, # of clusters at top level is O(√n).

- Total size of all cluster trees is O(*t n*):
  Each vertex appears at most in one cluster at each level.

Proofs: Graph Created is Sparse

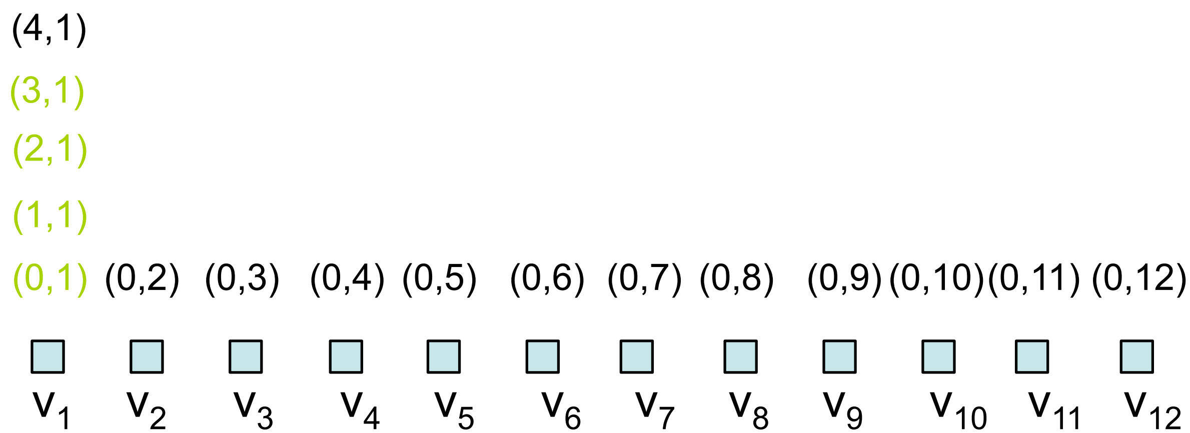
- $|T| = O(n)$ w.h.p:
  With high probability, # of clusters at top level is $O(\sqrt{n})$.
- Total size of all cluster trees is $O(t\ n)$:
  Each vertex appears at most in one cluster at each level.
- $|M(v)| = O(t\ n^{1/t} \log n)$ for each v w.h.p:

Proofs: Graph Created is Sparse

- |T| = O(n) w.h.p:
  With high probability, # of clusters at top level is O(√n).
- Total size of all cluster trees is O($t\ n$):
  Each vertex appears at most in one cluster at each level.
- |M(v)| = O($t\ n^{1/t}$ log n) for each v w.h.p:
  Each edge added to M(v) corresponds to an unselected label… hence # has geometric distribution.

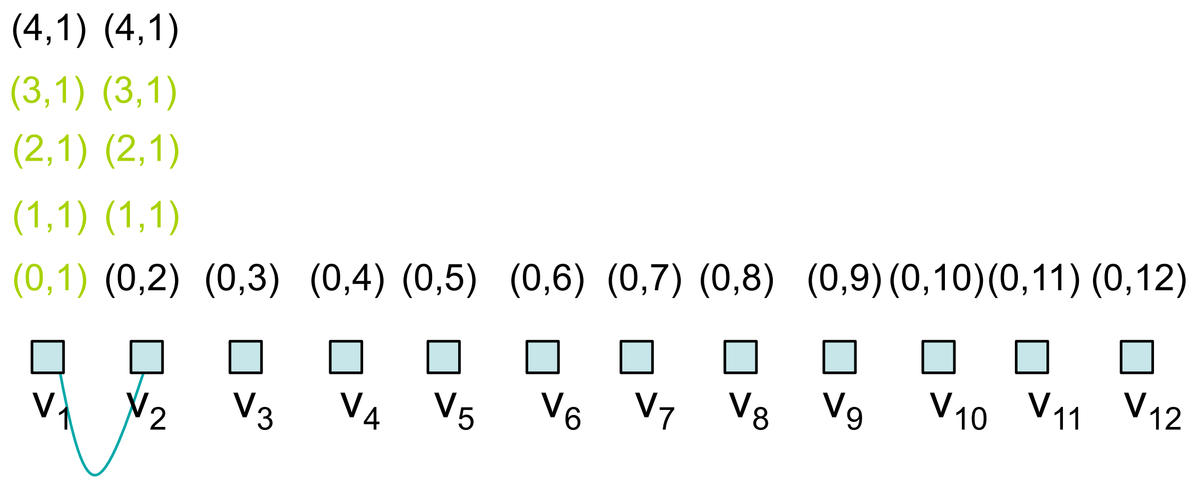
Proofs: Graph is a $2t+1$ spanner

- Spanning tree of i^{th} level cluster has diameter $\leq 2i$



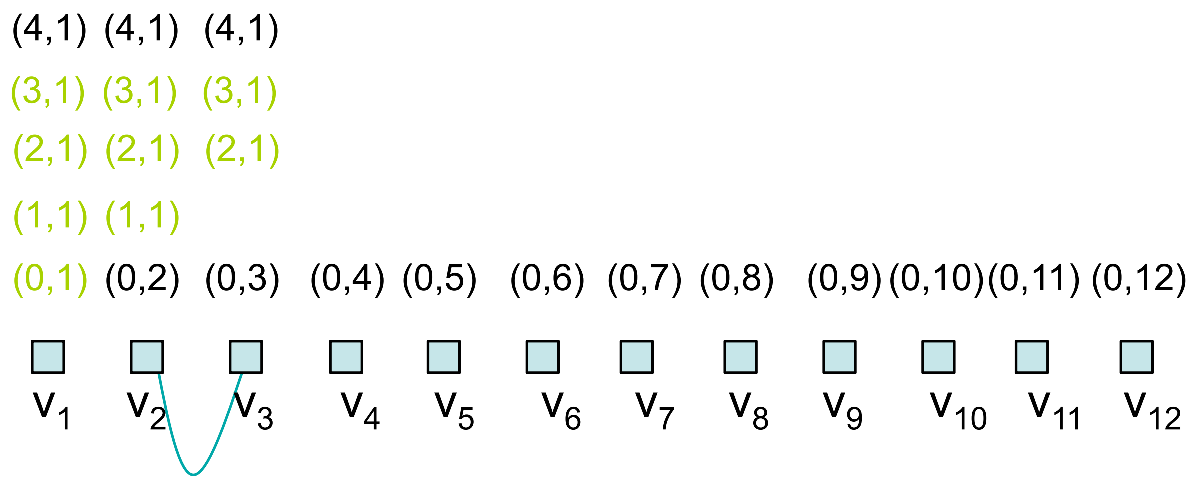
Proofs: Graph is a 2*t*+1 spanner

- Spanning tree of *i*th level cluster has diameter ≤ 2*i*



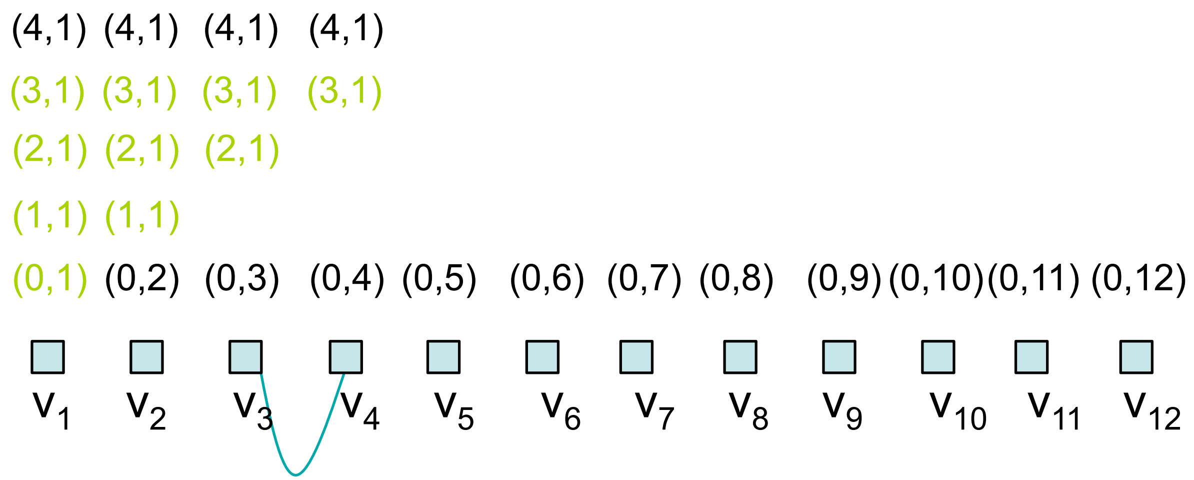
Proofs: Graph is a 2*t*+1 spanner

- Spanning tree of *i*th level cluster has diameter ≤ 2*i*



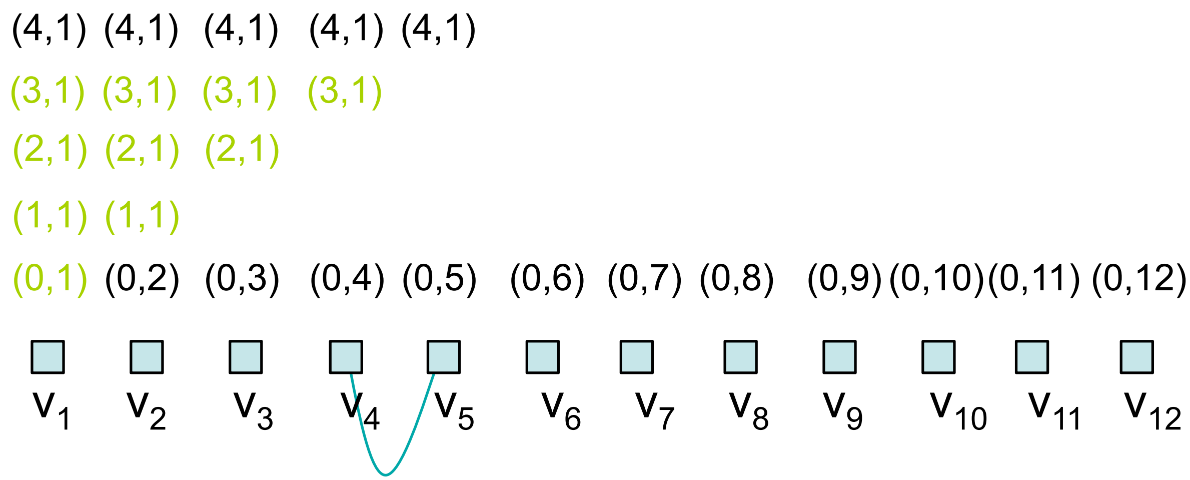
Proofs: Graph is a 2*t*+1 spanner

- Spanning tree of *i*th level cluster has diameter ≤ 2*i*



Proofs: Graph is a 2*t*+1 spanner

- Spanning tree of *i*th level cluster has diameter ≤ 2*i*



Proofs: Graph is a 2*t*+1 spanner

- Spanning tree of i^{th} level cluster has diameter ≤ 2*i*
- For each edge ignored there is a short detour:
  - If *u* and *v* have an label *l* in common they are ≤ *t* apart.
  - If *u* and *v* have top level labels then they are ≤ 2t+1 apart.
  - If (*u*,*v*) not added to M(v) there was an edge (*u'*,*v*) in M (*v*) with *u'* in same cluster as *u*. Then they are ≤ *t+1* apart.

2. Lower Bounds

2. Bounds

Estimating a Distance

Estimating a Distance

- Consider Zombie edges whose original presence can not be decided from the memory configuration.

Estimating a Distance

- Consider Zombie edges whose original presence can not be decided from the memory configuration.

Estimating a Distance

- Consider Zombie edges whose original presence can not be decided from the memory configuration.

- An edge (u,v) is *k-critical* if the shortest path from u to v in G\ (u,v) has length at least k.

Estimating a Distance

- Consider Zombie edges whose original presence can not be decided from the memory configuration.

- An edge (*u,v*) is *k-critical* if the shortest path from *u* to *v* in G\(*u,v*) has length at least *k.*

Estimating a Distance

- Consider Zombie edges whose original presence can not be decided from the memory configuration.
- An edge (*u,v*) is *k-critical* if the shortest path from *u* to *v* in G\(*u,v*) has length at least *k.*
- There exists a graph with $O(n^{1+\gamma})$ edges such that the majority of edges are $1/\gamma$-critical. Consider a random subgraph *H* formed by deleting at most half the $1/\gamma$-critical edges.

Estimating a Distance

- Consider Zombie edges whose original presence can not be decided from the memory configuration.
- An edge (*u,v*) is *k-critical* if the shortest path from *u* to *v* in G\(*u,v*) has length at least *k.*
- There exists a graph with $O(n^{1+\gamma})$ edges such that the majority of edges are $1/\gamma$-critical. Consider a random subgraph *H* formed by deleting at most half the $1/\gamma$-critical edges.

Estimating a Distance

- Consider Zombie edges whose original presence can not be decided from the memory configuration.
- An edge (u,v) is *k-critical* if the shortest path from u to v in $G \backslash (u,v)$ has length at least k.
- There exists a graph with $O(n^{1+\gamma})$ edges such that the majority of edges are $1/\gamma$-critical. Consider a random subgraph *H* formed by deleting at most half the $1/\gamma$-critical edges.

Estimating a Distance

- Consider Zombie edges whose original presence can not be decided from the memory configuration.
- An edge (u,v) is *k-critical* if the shortest path from u to v in $G\setminus(u,v)$ has length at least k.
- There exists a graph with $O(n^{1+\gamma})$ edges such that the majority of edges are $1/\gamma$-critical. Consider a random subgraph *H* formed by deleting at most half the $1/\gamma$-critical edges.

Estimating a Distance

- Consider Zombie edges whose original presence can not be decided from the memory configuration.
- An edge (*u,v*) is *k-critical* if the shortest path from *u* to *v* in G\(*u,v*) has length at least *k.*
- There exists a graph with $O(n^{1+\gamma})$ edges such that the majority of edges are $1/\gamma$-critical. Consider a random subgraph *H* formed by deleting at most half the $1/\gamma$-critical edges.

Estimating a Distance

- Consider Zombie edges whose original presence can not be decided from the memory configuration.
- An edge (*u,v*) is *k-critical* if the shortest path from *u* to *v* in G\(*u,v*) has length at least *k*.
- There exists a graph with $O(n^{1+\gamma})$ edges such that the majority of edges are $1/\gamma$-critical. Consider a random subgraph *H* formed by deleting at most half the $1/\gamma$-critical edges.

Estimating a Distance

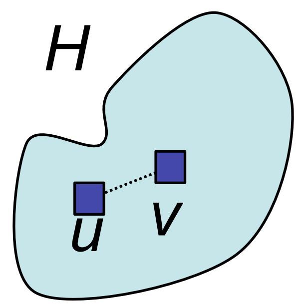
- Consider Zombie edges whose original presence can not be decided from the memory configuration.

- An edge (*u,v*) is *k-critical* if the shortest path from *u* to *v* in G\(*u,v*) has length at least *k.*

- There exists a graph with $O(n^{1+\gamma})$ edges such that the majority of edges are $1/\gamma$-critical. Consider a random subgraph *H* formed by deleting at most half the $1/\gamma$-critical edges.

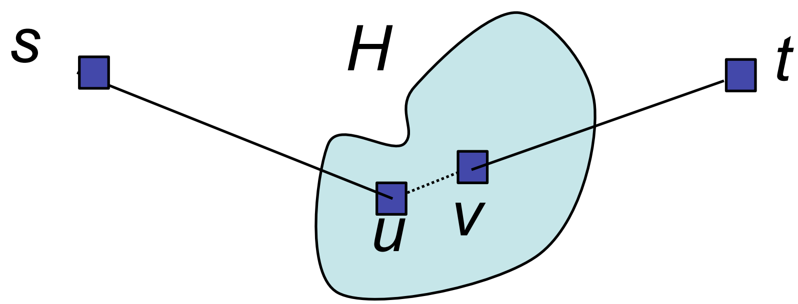
Estimating a Distance

- Consider Zombie edges whose original presence can not be decided from the memory configuration.
- An edge (u,v) is *k-critical* if the shortest path from u to v in G\ (u,v) has length at least k.
- There exists a graph with $O(n^{1+\gamma})$ edges such that the majority of edges are $1/\gamma$-critical. Consider a random subgraph H formed by deleting at most half the $1/\gamma$-critical edges.



Estimating a Distance

- Consider Zombie edges whose original presence can not be decided from the memory configuration.
- An edge (u,v) is *k-critical* if the shortest path from u to v in $G \setminus (u,v)$ has length at least k.
- There exists a graph with $O(n^{1+\gamma})$ edges such that the majority of edges are $1/\gamma$-critical. Consider a random subgraph H formed by deleting at most half the $1/\gamma$-critical edges.

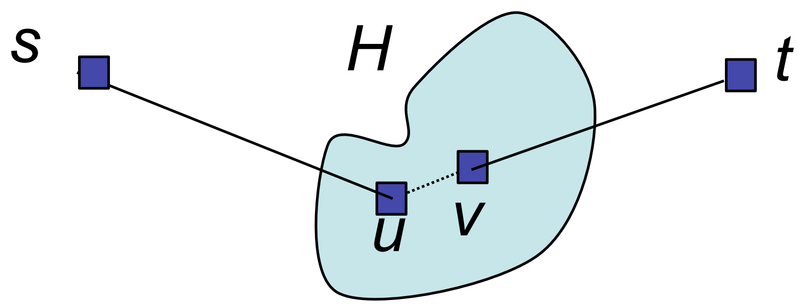


Estimating a Distance

- Consider Zombie edges whose original presence can not be decided from the memory configuration.

- An edge (u,v) is *k-critical* if the shortest path from u to v in G\(u,v) has length at least k.

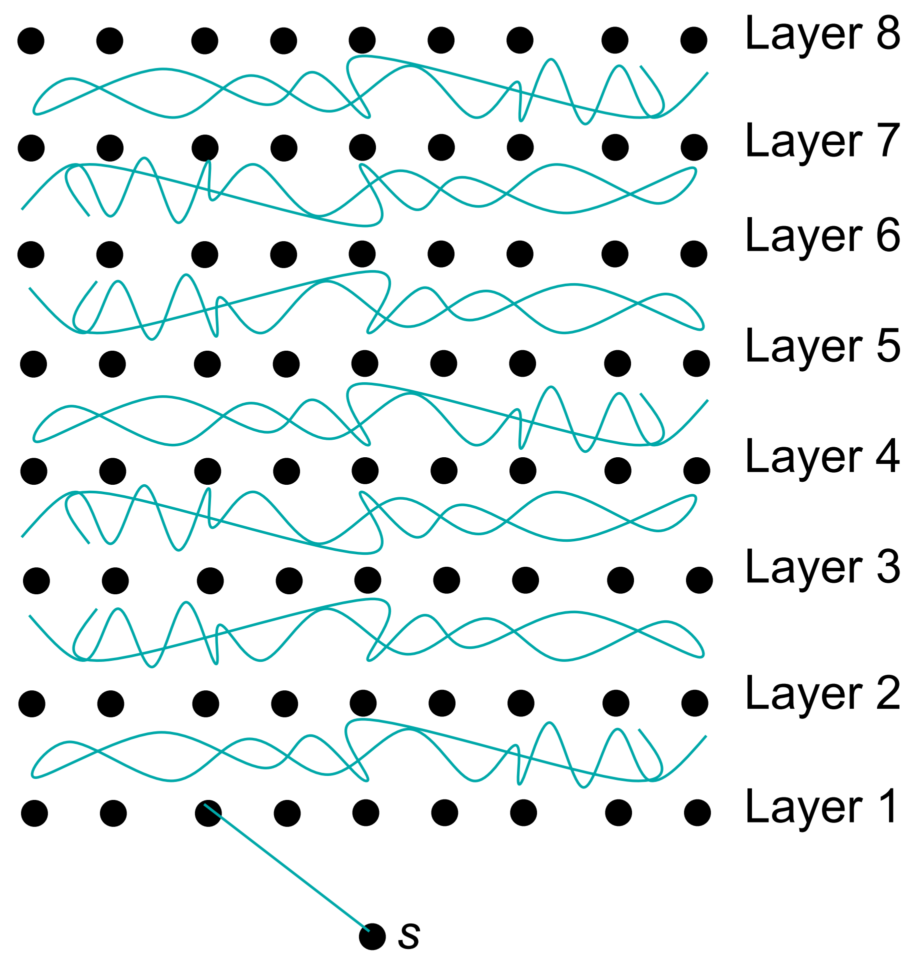
- There exists a graph with $O(n^{1+\gamma})$ edges such that the majority of edges are $1/\gamma$-critical. Consider a random subgraph H formed by deleting at most half the $1/\gamma$-critical edges.



- In one pass it is not possible to approximate the distance to better than $1/\gamma$ factor using space $o(n^{1+\gamma})$.

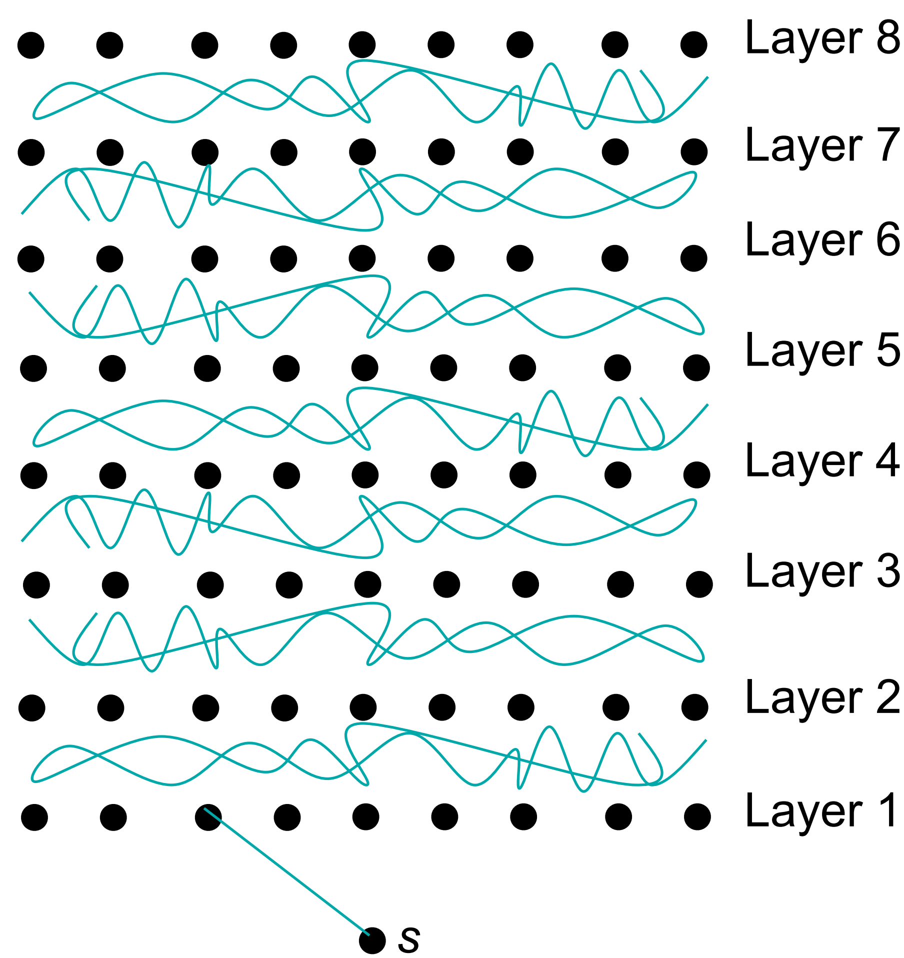
3. Hardness of Computing a BFS

Layered Graph



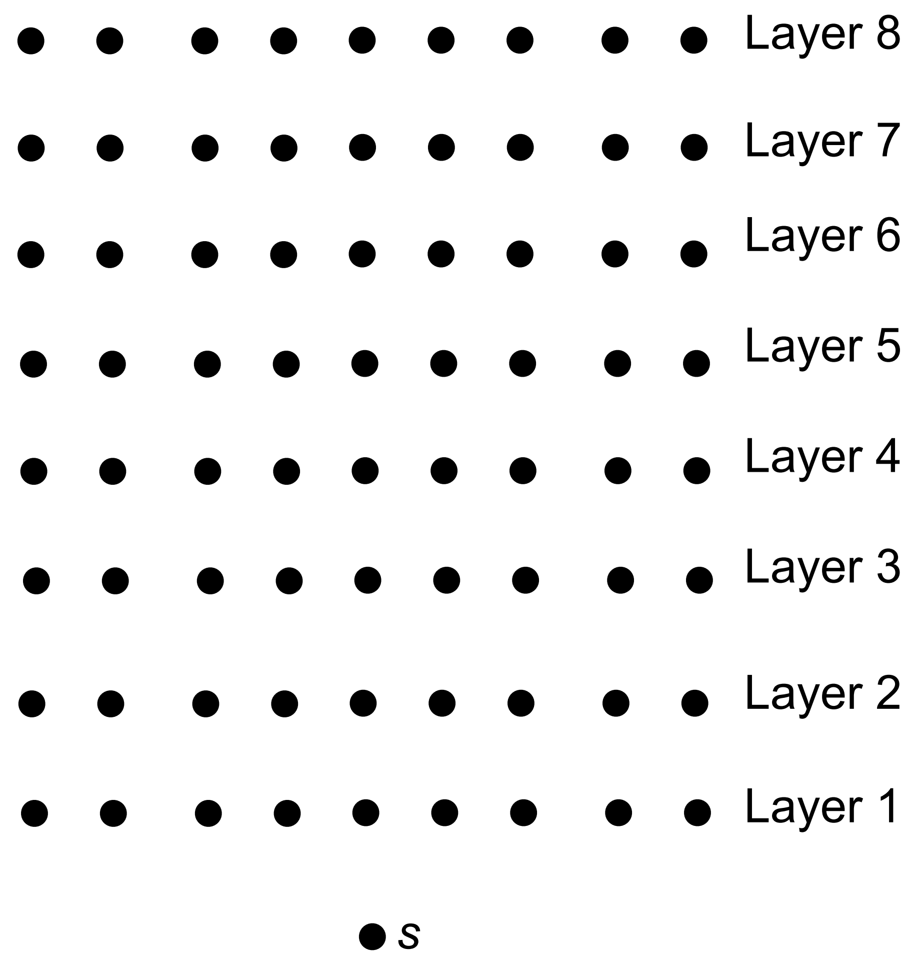
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



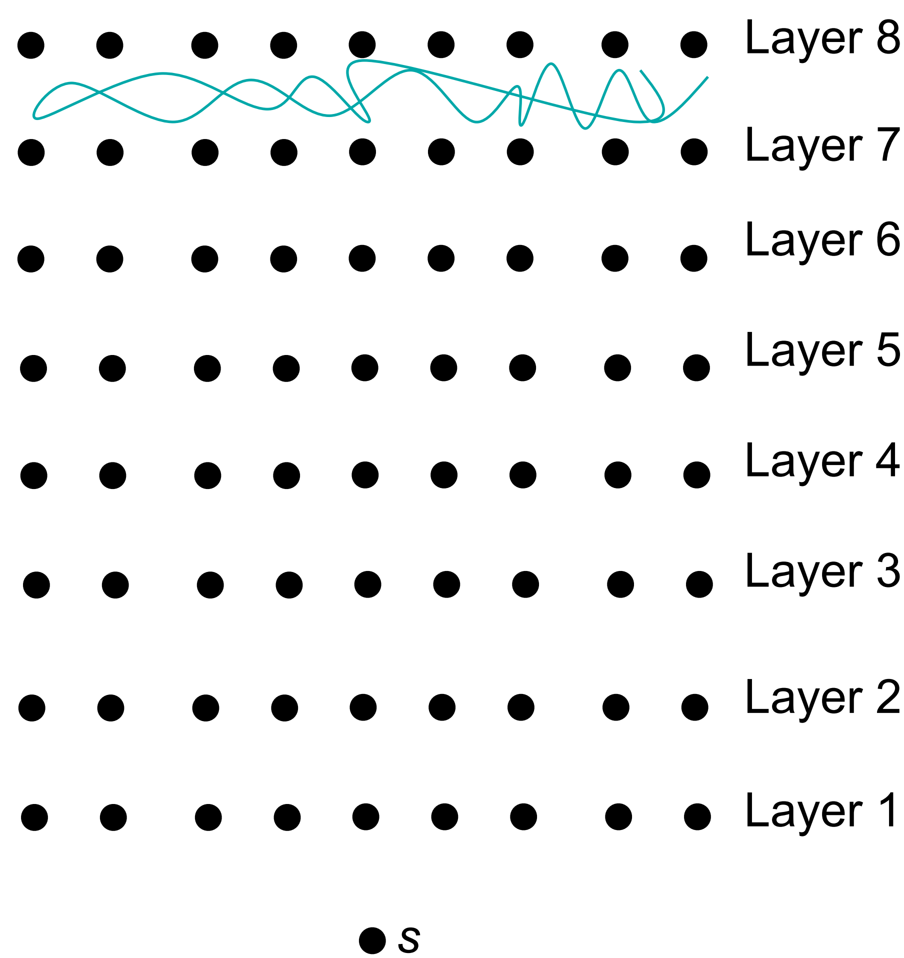
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



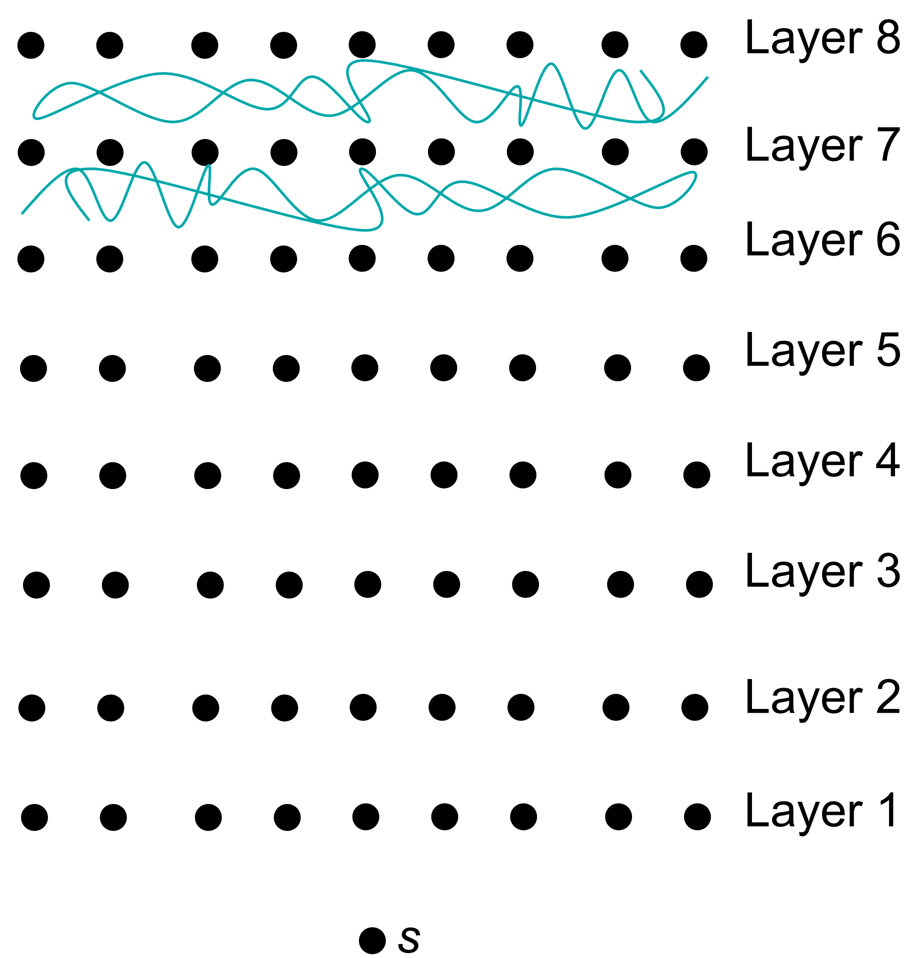
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



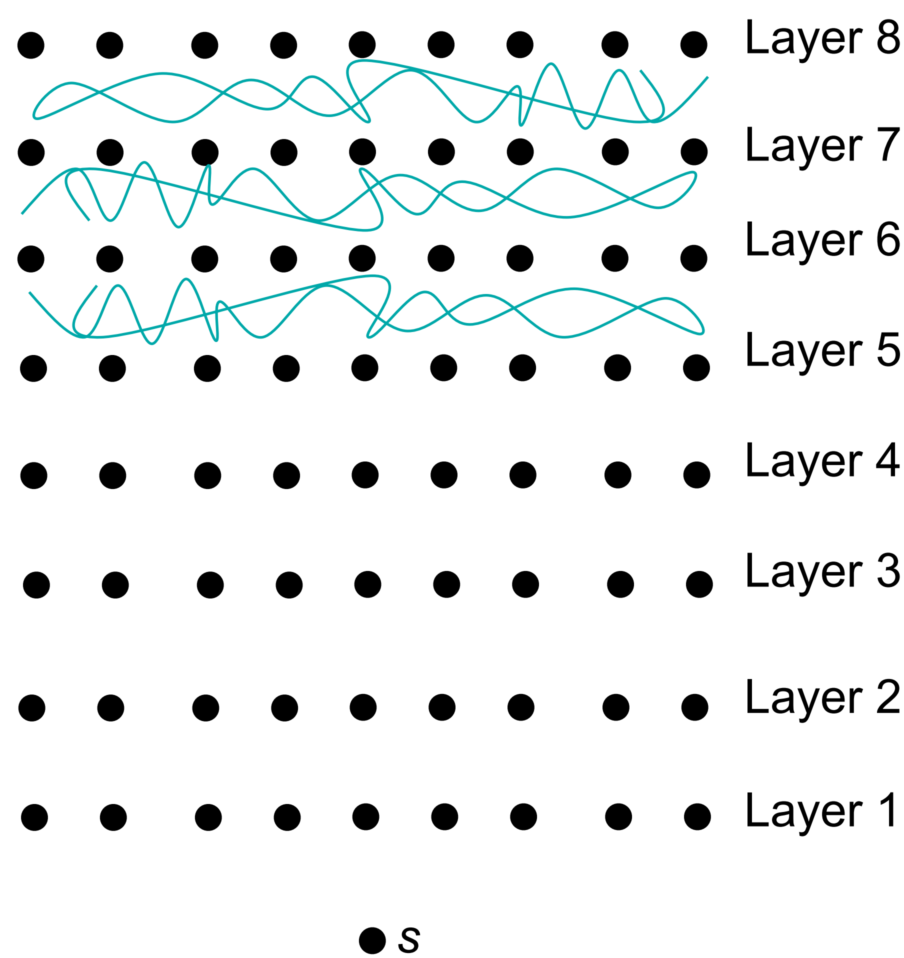
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



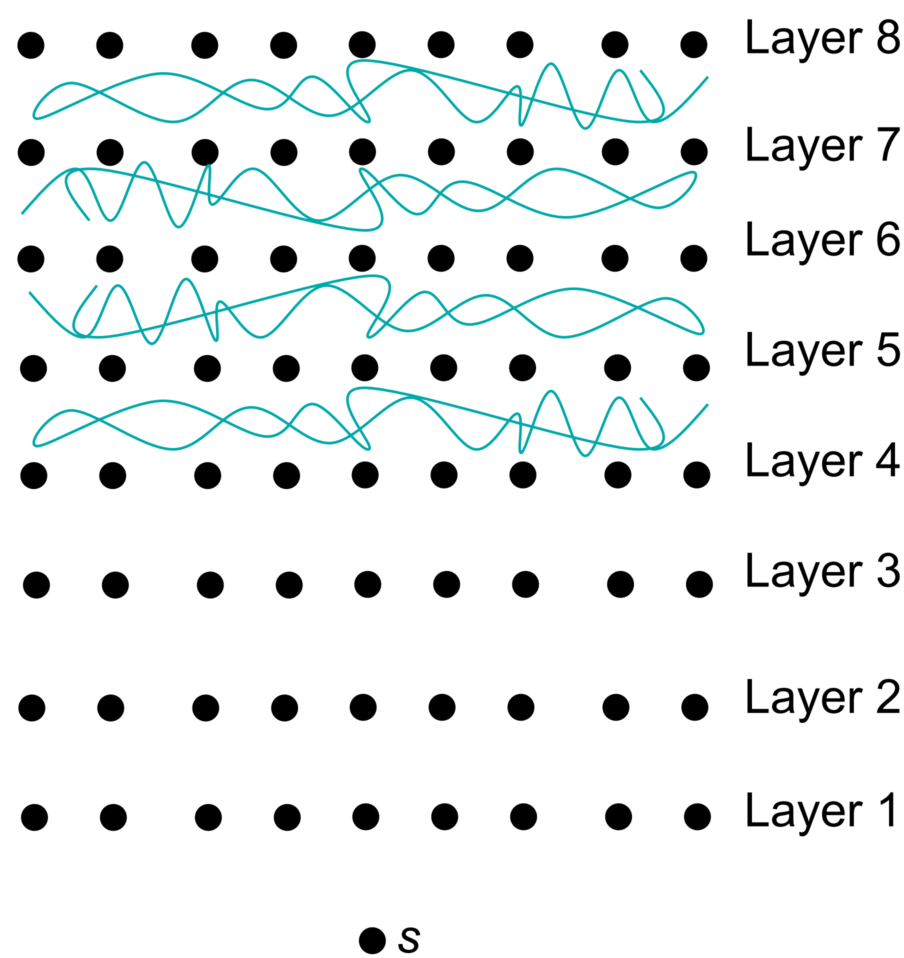
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



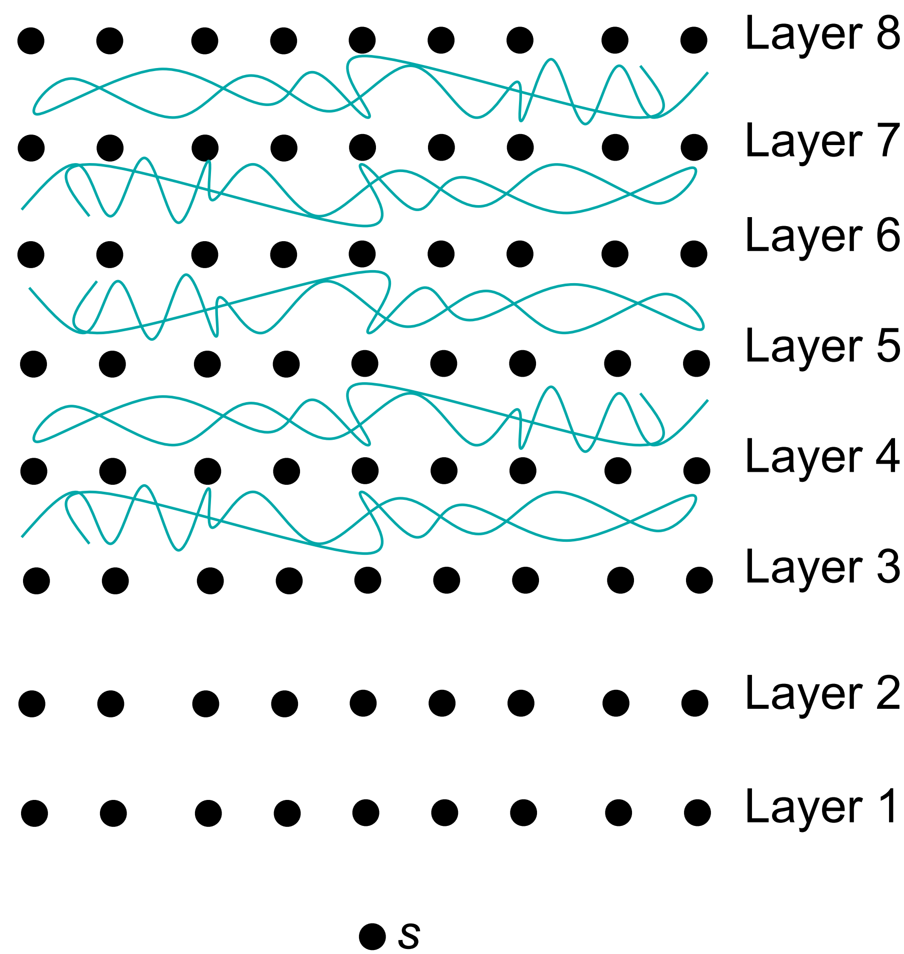
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



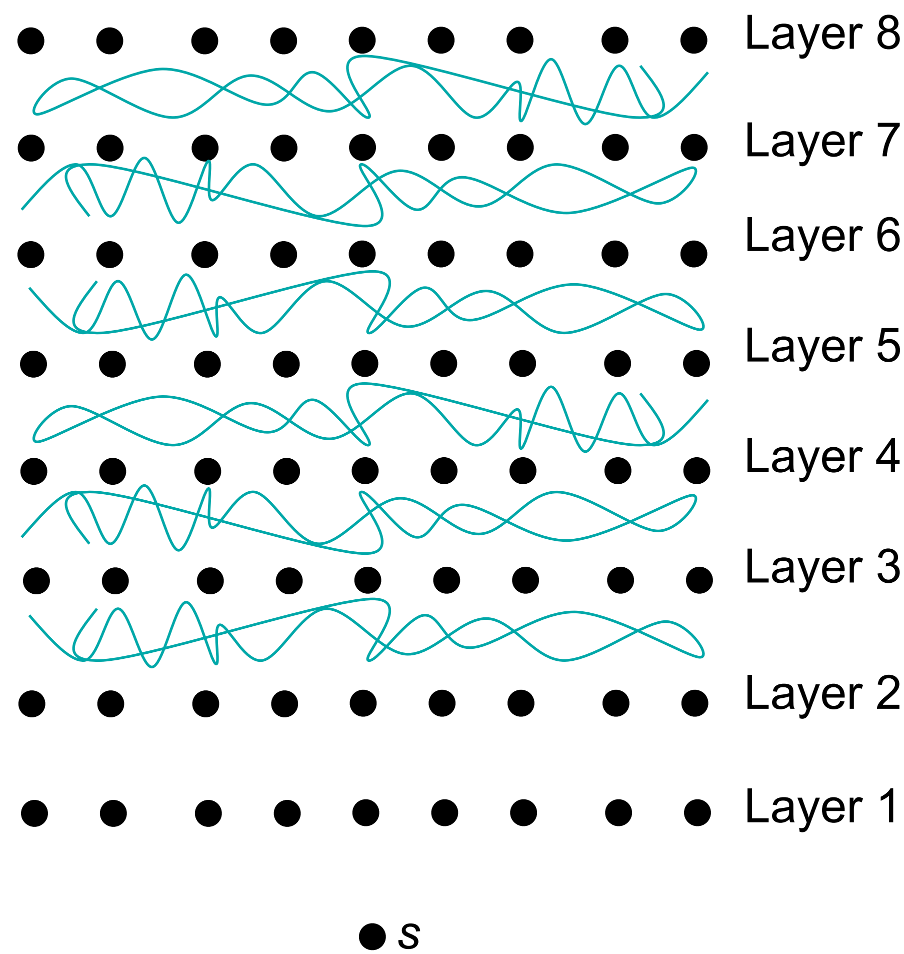
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



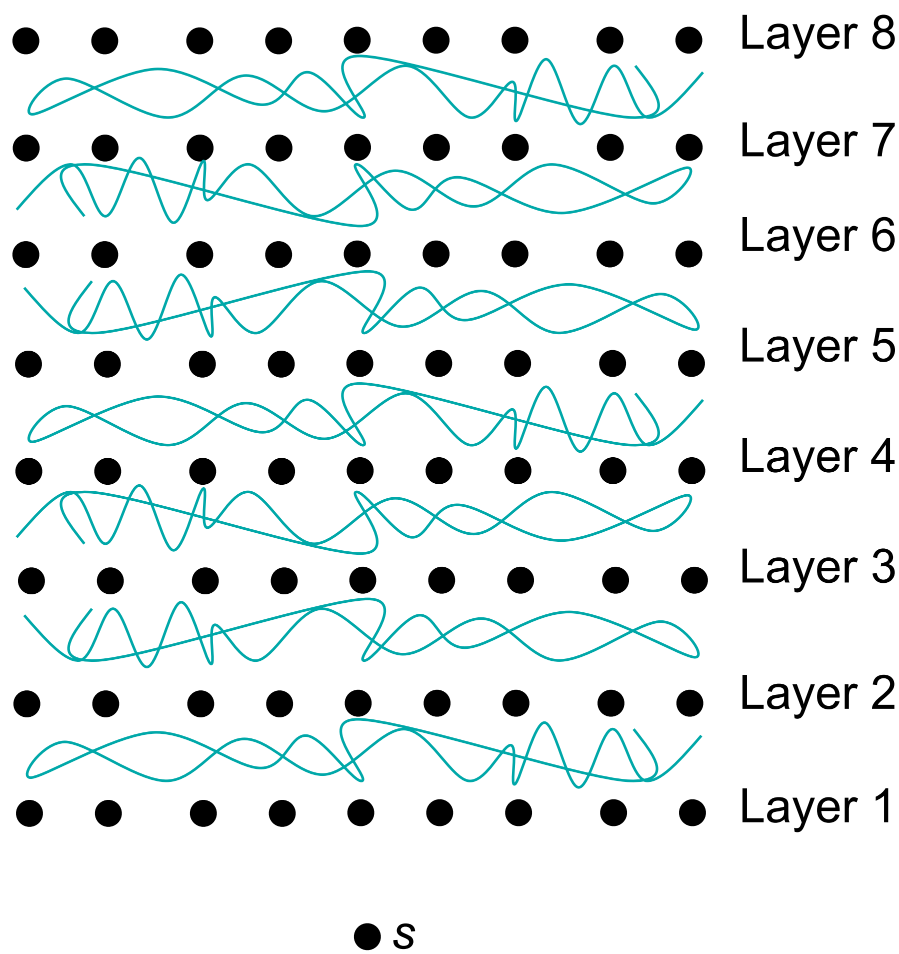
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



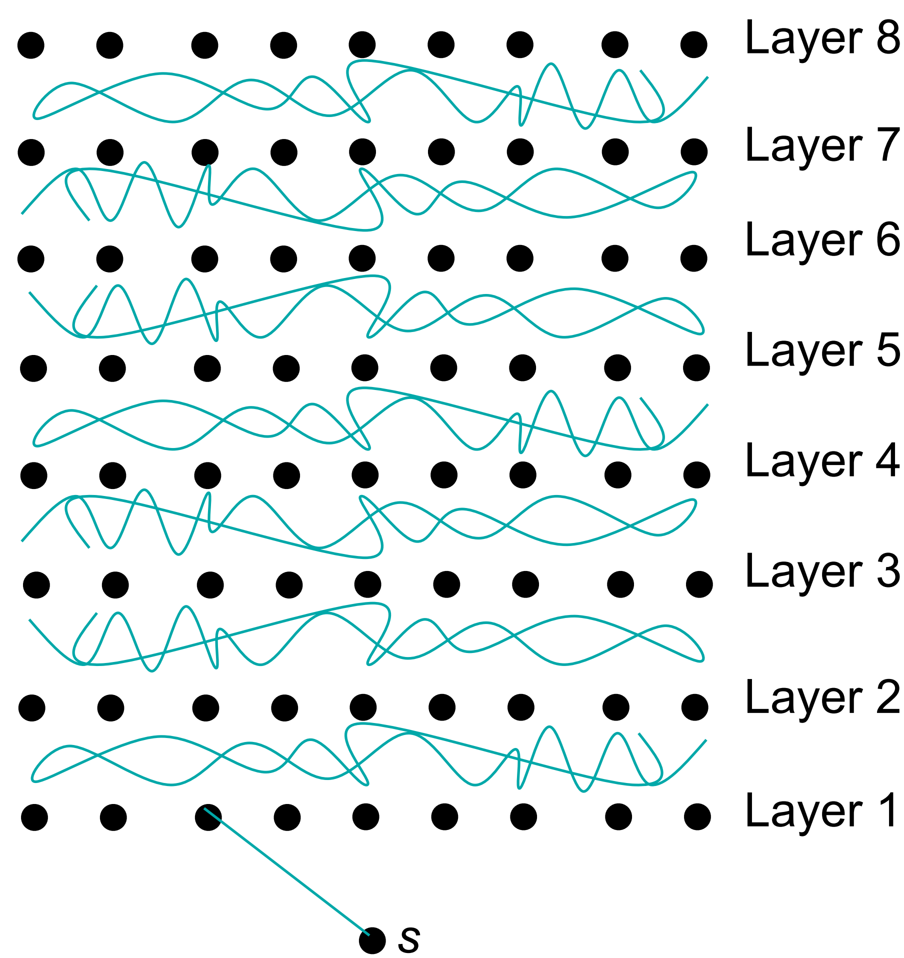
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



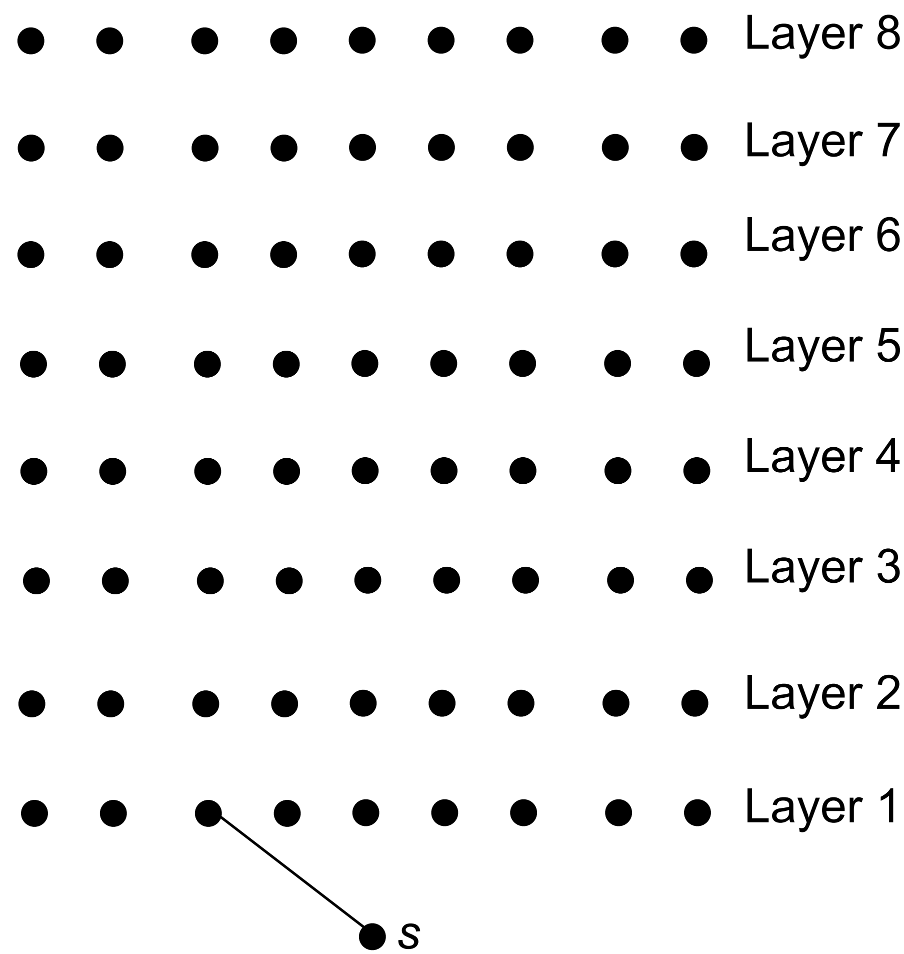
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



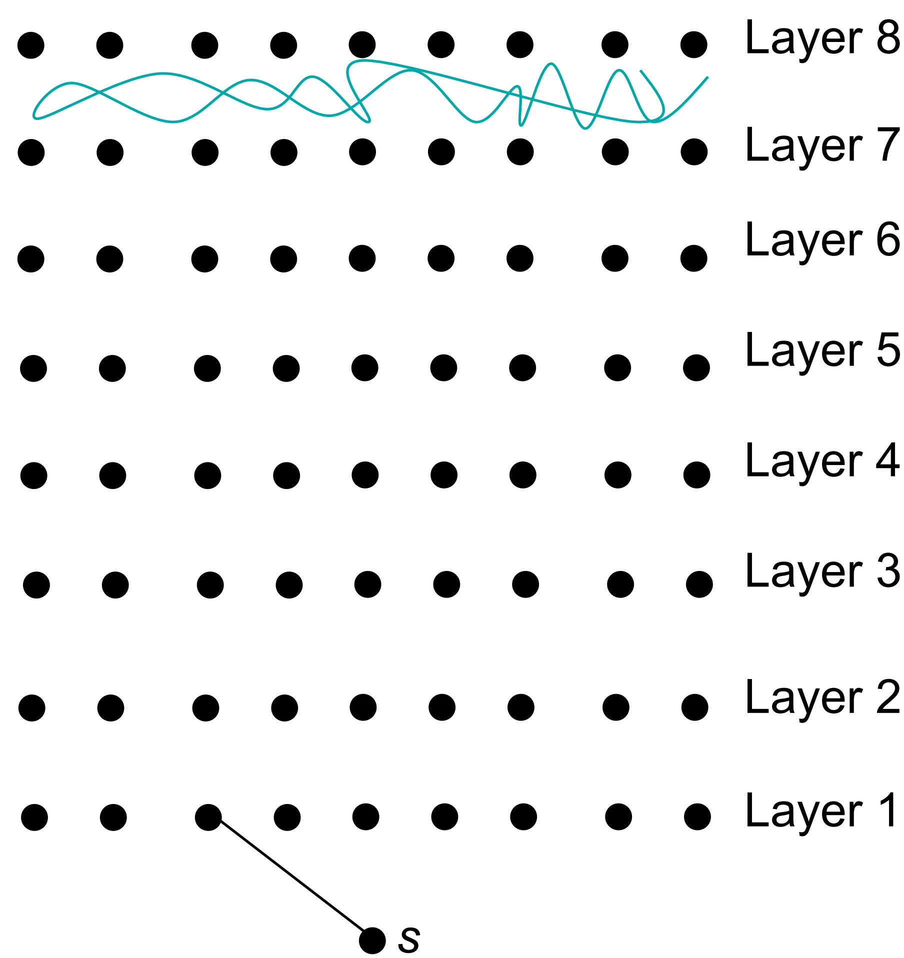
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



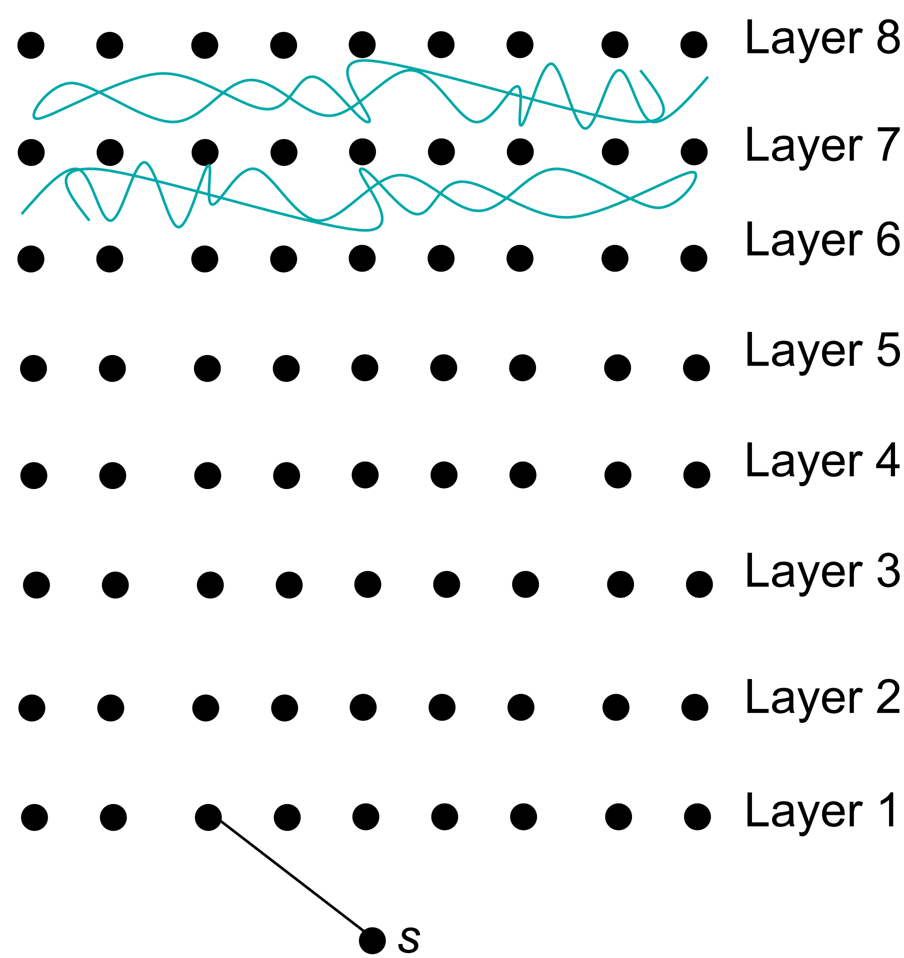
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



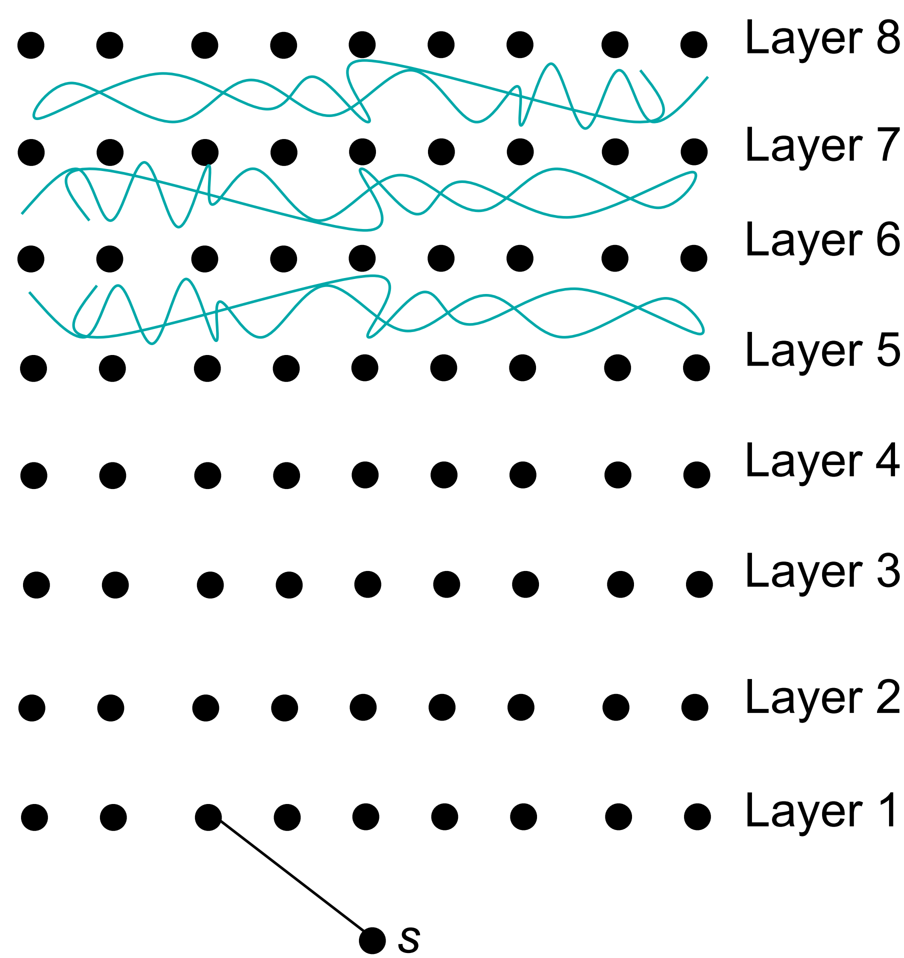
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



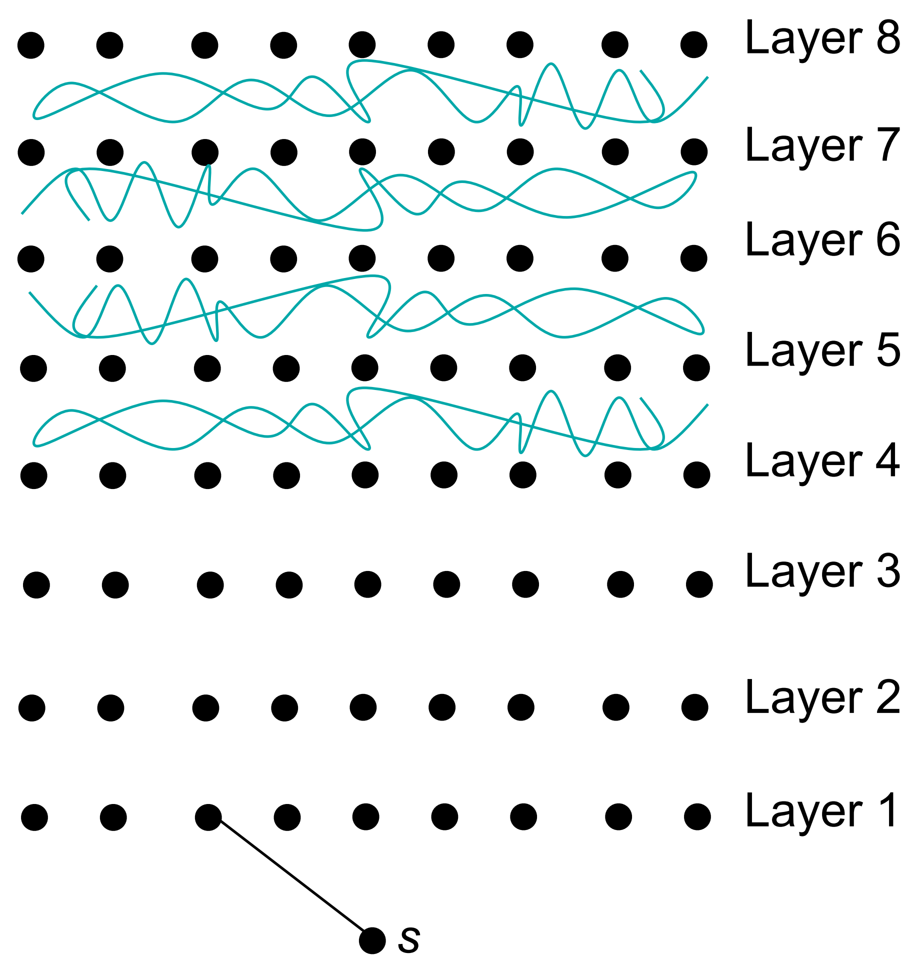
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



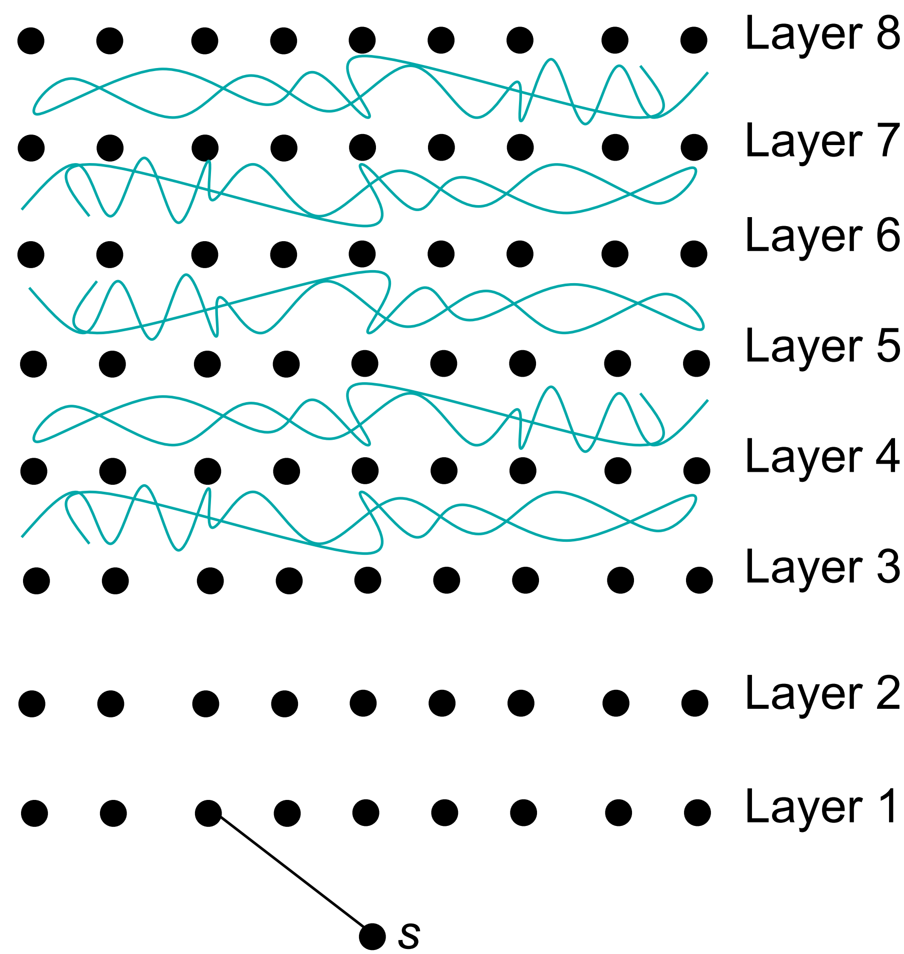
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



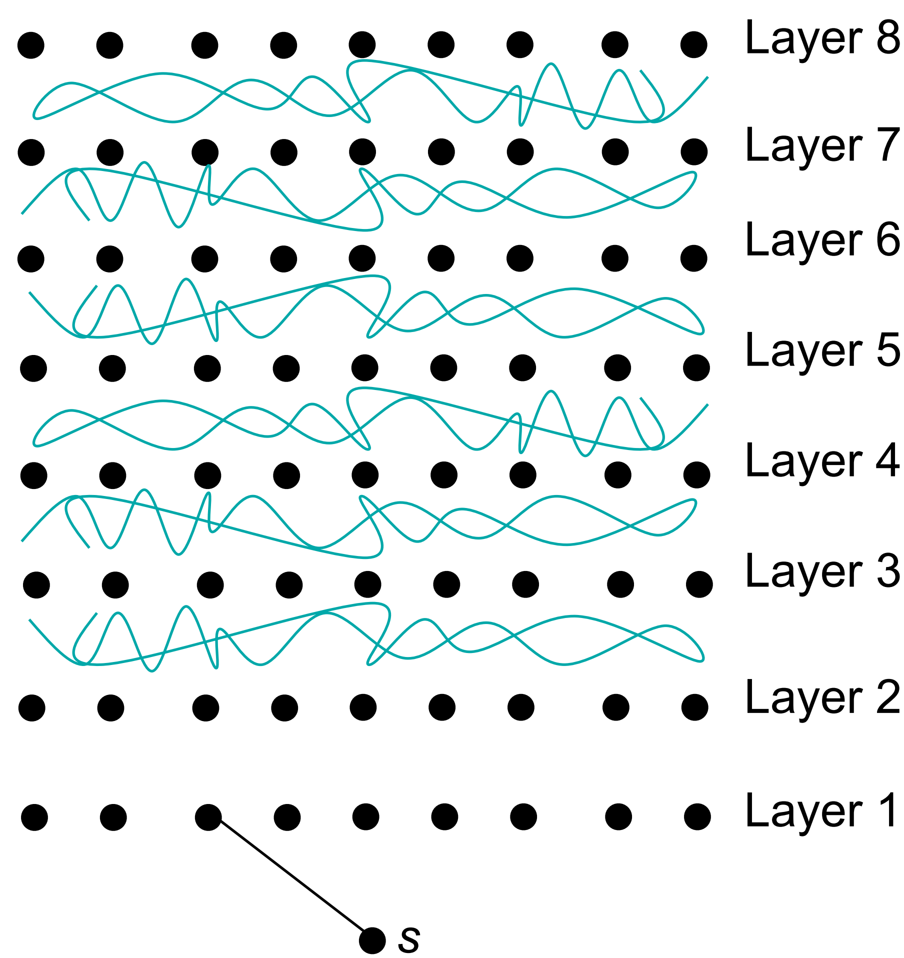
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



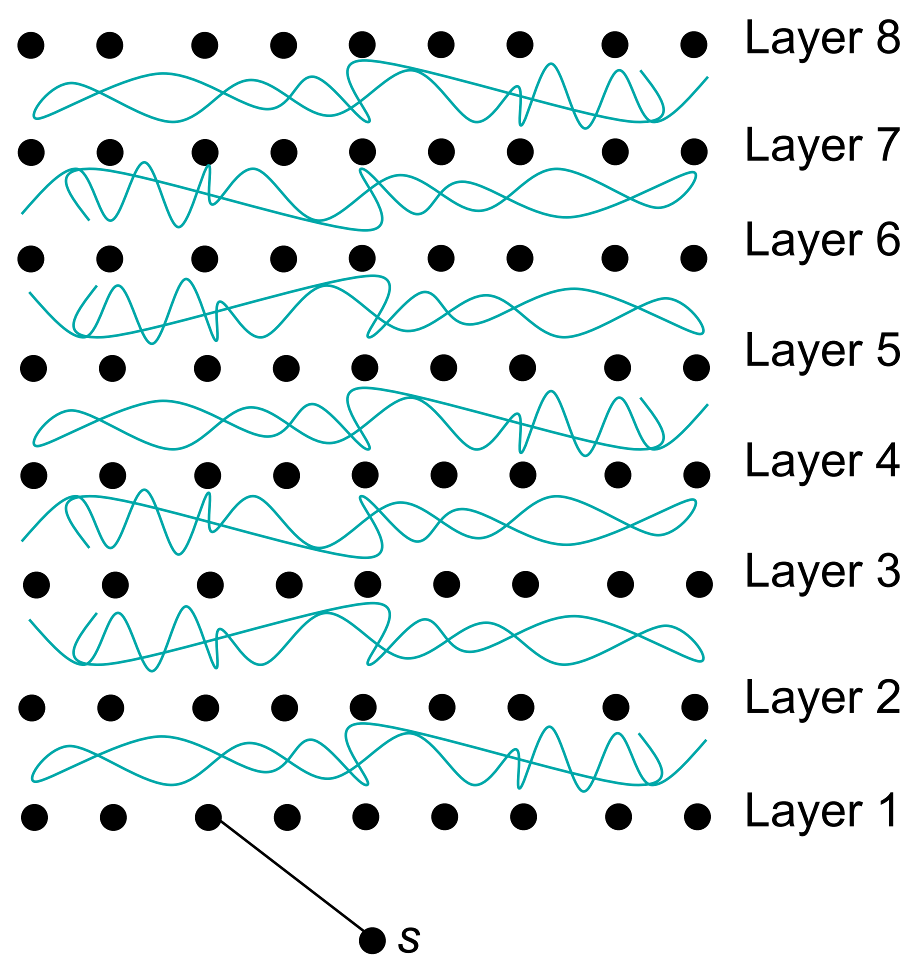
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



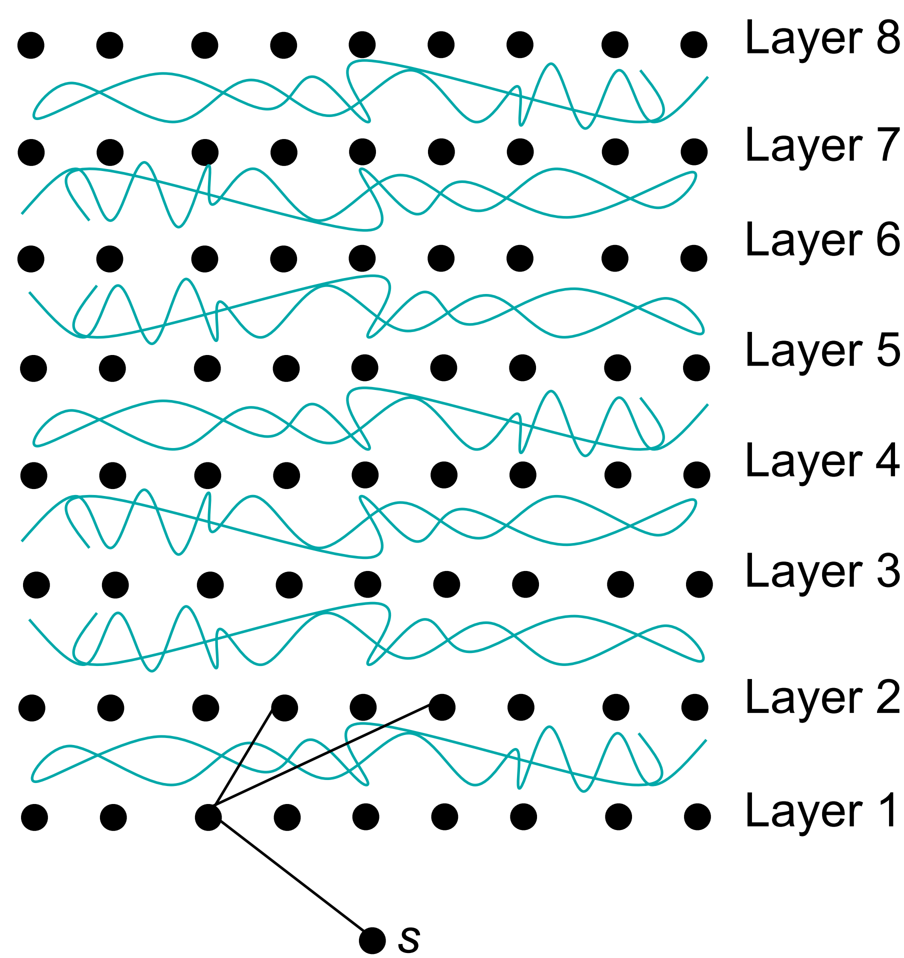
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



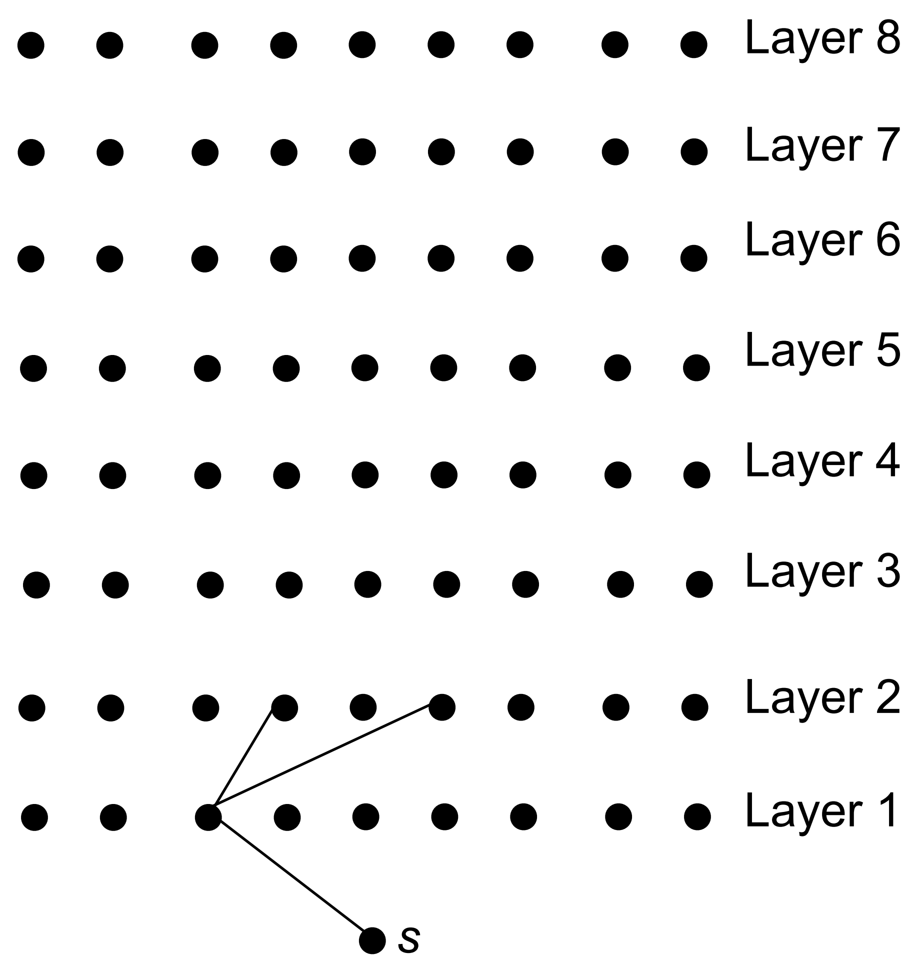
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



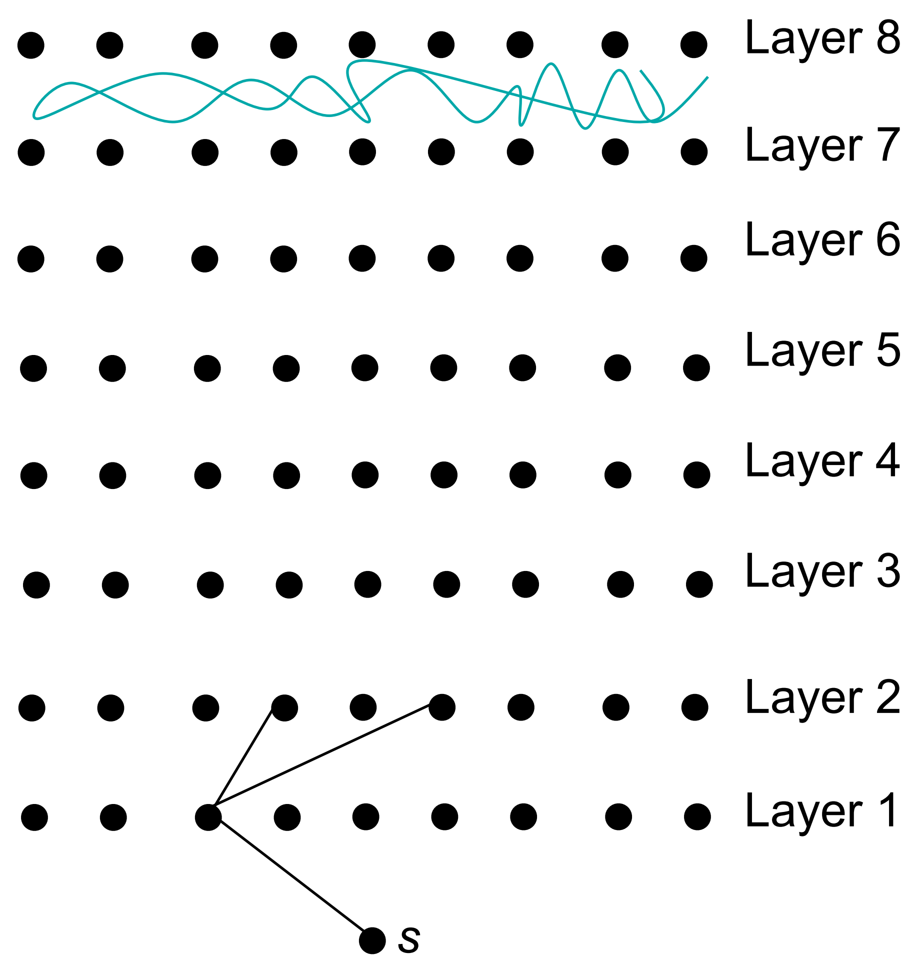
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



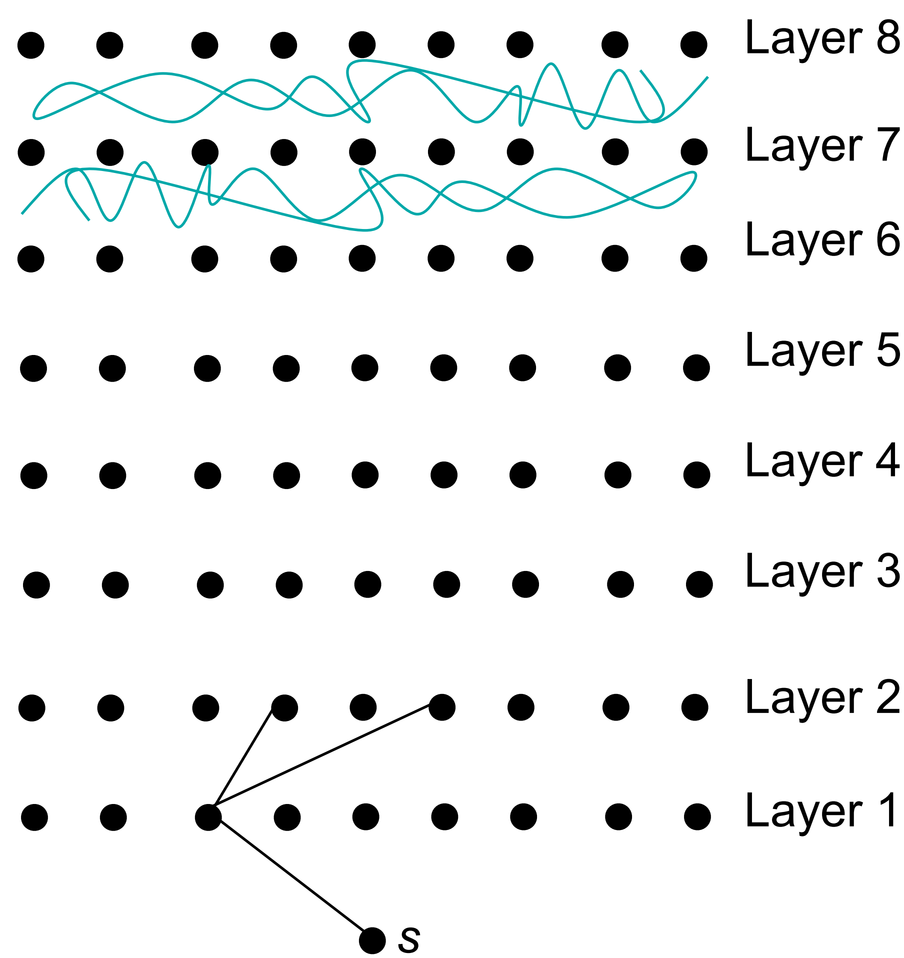
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



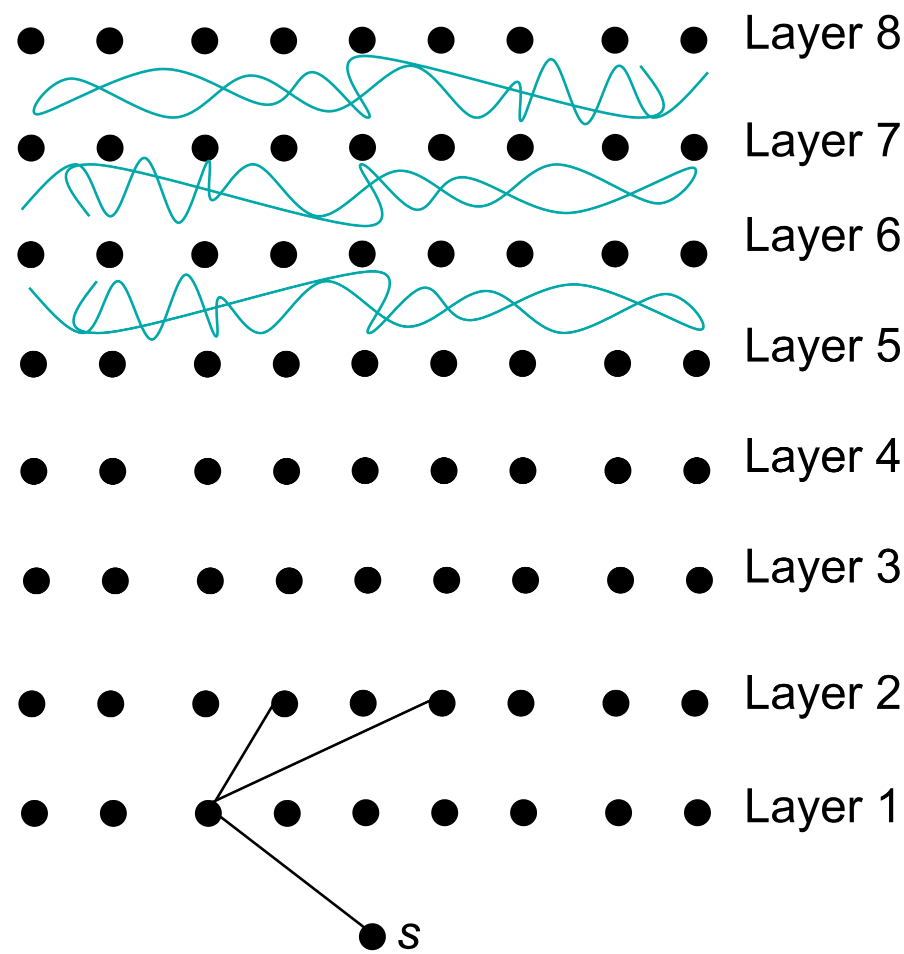
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



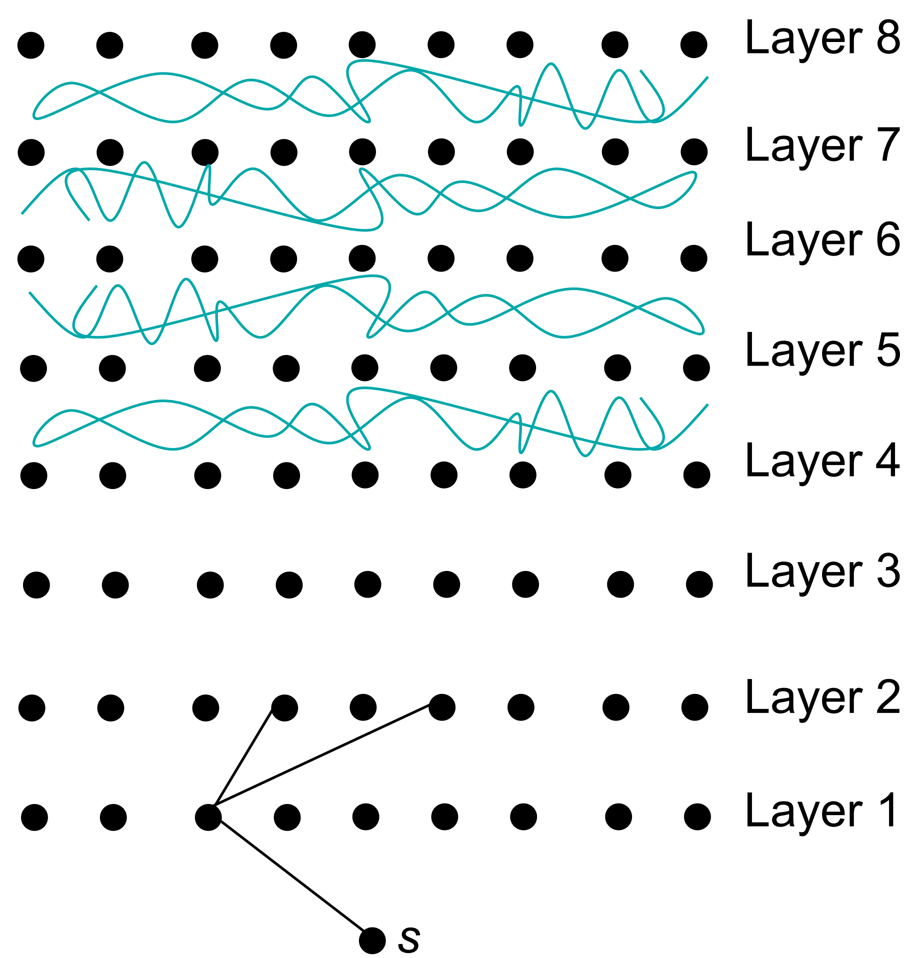
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



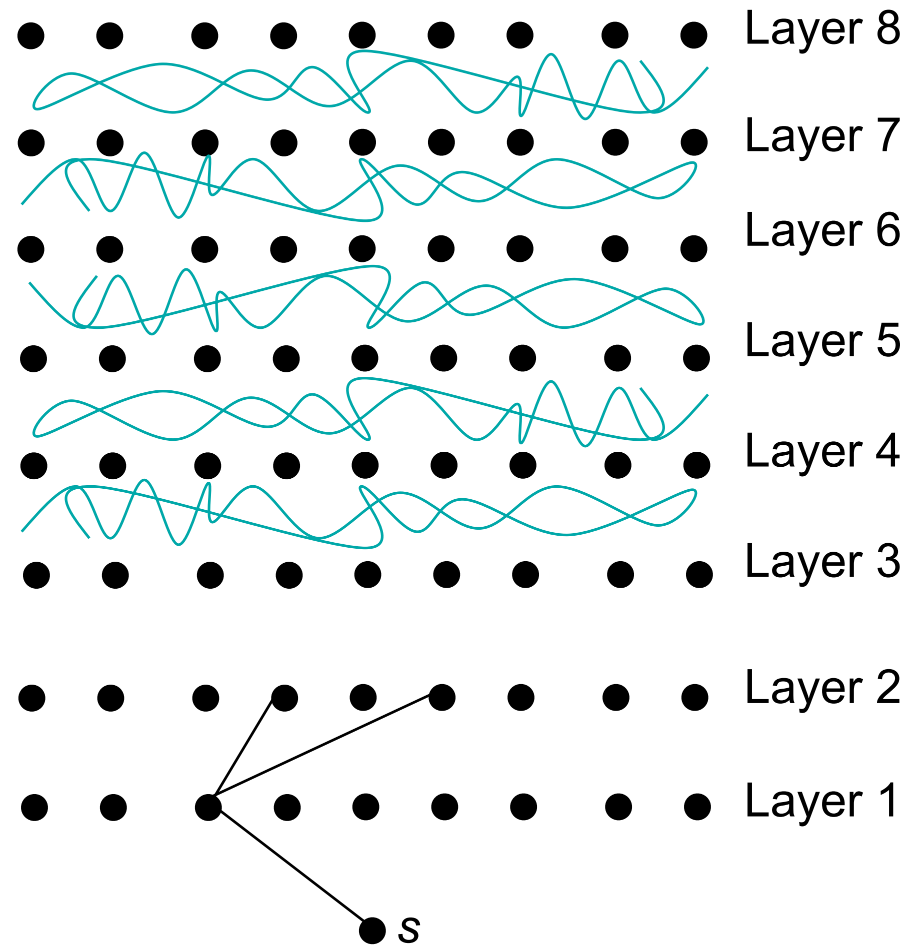
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



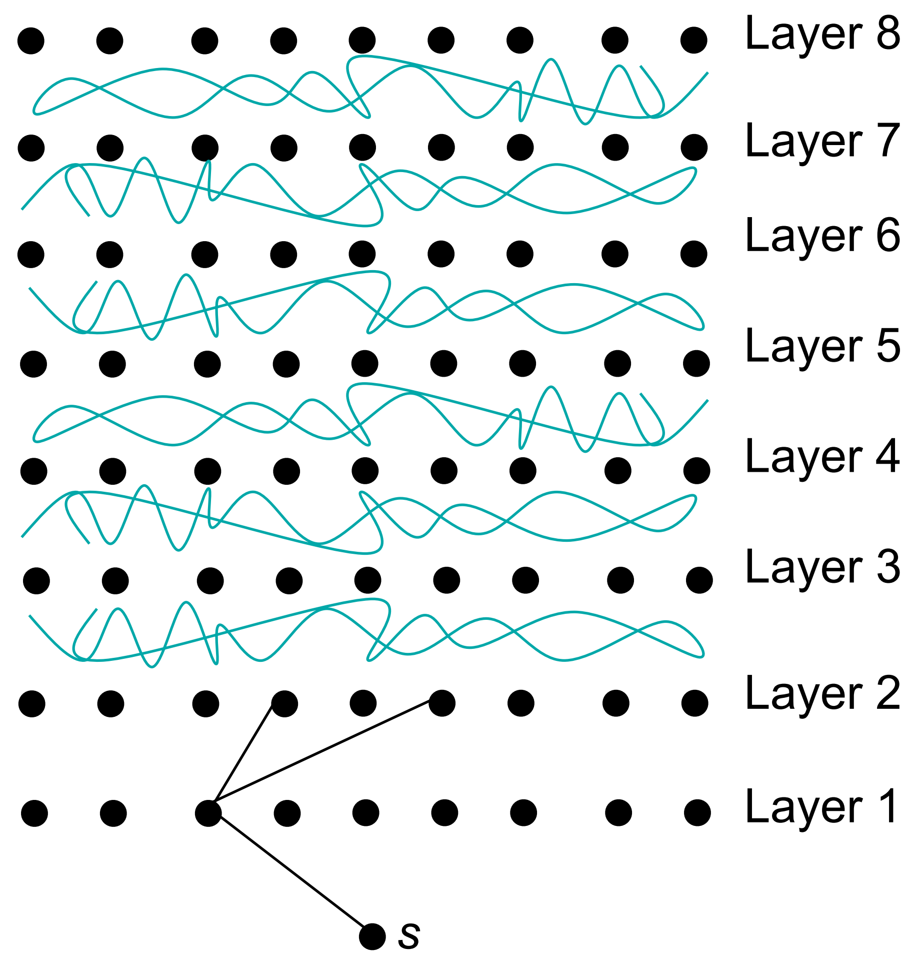
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



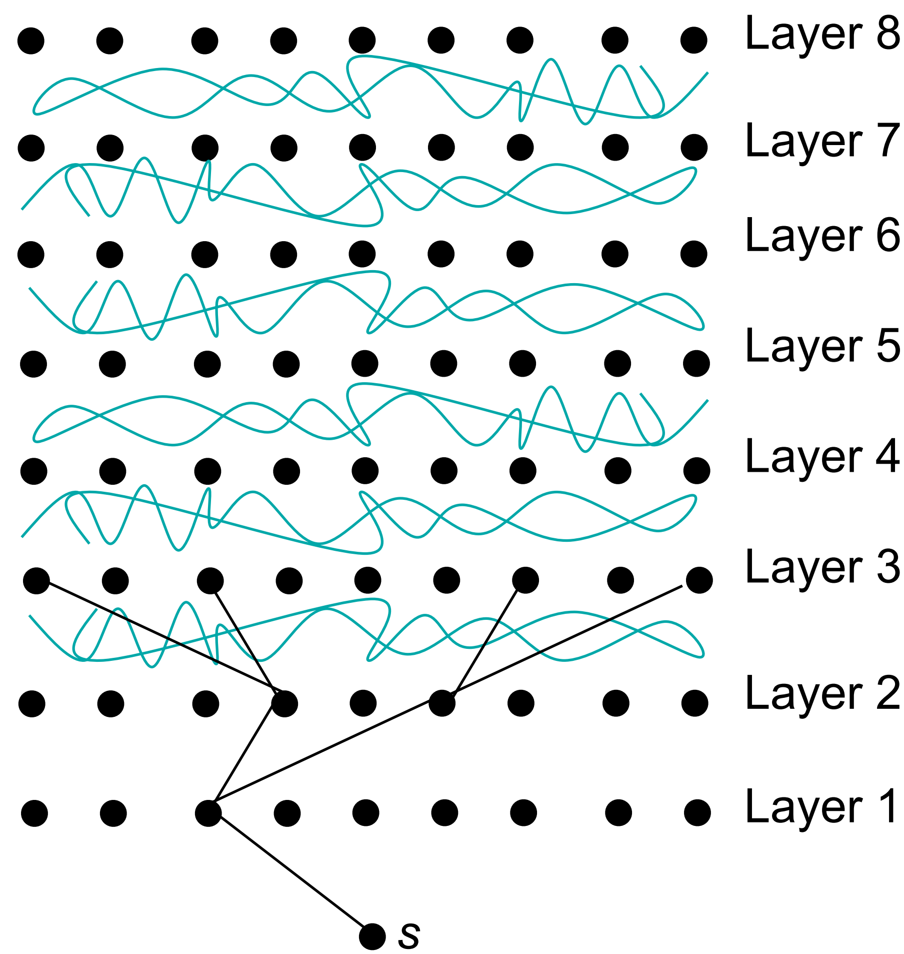
Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



Layered Graph

Problem:
Find all vertices in layer *i* that are a distance *i* from vertex *s*.



Specifics…

- Each vertex in layer *i* is connected to $({}^{n-1}/_{d})^{1/2d}$ vertices in layer *i+1* where *d* is the number of layers.
- Thm: For all $d \in \{1, \ldots, t/2\}$, doing a BFS takes *d* passes when the space restriction is $o(n^{1+1/t})$

Further Work?

- Estimating Distances in Multiple Passes?
- How important is the order of the edges?

Questions?

Questions?