CS140: Algorithms

Z Sweedyk
Lecture 15

Graph Algorithms

- Topological Sort
- Single Source Shortest Path
- All Pairs Shortest Path

Topology Sort

- Input: Directed Acyclic Graph (DAG)
- Output: Vertices of G sorted to satisfy the following condition: if there is a path from u to v in G then u occurs after v in the list.

Topological Sort – Take 1

TopoSort(G)

If G is empty return
Choose vertex v with in-degree 0.
Return v, TopoSort(G-v)

Topological Sort – Take 2

TopoSort(G)

Perform DFS with timestamp
Output vertices by ____________________

Single Source Shortest Path

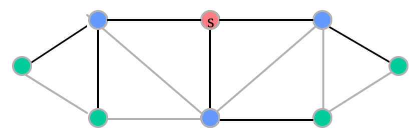
- Input: Graph G with a designated start vertex s.
- Output: For each vertex v, the distance between s and v.

Defs: Path Length, Distance

- In an unweighted graph
  - the length of a path is the number of edges in the path
  - the distance between two vertices is the length of a shortest path between the vertices
- We use $d_G(u,v)$ to denote the distance between u and v in G

Example: Distance

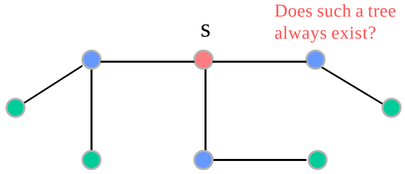
$d_G(s,s)=0$, $d_G(s,u)=1$, $d_G(s,v)=2$



Shortest path tree for s

A spanning tree of G such that the path between s and a vertex v in T is a shortest path in G.

Does such a tree always exist?



Proof of Existence: Shortest path tree for s

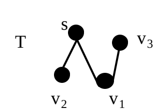
- Order the vertices of G by distance from s:
  $v_0=s, v_1, v_2, \ldots, v_n$
- Claim: There is a subtree T of G on vertices $\{v_0, \ldots, v_i\}$ such that for every v_j, $d_T(s,v_j)=d_G(s,v_j)$.

Base Case

- When i=0 the claim holds.

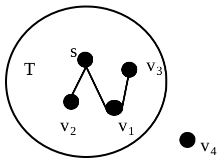
Inductive Hypothesis

- There is a tree T such that $d_T(s,v)= d_G(s,v)$ for each v in $\{s=v_0, \ldots, v_{i-1}\}$



Now add v_i

- We need to exhibit T′ such that $d_T(s,v_k)= d_G(s,v_k)$ for each v in $\{s=v_0, \dots , v_{i-1},v_i\}$.

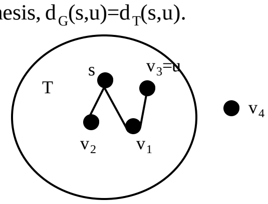


Observe

- Let s, …, u, v_i be a shortest path between s and v_i in G.
- Since i>0 u ≠ v_i.
- Thus $d_G(s,v_i)=1+d_G(s,u)>d_G(s,u)$.
- Therefore u precedes v_i in the ordering of vertices; i.e. $u=v_j$ for some j<i.

Proof of Claim

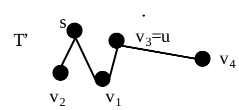
- So u is already in T and, by our induction hypothesis, $d_G(s,u)=d_T(s,u)$.



$T'=T+(u,v_i)$

- $d_{T'}(s,v_i)=1+d_T(s,u)=d_G(s,v_i)$

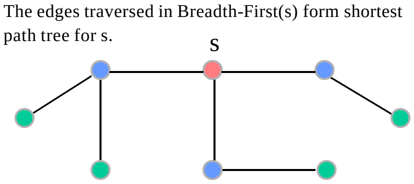
QED



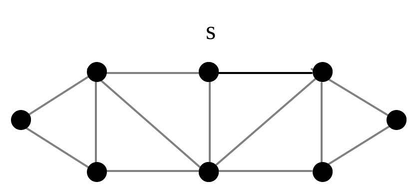
Shortest path tree for s

The path between s and v in T is a shortest path in G.

The edges traversed in Breadth-First(s) form shortest path tree for s.

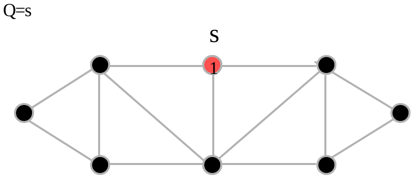


Breadth-first(s) tree:
All vertices and the edges traversed



Breadth-first(s)

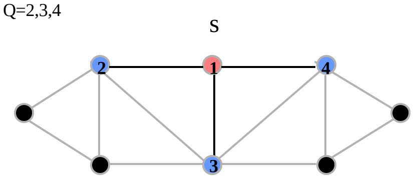
Q=s



11/07/01 CS140 Lec 15 19

Breadth-first(s)

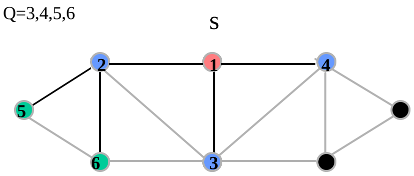
Q=2,3,4



11/07/01 CS140 Lec 15 20

Breadth-first(s)

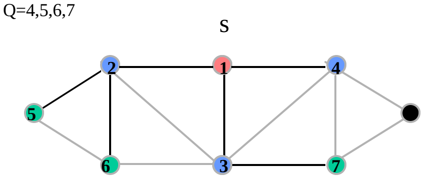
Q=3,4,5,6



11/07/01 CS140 Lec 15 21

Breadth-first(s)

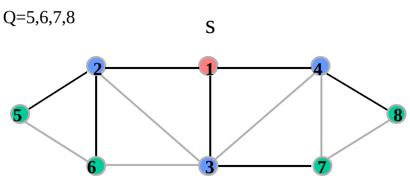
Q=4,5,6,7



11/07/01 CS140 Lec 15 22

Breadth-first(s)

Q=5,6,7,8



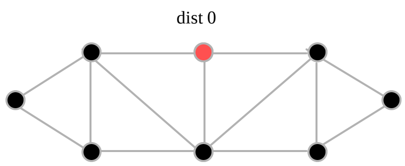
11/07/01 CS140 Lec 15 23

Single Source Shortest Path

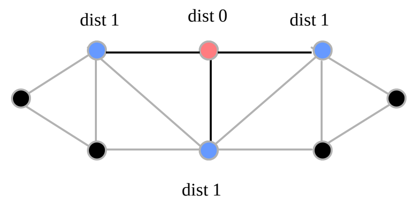
- Input: Graph G with a designated start vertex s.
- Output: For each vertex v, the length of the shortest path between s and v.
- Algorithm: Modify Breadth-first to compute d(s,v) along the way.

11/07/01 CS140 Lec 15 24

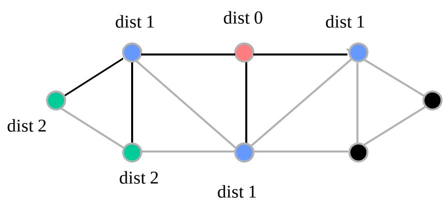
Breadth-first search
Compute distance from s



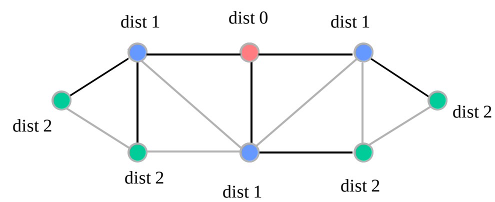
Breadth-first search
Compute distance from s



Breadth-first search
Compute distance from s



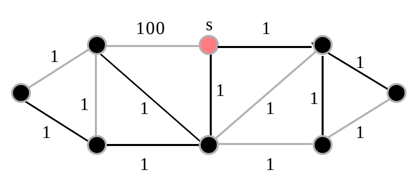
Breadth-first search
Compute distance from s



Running Time

- The modified breadth-first algorithm for single source shortest path in an unweighted graph is: ____________________

What if G is weighted?



Single Source Shortest Path

- Input: Weighted graph G with a designated start vertex s. Weights are positive!
- Output: For each vertex v, the length of the **shortest path** between s and v.

11/07/01 CS140 Lec 15 31

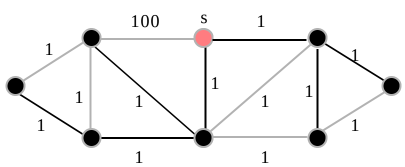
Path Length

- In a weighted graph the length of a path is the sum of the weights of the edges of the path.

11/07/01 CS140 Lec 15 32

Shortest path tree for s

The path between s and v in T is a **shortest path** in G.



11/07/01 CS140 Lec 15 33

Shortest path tree for s

- Is it clear such a tree exists? YES by same argument.

11/07/01 CS140 Lec 15 34

Shortest path tree for s

- Claim: Let the vertices of G be sorted by distance from s. Then there is a subtree T of G on vertices $\{v_0, \dots, v_i\}$ such that $0 \le k \le i$, $d_T(s,v) = d_G(s,v)$ for each v in T.
- If you have a tree for $\{v_0, \dots, v_i\}$ can you find the tree for $\{v_0, \dots, v_i, v_{i+1}\}$?

11/07/01 CS140 Lec 15 35

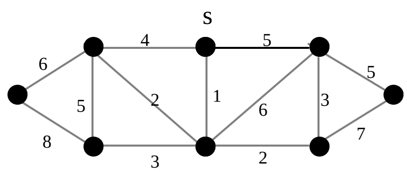
Shortest path tree for s

- Suppose you don't know the ordering?
- At each step find the vertex $v \notin T$ that minimizes

  $\min_{u \in T} d_T(s,u) + w(u,v)$.

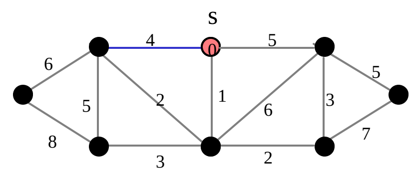
11/07/01 CS140 Lec 15 36

Dykstra's Algorithm



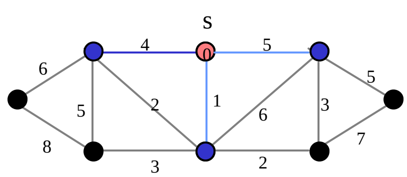
Dykstra's Algorithm

Number in node u indicate $d_G(s,u)$



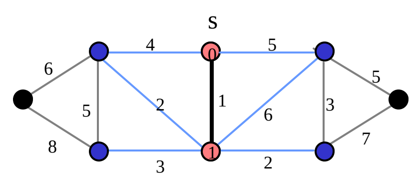
Dykstra's Algorithm

Number in node u indicate $d_G(s,u)$



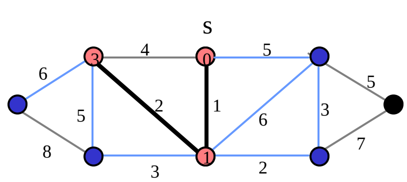
Dykstra's Algorithm

Number in node u indicate $d_G(s,u)$



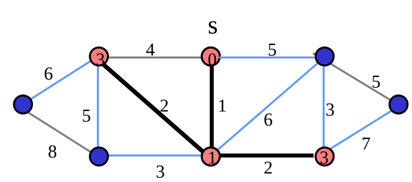
Dykstra's Algorithm

Number in node u indicate $d_G(s,u)$



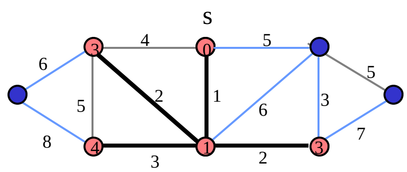
Dykstra's Algorithm

Number in node u indicate $d_G(s,u)$



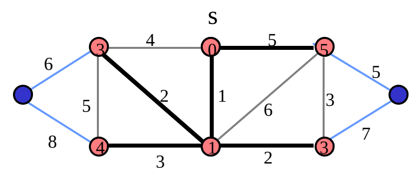
Dykstra’s Algorithm

Number in node u indicate $d_G(s,u)$



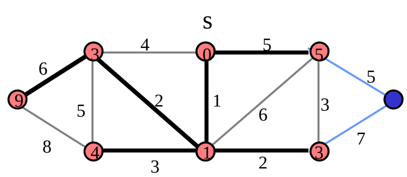
Dykstra’s Algorithm

Number in node u indicate $d_G(s,u)$



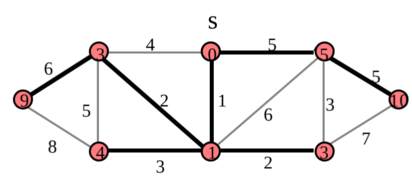
Dykstra’s Algorithm

Number in node u indicate $d_G(s,u)$



Dykstra’s Algorithm

Number in node u indicate $d_G(s,u)$



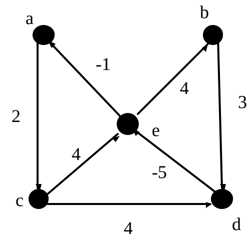
Does this sound familiar?

- Prim’s algorithm for MST is VERY similar.
- The implementation details are almost identical.

All Pairs Shortest Path (directed version)

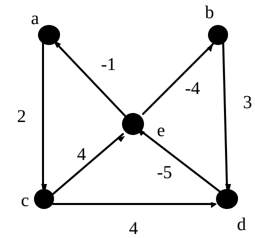
- Input: Weighted digraph G
- Output: For each pair of vertices x,y the distance between x and y in G

What is d(b,a)?



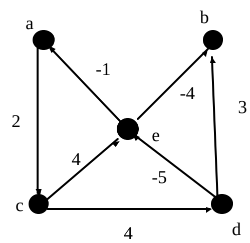
11/07/01 CS140 Lec 15 49

What is d(b,a)?



11/07/01 CS140 Lec 15 50

What is d(b,a)?



11/07/01 CS140 Lec 15 51

Three Cases:

- There is no path from a to b
- There is a path from a to b but no shortest path
- There is a shortest path from a to b

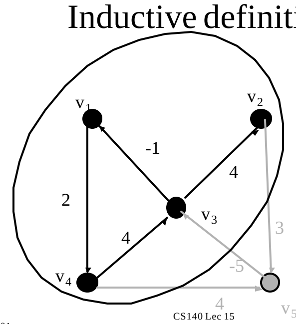
11/07/01 CS140 Lec 15 52

Shortest path algorithm should

- Determine which case holds
  - There is no path from a to b
  - There is a path from a to b but no shortest path
  - There is a shortest path from a to b
- Find the length of the shortest path when one exists

11/07/01 CS140 Lec 15 53

All Pairs Shortest Path Inductive definition



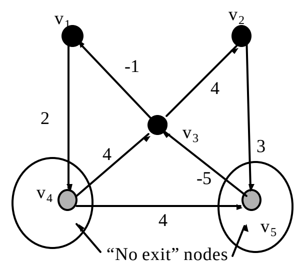
Solve and then what?

11/07/01 CS140 Lec 15 54

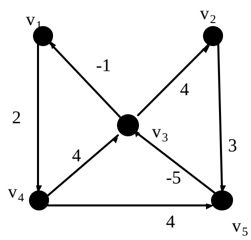
K-limited paths

- A path from v_i to v_j is k-limited if the intermediate vertices in the path are numbered k or less

3-limited paths



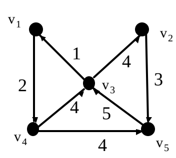
What is the shortest 5-limited path from v_1 to v_2?



Floyd-Warshall algorithm

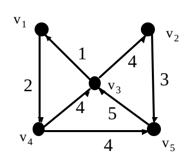
- $D^k(i,j)$ is the length of a shortest k-limited path from v_i to v_j
- $D^k(i,j) = \min(D^{k-1}(i,j),\ D^{k-1}(i,k)+ D^{k-1}(k,j))$
- $D^0(i,j) = w(<v_i,v_j>)$

D^0



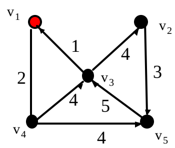
	1	2	3	4	5
1					
2					
3					
4					
5					

D^0



	1	2	3	4	5
1	0	∞	∞	2	∞
2	∞	0	∞	∞	3
3	1	2	0	∞	∞
4	∞	∞	4	0	4
5	∞	∞	5	∞	0

D^1



	1	2	3	4	5
1					
2					
3					
4					
5					

$D^1(i,j)=\min(D^0(i,j),\ D^0(i,1)+D^0(1,j))$

D^0

	1	2	3	4	5
1	0	∞	∞	2	∞
2	∞	0	∞	∞	3
3	1	2	0	∞	∞
4	∞	∞	4	0	4
5	∞	∞	5	∞	0

D^1

	1	2	3	4	5
1					
2					
3					
4					
5					

$D^1(i,j)=\min(D^0(i,j),\ D^0(i,1)+D^0(1,j))$

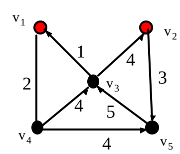
D^0

	1	2	3	4	5
1	0	∞	∞	2	∞
2	∞	0	∞	∞	3
3	1	4	0	3	∞
4	∞	∞	4	0	4
5	∞	∞	5	0	0

D^1

	1	2	3	4	5
1	0	∞	∞	2	∞
2	∞	0	∞	∞	3
3	1	4	0	3	∞
4	∞	∞	4	0	4
5	∞	∞	5	0	0

D^2



D^1

	1	2	3	4	5
1	0	∞	∞	2	∞
2	∞	0	∞	∞	3
3	1	4	0	3	∞
4	∞	∞	4	0	4
5	∞	∞	5	0	0

$D^2(i,j)=\min(D^1(i,j),\ D^1(i,2)+D^1(2,j))$

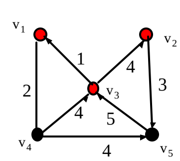
D^1

	1	2	3	4	5
1	0	∞	∞	2	∞
2	∞	0	∞	∞	3
3	1	4	0	∞	∞
4	∞	∞	4	0	4
5	∞	∞	5	∞	0

D^2

	1	2	3	4	5
1	0	∞	∞	2	∞
2	∞	0	∞	∞	3
3	1	4	0	3	**7**
4	∞	∞	4	0	4
5	∞	∞	5	∞	0

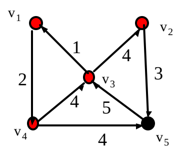
And so on …



D^3

	1	2	3	4	5
1	0	∞	∞	2	∞
2	∞	0	∞	∞	3
3	1	4	0	3	7
4	**5**	**8**	4	0	4
5	**6**	**9**	5	**8**	0

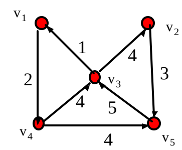
And so on …



D^4

	1	2	3	4	5
1	0	**10**	**6**	2	**6**
2	∞	0	∞	∞	3
3	1	4	0	3	7
4	5	8	4	0	4
5	6	9	5	8	0

And so on …



D^5

	1	2	3	4	5
1	0	10	6	2	6
2	9	0	8	11	3
3	1	4	0	3	7
4	5	8	4	0	4
5	6	9	5	8	0

Floyd-Warshall algorithm

- $D^0(i,j) = w(<v_i,v_j>)$
- For i=1 to n
  Compute D^i from D^{i-1}
- Return D^n

Floyd-Warshall algorithm Running Time

- n Tables
- Each is n X n
- Each table entry takes O(1)

- $O(n^3)$