Outline

- MST: Run Time analysis
  - Prim's
  - Kruskal's
- Single Source Shortest Path
  - Breadth-First Search
  - Dykstra's algorithm

Prim's Algorithm

Choose a vertex w ε V

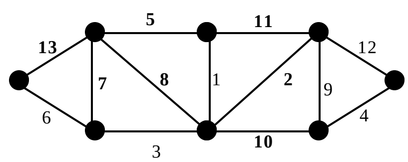
F={w}

While V-F ≠ ϕ

  Let e be a minimum weight edge that emerges from F

  F=F+{e}

Prim's example



Prim's Algorithm Running Time: O(n(?))

Choose a vertex w ε V

F={w}

While V-F ≠ ϕ

  Let e be a minimum weight edge that emerges from F

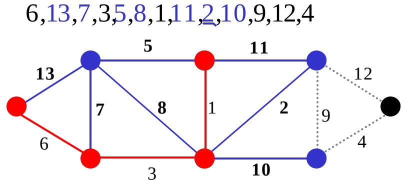
  F=F+{e}

How should we implement this?

Consider naïve approach …

- Go through edge list to find least weight edge emerging from F:

  6,13,7,3,5,8,1,11,2,10,9,12,4



Prim's Algorithm Running Time: O(nm)

Choose a vertex w ε V

F={w}

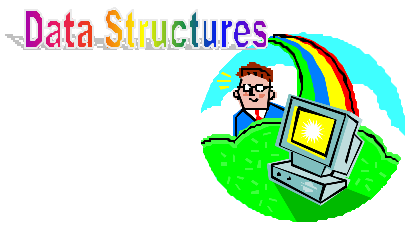
While V-F ≠ ϕ

  Let e be a minimum weight edge that emerges from F

  F=F+{e}

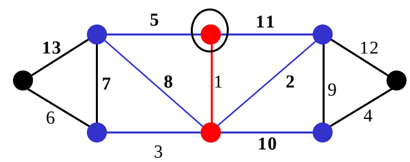
Naïve approach O(m)

What to do?



Fringe vertex

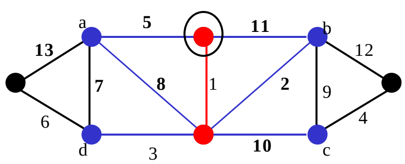
v is a fringe vertex if it is in V-F and it is connected by an edge to a vertex u in F



A less naïve approach …

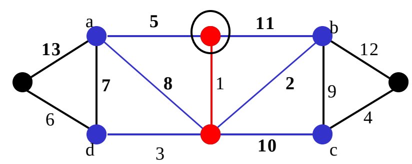
List of fringe vertices and for each its minimum weight edge to F

[b,2], [d,3], [a,5],[c,10]



And even better…

- Keep the fringe vertices in a HEAP!!!



Prim's Algorithm
A Better Implementation

Choose a vertex x ε V

F={x}, H=ϕ

For each e=(u,x): Add record [u,e] to heap H keyed on w(e)

While H ≠ ϕ

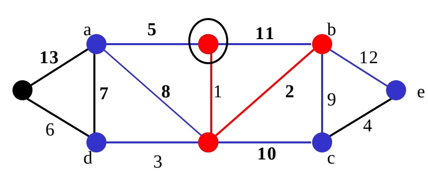
[u,e]=Find-min(H)

Add u and e to F

For each edge incident to u: Update heap

Update heap: [b,2],[d,3],[a,5],[c,10]

- [b,2]: remove [b,2] from heap [d,3],[a,5],[c,10]
- Add new fringe vertices: [d,3],[a,5],[c,10],[e,12]
- Update edge weights: [d,3],[a,5],[c,9],[e,12]



Are these standard operations?

- Extract-min
- Add element to heap
- Reduce key of element in heap ?!?

11/06/00 CS140(16) 13

Decrease key

- Next homework assignment: Design decrease key algorithm for heaps that runs in time O(lg(n)).

11/06/00 CS140(16) 14

Prim's Algorithm Running Time

Choose a vertex x ε V
F={x}, H=ϕ
For each e=(u,x): **Insert**
While H ≠ ϕ
  [u,e]=**Extract-min**(H)
  Add u and e to F
  For each edge incident to u: **Insert or Decrease-key** or do nothing

11/06/00 CS140(16) 15

Prim's Algorithm Running Time

- Heap operations across algorithm:
  - n Extract-mins O(lg(n)) each
  - n Inserts O(lg(n)) each
  - m-n Decrease-keys O(lg(n)) each
  - m Do nothings O(1) each
- Total time is O(m lg(n))

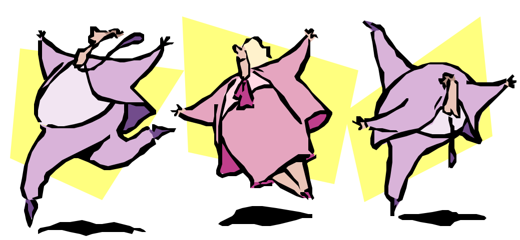
11/06/00 CS140(16) 16

But wait …suppose we could decrease-key in time O(1)

- Heap operations across algorithm:
  - n Extract-mins O(lg(n)) each
  - n Inserts O(lg(n)) each
  - m-n Decrease-keys ~~O(lg(n))~~ O(1) each
  - m Do nothings O(1) each
- Then total time is O(m + n lg(n))

11/06/00 CS140(16) 17

Bravo, bravo …



11/06/00 CS140(16) 18

Do It With Fibonacci Heaps

Huh?

Don't worry– it works!

Kruskal's Algorithm

Let $e_1, e_2, \ldots, e_m$ be the edges of G sorted by increasing weight. O(m log m)

F=V O(n)

For i=1 to m

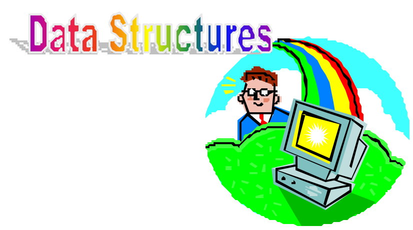
If F+{e_i} is acyclic then F=F+{e_i}. O(n)

Return(F)

Kruskal's Algorithm

- O(m log m + nm)

But Wait …



Union-Find Data Structure

- Disjoint-sets: $S_1, S_2, \ldots, S_k$
- Operations:
  - MakeSet(x)
  - Union(x,y)
  - FindSet(x)

Kruskal's Algorithm

Let $e_1, e_2, \ldots, e_m$ be the edges of G sorted by increasing weight. O(m log m)

F=V : For each v in V MakeSet(v)

For i=1 to m

If F+{e_i} is acyclic then F=F+{e_i}:

Assume e_i=(u,v)

If FindSet(u) $\neq$ FindSet(v) then Union(u,v)

Return(F)

Union-Find Data Structure

- Disjoint-sets: $S_1, S_2, \ldots, S_k$
- Operations:
  - MakeSet(x) O(1)
  - Union(x,y)
  - FindSet(x)

Running Time Analysis

- Prim's (and Dijkstra's)
  - Binary heap: O(m lg(n))
  - Fibonnaci heaps: O(m+n lg(n))
- Kruskal's
  - DFS: O(m lg(m) + n^2)
  - Union Find: O(m lg(m))

Single Source Shortest Path

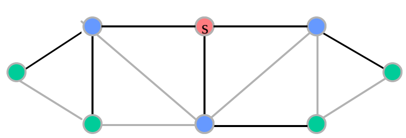
- Input: Graph G with a designated start vertex s.
- Output: For each vertex v, the distance between s and v.

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

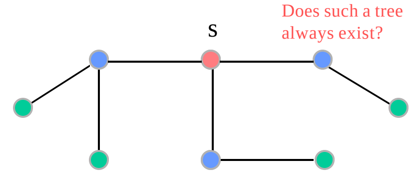
$d_G(s,s)=0$, $d_G(s,u)=1$, $d_G(s,v)=2$



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

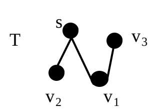
- Order the vertices of G by distance from s: $v_0 = s, v_1, v_2, \ldots, v_n$
- Claim: There is a subtree T of G on vertices $\{v_0, \ldots, v_i\}$ such that for every v_j, $d_T(s,v_j) = d_G(s,v_j)$.

Base Case

- When i=0 the claim holds.

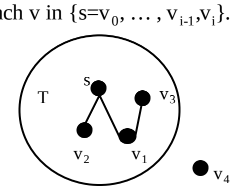
Inductive Hypothesis

- 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 $\{s=v_0, \ldots, v_{i-1}, v_i\}$.

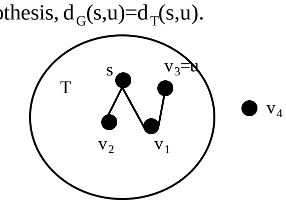


Observe

- Let $s, \ldots, u, v_i$ be a shortest path between s and v_i in G.
- Since i>0 $u \neq v_i$.
- 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

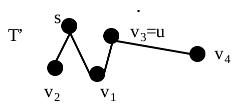
- So u is already in T and, by our induction hypothesis, $d_G(s,u) = d_T(s,u)$.



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

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

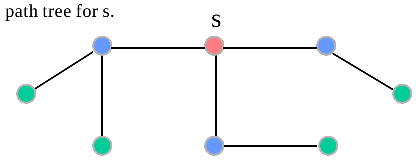
QED



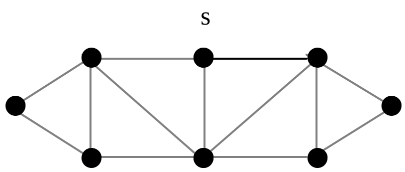
Shortest path tree for s

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.

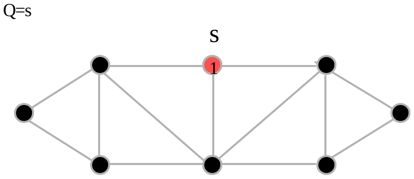


Breadth-first(s) tree:
All vertices and the edges traversed



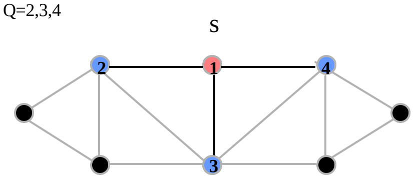
Breadth-first(s)

Q=s



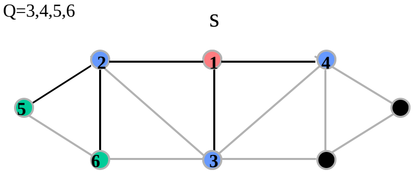
Breadth-first(s)

Q=2,3,4



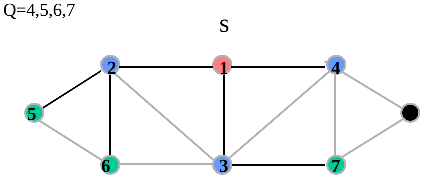
Breadth-first(s)

Q=3,4,5,6



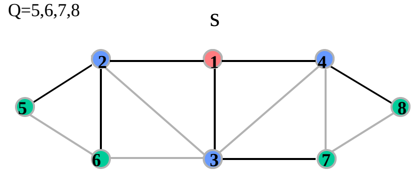
Breadth-first(s)

Q=4,5,6,7



Breadth-first(s)

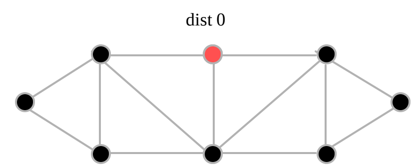
Q=5,6,7,8



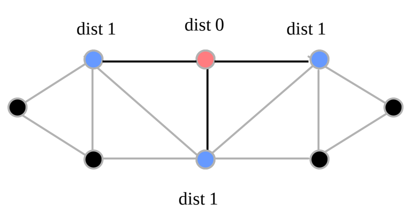
Single Source Shortest Path

- 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.

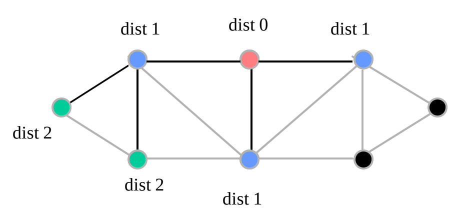
Breadth-first search
Compute distance from s



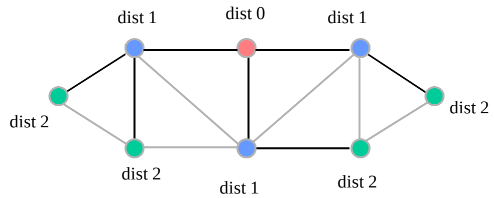
Breadth-first search
Compute distance from s



Breadth-first search
Compute distance from s



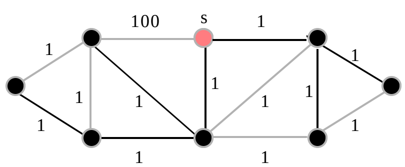
Breadth-first search
Compute distance from s



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

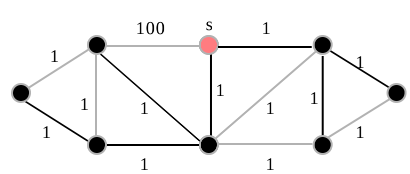
- Input: Weighted graph G with a designated start vertex s. Weights are positive!
- Output: For each vertex v, the length of the **shortest path** between s and v.

Path Length

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

Shortest path tree for s

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



Shortest path tree for s

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

Shortest path tree for s

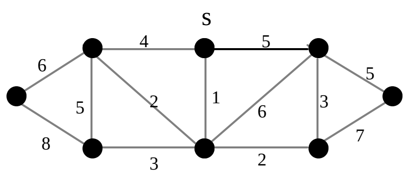
- Claim: Let the vertices of G be sorted by distance from s. Then there is a subtree T of G on vertices $\{v_0, \ldots, v_i\}$ such that $0 \le k \le i$, $d_T(s,v) = d_G(s,v)$ for each v in T.
- If you have a tree for $\{v_0, \ldots, v_i\}$ can you find the tree for $\{v_0, \ldots, v_i, v_{i+1}\}$?

Shortest path tree for s

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

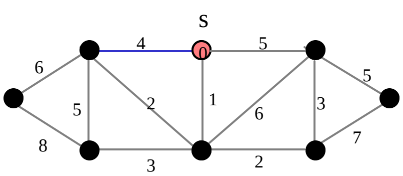
  $\min_{u \in T} d_T(s,u)+w(u,v)$.

Dykstra's Algorithm



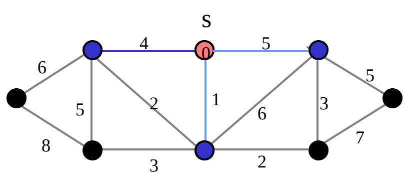
Dykstra's Algorithm

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



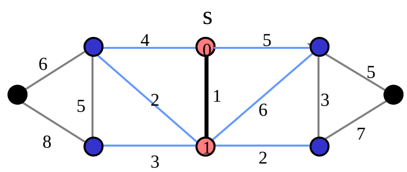
Dykstra's Algorithm

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



Dykstra's Algorithm

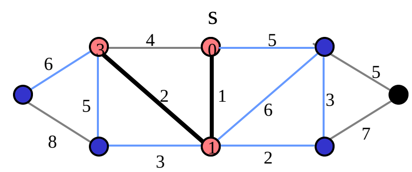
Number in node u indicate $d_G(s,u)$



11/06/00 CS140(16) 61

Dykstra's Algorithm

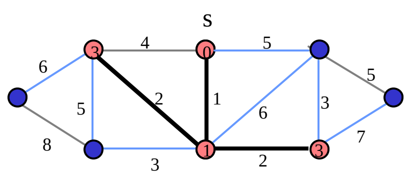
Number in node u indicate $d_G(s,u)$



11/06/00 CS140(16) 62

Dykstra's Algorithm

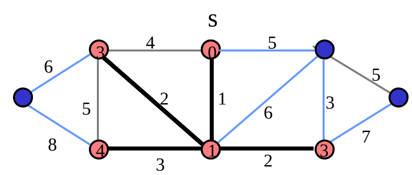
Number in node u indicate $d_G(s,u)$



11/06/00 CS140(16) 63

Dykstra's Algorithm

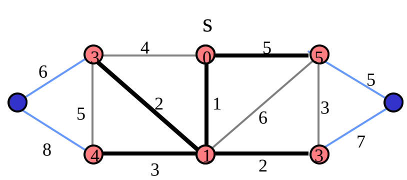
Number in node u indicate $d_G(s,u)$



11/06/00 CS140(16) 64

Dykstra's Algorithm

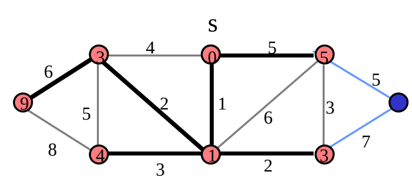
Number in node u indicate $d_G(s,u)$



11/06/00 CS140(16) 65

Dykstra's Algorithm

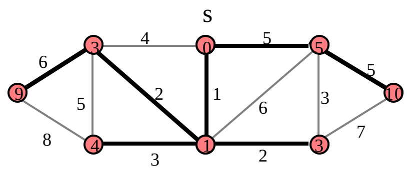
Number in node u indicate $d_G(s,u)$



11/06/00 CS140(16) 66

Dykstra's Algorithm

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



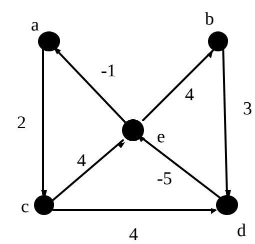
Does this sound familiar?

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

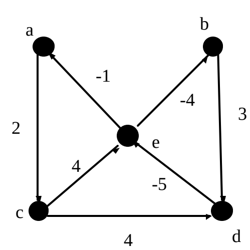
All Pairs Shortest Path (directed version)

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

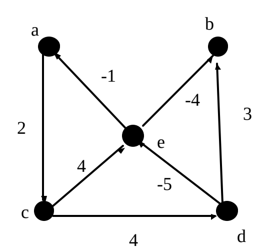
What is d(b,a)?



What is d(b,a)?



What is d(b,a)?



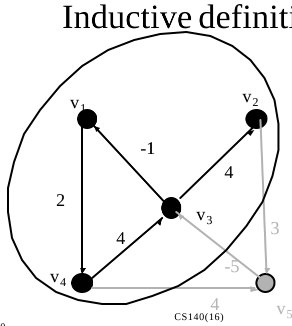
Three Cases:

- 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

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

All Pairs Shortest Path Inductive definition

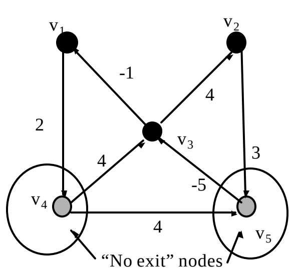


Solve and then what?

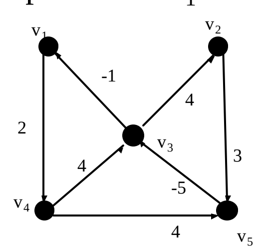
K-limited paths

- 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



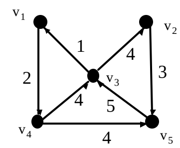
What is the shortest 5-limited path from v_1 to v_2?



Floyd-Warshall algorithm

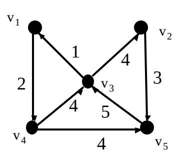
- $D^k(i,j)$ is the length of a shortest k-limited path from v_i to v_j
- $D^k(i,j) = \min(D^{k-1}(i,j),\ D^{k-1}(i,k)+ D^{k-1}(k,j))$
- $D^0(i,j) = w(<v_i,v_j>)$

D^0



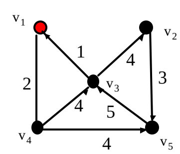
	1	2	3	4	5
1					
2					
3					
4					
5					

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					

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

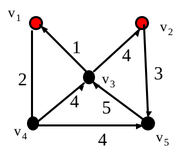
D^0

	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

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

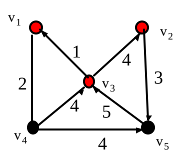
D^1

	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

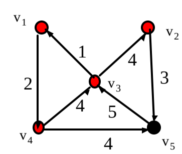
And so on …



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

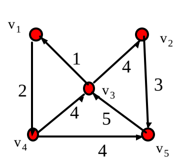
And so on …



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)

⬇

- $O(n^3)$

11/06/00 CS140(16) 91