Graph (continue)

Section 10 in the textbook

1

Graph terminologies (Recap)

1. Graph types
   - Simple graph
   - Multigraph
   - Pseudo graph
   - Directed graphs (Simple directed graph, Directed Multigraph)
   - Mixgraph
2. Special graphs
   - Complete graph
   - Cyles (C_n)
   - Wheels (W_n)
   - n-Cubes (Q_n)
   - Bipartite graphs($K_{m,n}$), Complete Bipartite graphs
3. Adjacent/Neighbor/Incidents
4. Degree of vertex/In-degree Vs. Out-degree
5. Handshaking theorem

$$\sum_{v \in V} \deg(v) = 2 \cdot |E|$$

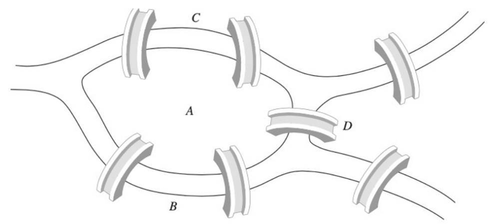
6. Graph representations
   - Adjacency matrices
   - Incidence matrices
7. Isomorphism of graphs
   - Same invariants (vertices/edges/degrees of vertices)
8. Connectivity
   - Simple Path, Simple Circuit
   - Connected/Connectedness Graphs

2

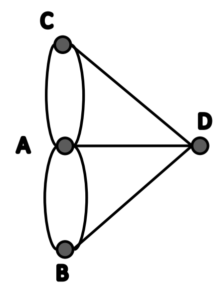
Euler Paths and Circuits

The town of Kö nigsberg, Prussia (now called Kaliningrad and part of the Russian republic), was divided into four sections by the branches of the Pregel River. These four sections included the two regions on the banks of the Pregel, Kneiphof Island, and the region between the two branches of the Pregel. In the eighteenth century seven bridges connected these regions.

The townspeople took long walks through town on Sundays. They wondered whether it was possible to start at some location in the town, travel across all the bridges once without crossing any bridge twice, and return to the starting point.



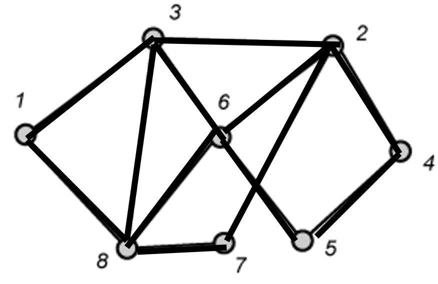
The seven bridges of Kö nigsberg.



Multigraph model of Kö nigsberg

Euler Paths and Circuits

Definition: An *Euler circuit* in a graph *G* is a simple circuit containing every edge of *G*. An *Euler path* in *G* is a simple path containing every edge of *G*.



An Euler circuit is:
1-8-3-6-8-7-2-4-5-6-2-3-1

Euler Paths and Circuits

Theorem: A connected multigraph with at least two vertices has an Euler circuit if and only if each of **its vertices has even degree**.

Proof:

(A) If there's an Euler circuit, then every vertex has even degree

- Suppose the Euler Circuit begin with a vertex a, and continue with an edge incident with a, say $\{a, b\}$. The edge $\{a, b\}$ contribute 1 to $\deg(a)$
- Each time the circuit passes through a vertex, it contributes two to the vertex's degree. (one when entering, and one when leaving)
- Finally, the circuit terminates where it started, contributes on to $\deg(a)$
- Therefore, $\deg(a)$ must be even. A vertex other than a has even degree.

Euler Paths and Circuits

(B) If every vertex has even degree, there is an Euler circuit.

- We will form a simple circuit that begins at an arbitrary vertex a of *G*, building it edge by edge.
- Let $x_0 = a$. First, we arbitrarily choose an edge $\{x_0, x_1\}$ incident with a which is possible because *G* is connected.
- We continue by building a simple path $\{x_0, x_1\}, \{x_1, x_2\}, \ldots, \{x_{n-1}, x_n\}$, successively adding edges one by one to the path until we cannot add another edge to the path.
- The path we have constructed must terminate because the graph has a finite number of edges, so we are guaranteed to eventually reach a vertex for which no edges are available to add to the path.
- The path begins at a with an edge of the form $\{a, x\}$, it must terminate at a with an edge of the form $\{y, a\}$ because, every time we enter and leave a vertex of even degree, there are an even number of edges incident with this vertex that we have not yet used in our path.

Euler Paths and Circuits

- An Euler circuit has been constructed if all the edges have been used. Otherwise, consider the subgraph *H* obtained from *G* by deleting the edges already used and vertices that are not incident with any remaining edges.
- G is connected, H has at least one vertex w in common with the circuit that has been deleted.
- Every vertex in H has even degree because in G all vertices had even degree, and for each vertex, pairs of edges incident with this vertex have been deleted to form H.
- Beginning at w, construct a simple path in *H* by choosing edges as long as possible, as was done in *G*. This path must terminate at w.
- Next, form a circuit in *G* by splicing the circuit in *H* with the original circuit in *G* (this can be done because w is one of the vertices in this circuit). Continue this process until all edges have been used. This produces an Euler circuit.

Euler Paths and Circuits

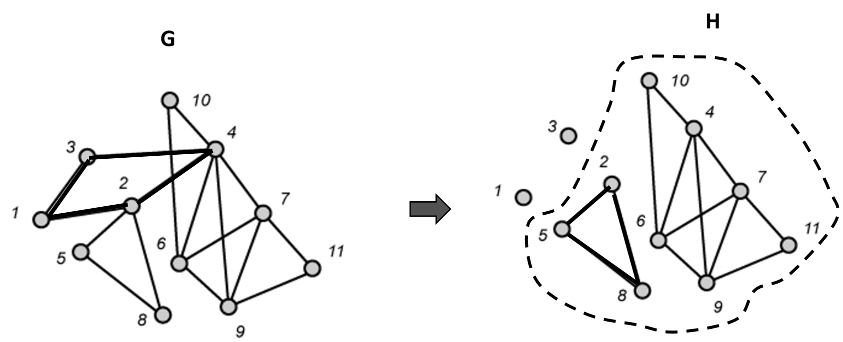
From (A) and (B) the theorem is proven.

```
Procedure Euler(G: connected with all vertices of even degree)
  circuit := a circuit in G beginning at arbitrary vertex.
  H := G with the edges of this circuit removed

  While H has edges
    subcircuit := a circuit in H beginning at a nonisolated vertex
    H:=H with edges of subcircuit and all isolated vertices removed
    circuit := circuit with subcircuit inserted appropriate vertex

Return circuit {circuit is an Euler circuit}
```

Euler path, Euler circuit



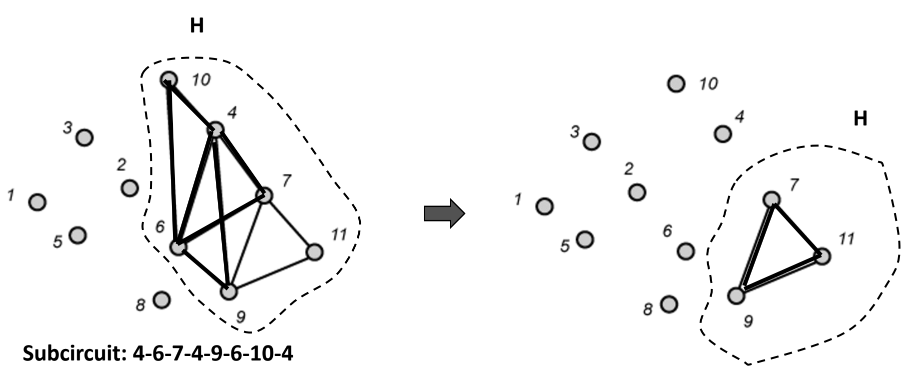
Subcircuit := 2-5-8-2

Circuit: = 1-2-4-3-1

Circuit := 1-2-5-8-2-4-3-1

9

Euler path, Euler circuit



Subcircuit: 4-6-7-4-9-6-10-4

Subcircuit: 7-11-9-7

Circuit: 1-2-5-8-2-4-6-7-4-9-6-10-4-3-1

Circuit: 1-2-5-8-2-4-3-1

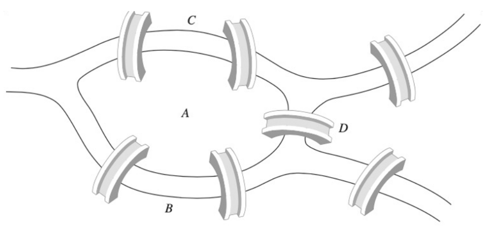
Circuit: 1-2-5-8-2-4-6-7-11-9-4-9-6-10-4-3-1

10

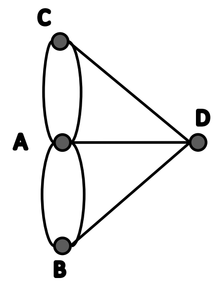
Euler path, Euler circuit

Is there Euler circuit in the town of Ko ̈nigsberg?

Answer: No!



The seven bridges of Ko ̈nigsberg.

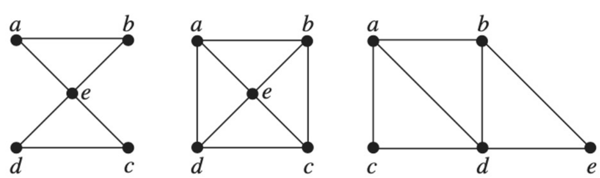


Multigraph model of Ko ̈nigsberg

11

Euler path, Euler circuit

Which of following graphs have and Euler circuit?



G_1 G_2 G_3

Answer:

- G_1 has an Euler circuit a-b-e-d-c-e-a
- G_2 contains vertices of odd degree. It doesn't have Euler circuit
- G_3 contains vertices of odd degree. It doesn't have Euler circuit

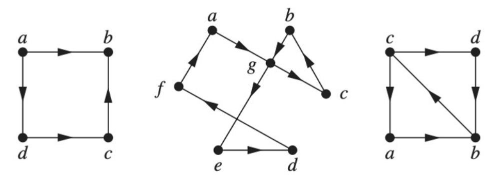
12

Euler path, Euler circuit

How about directed graph? $\sum_{v \in V} \deg(v) = \sum_{v \in V} \deg^{-}(v) + \sum_{v \in V} \deg^{+}(v)$

Directed graphs have an Euler circuit if satisfy following condition

- All vertices with nonzero degree belong to a single strong connected component
- In-degree is equal to out-degree for every vertex



H_1 H_2 H_3

H_1 Doesn't have Euler Circuit

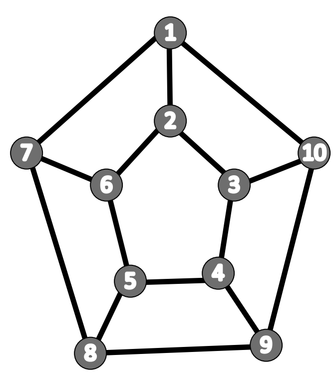
H_2 has Euler circuit a-g-c-b-g-e-d-f-a

H_3 Has Euler path c-a-b-c-d-b.

Hamilton Path and Circuits

Definition

- A simple path in a graph G that passes through every vertex exaclty once is call Hamilton path.
- A simple circuit in a graph G that passes through every vertex exactly once is call Hamilton circuit.



Hamilton circuit: **1-2-3-4-4-6-7-8-9-10-1**

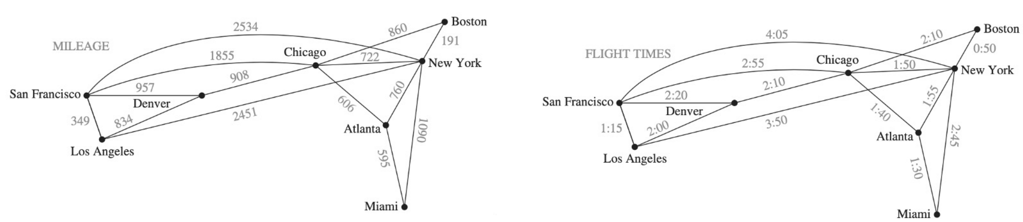
Hamilton Path and Circuits

Dirac's Theorem: If G is a simple graph with n vertices with $n \geq 2$ such that the degree of every vertex in G is at least n/2, then G has a Hamilton circuit

Ore's Theorem: If G is a simple graph with n vertices with $n \geq 3$ such that $\deg(u) + \deg(v) \geq n$ for every pair of nonadjacent vertices u and v in G, then G has a Hamilton circuit.

Shortest-Path Problems

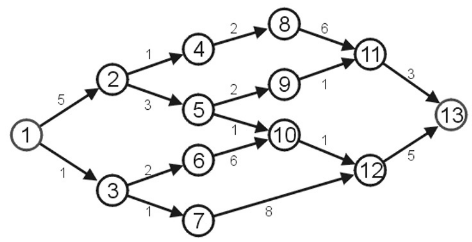
Definition: Graph G that have a number assigned to each edge are called **weighted graphs**. The **length** of a path in a weighted graph is the sum of the weights of the edges of this path.



Given a weighted directed graph, one common problem is finding the shortest path between two given vertices.

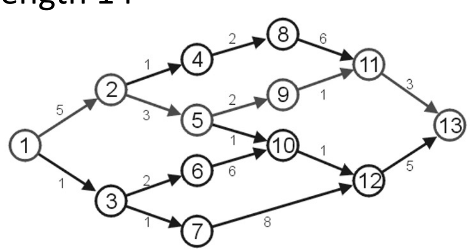
Shortest Path

Given the graph below, suppose we wish to find the shortest path from vertex 1 to vertex 13



Shortest Path

After some consideration, we may determine that the shortest path is as follows, with length 14



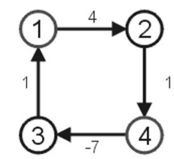
Other paths exists, but they are longer

Negative Cycles

Clearly, if we have negative vertices, it may be possible to end up in a cycle whereby each pass through the cycle decreases the total *length*

Thus, a shortest length would be undefined for such a graph

Consider the shortest path from vertex 1 to 4...



We will only consider non-negative weights.

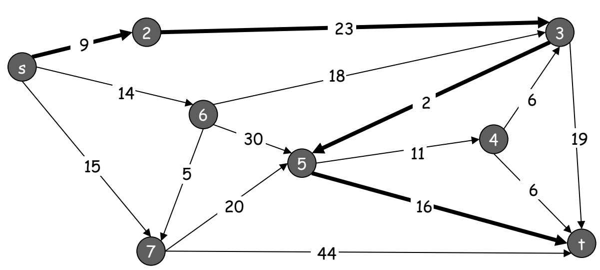
19

Shortest Path Example

Given:

- Weighted Directed graph G = (V, E).
- Source *s*, destination *t*.

Find shortest directed path from s to t.



Cost of path s-2-3-5-t
= 9 + 23 + 2 + 16
= 48.

20

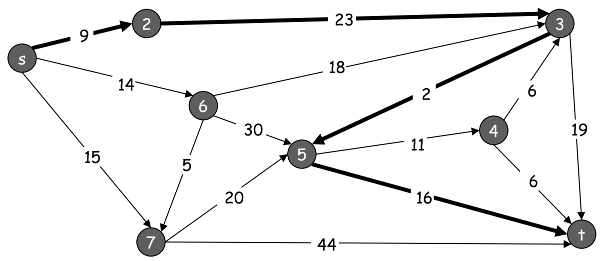
Discussion Items

How many possible paths are there from s to t?

Can we safely ignore cycles? If so, how?

Any suggestions on how to reduce the set of possibilities?

Can we determine a lower bound on the complexity like we did for comparison sorting?



Key Observation

A key observation is that if the shortest path contains the node v, then:

- It will only contain v once, as any cycles will only add to the length.
- The path from s to v must be the shortest path to v from s.
- The path from v to t must be the shortest path to t from v.

Thus, if we can determine the shortest path to all other vertices that are incident to the target vertex we can easily compute the shortest path.

- Implies a set of sub-problems on the graph with the target vertex removed.

Dijkstra's Algorithm

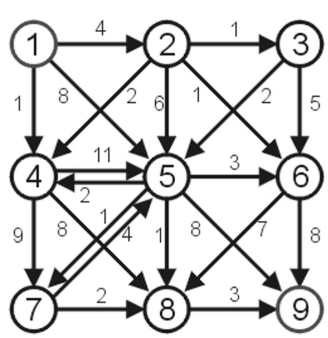
Works when all of the weights are positive.

Provides the shortest paths from a source to **all** other vertices in the graph.

- Can be terminated early once the shortest path to *t* is found if desired.

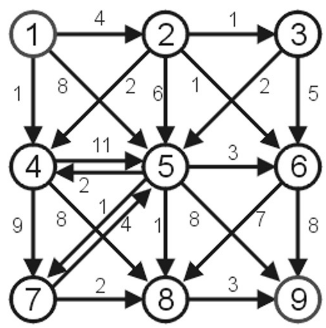
Shortest Path

Consider the following graph with positive weights and cycles.



Dijkstra’s Algorithm

A first attempt at solving this problem might require an array of Boolean values, all initially false, that indicate whether we have found a path from the source.

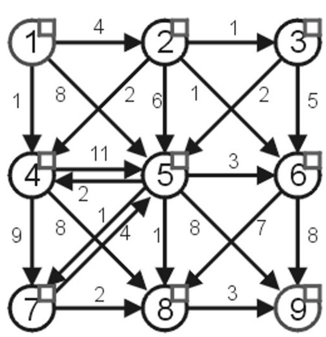


1	F
2	F
3	F
4	F
5	F
6	F
7	F
8	F
9	F

25

Dijkstra’s Algorithm

Graphically, we will denote this with check boxes next to each of the vertices (initially unchecked)



26

Dijkstra’s Algorithm

We will work bottom up.

- Note that if the starting vertex has any adjacent edges, then there will be one vertex that is the shortest distance from the starting vertex. This is the shortest reachable vertex of the graph.

We will then try to extend any ***existing*** paths to new vertices.

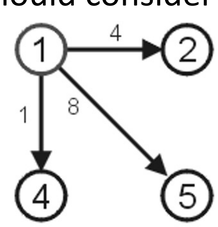
Initially, we will start with the path of length 0

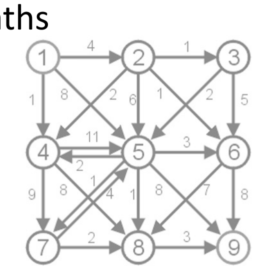
- this is the trivial path from vertex 1 to itself

Dijkstra’s Algorithm

If we now extend this path, we should consider the paths

- (1, 2) length 4
- (1, 4) length 1
- (1, 5) length 8



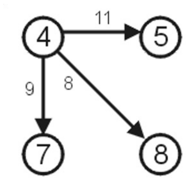


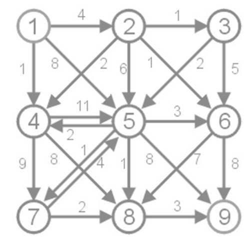
The *shortest* path so far is (1, 4) which is of length 1.

Dijkstra’s Algorithm

Thus, if we now examine vertex 4, we may deduce that there exist the following paths:

- (1, 4, 5) length 12
- (1, 4, 7) length 10
- (1, 4, 8) length 9



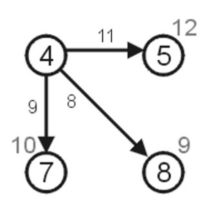


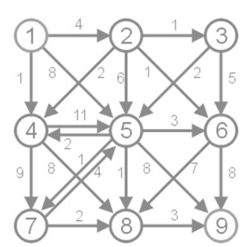
Dijkstra’s Algorithm

We need to remember that the length of that path from node 1 to node 4 is 1

Thus, we need to store the length of a path that goes through node 4:

- 5 of length 12
- 7 of length 10
- 8 of length 9

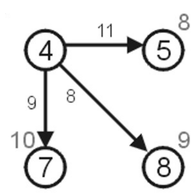


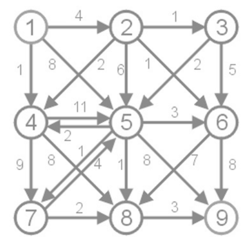


Dijkstra’s Algorithm

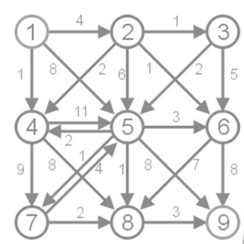
We have already discovered that there is a path of length 8 to vertex 5 with the path (1, 5).

Thus, we can safely ignore this longer path.





Dijkstra’s Algorithm



We now know that:

- There exist paths from vertex 1 to vertices {2,4,5,7,8}.
- We know that the shortest path from vertex 1 to vertex 4 is of length 1.
- We know that the shortest path to the other vertices {2,5,7,8} is at most the length listed in the table to the right.

Vertex	Length
1	0
2	4
4	1
5	8
7	10
8	9

Dijkstra’s Algorithm

There cannot exist a shorter path to either of the vertices 1 or 4, since the distances can only increase at each iteration.

We consider these vertices to be ***visited***

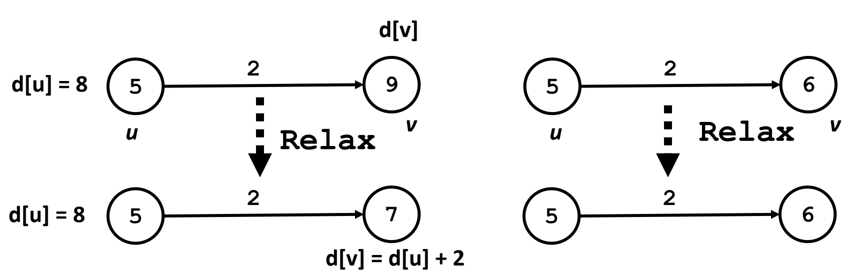
If you only knew this information and nothing else about the graph, what is the possible lengths from vertex 1 to vertex 2? What about to vertex 7?

Vertex	Length
1	0
2	4
4	1
5	8
7	10
8	9

Relaxation

Maintaining this shortest discovered distance d[*v*] is called **relaxation**:

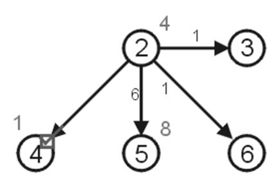
```
Relax(u,v,w) {
  if (d[v] > d[u]+w) then
    d[v]=d[u]+w;
}
```



Dijkstra’s Algorithm

In Dijkstra’s algorithm, we always take the next unvisited vertex which has the current shortest path from the starting vertex in the table.

This is vertex 2

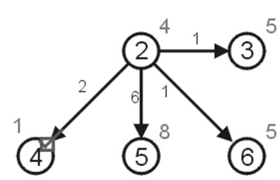


Vertex	Length
1	0
2	4
4	1
5	8
7	10
8	9

Dijkstra’s Algorithm

We can try to update the shortest paths to vertices 3 and 6 (both of length 5) however:

- there already exists a path of length 8 < 10 to vertex 5 (10 = 4 + 6)
- we already know the shortest path to 4 is 1



Dijkstra's Algorithm

To keep track of those vertices to which no path has reached, we can assign those vertices an initial distance of either

- infinity (∞),
- a number larger than any possible path, or
- a negative number

For demonstration purposes, we will use ∞

As well as finding the length of the shortest path, we'd like to find the corresponding shortest path

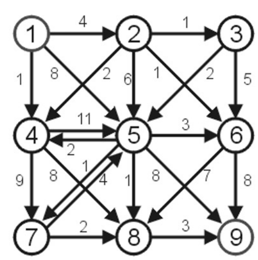
Each time we update the shortest distance to a particular vertex, we will keep track of the predecessor used to reach this vertex on the shortest path.

Dijkstra's Algorithm

We will store a table of pointers, each initially 0. This table will be updated each time a distance is updated

Graphically, we will display the reference to the preceding vertex by a red arrow

- if the distance to a vertex is ∞, there will be no preceding vertex
- otherwise, there will be exactly one preceding vertex



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

Dijkstra's Algorithm

Thus, for our initialization:

- we set the current distance to the initial vertex as 0
- for all other vertices, we set the current distance to ∞
- all vertices are initially marked as unvisited
- set the previous pointer for all vertices to null

39

Dijkstra's Algorithm

Thus, we iterate:

- find an unvisited vertex which has the shortest distance to it
- mark it as visited
- for each unvisited vertex which is adjacent to the current vertex:
  - add the distance to the current vertex to the weight of the connecting edge
  - if this is less than the current distance to that vertex, update the distance and set the parent vertex of the adjacent vertex to be the current vertex

40

Dijkstra's Algorithm

Halting condition:

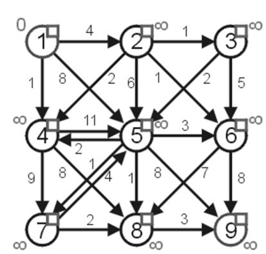
- we successfully halt when the vertex we are visiting is the target vertex
- if at some point, all remaining unvisited vertices have distance ∞, then no path from the starting vertex to the end vertex exits

Note: We do not halt just because we have updated the distance to the end vertex, we have to **visit** the target vertex.

Example

Consider the graph:

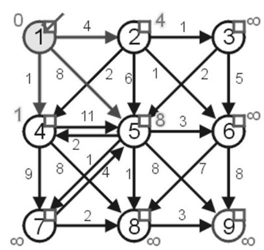
- the distances are appropriately initialized
- all vertices are marked as being unvisited



Example

Visit vertex 1 and update its neighbours, marking it as visited

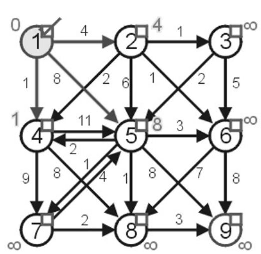
- the shortest paths to 2, 4, and 5 are updated

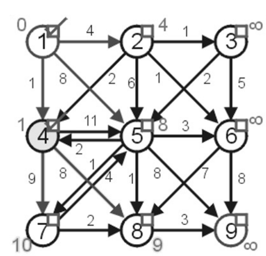


Example

The next vertex we visit is vertex 4

- vertex 5 $1 + 11 \geq 8$ don't update
- vertex 7 $1 + 9 < \infty$ update
- vertex 8 $1 + 8 < \infty$ update

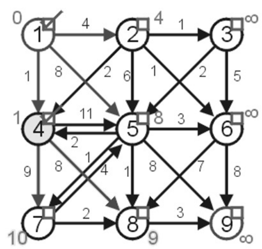


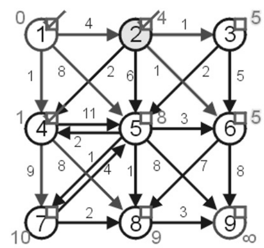


Example

Next, visit vertex 2

- vertex 3 $4 + 1 < \infty$ update
- vertex 4 already visited
- vertex 5 $4 + 6 \geq 8$ don't update
- vertex 6 $4 + 1 < \infty$ update



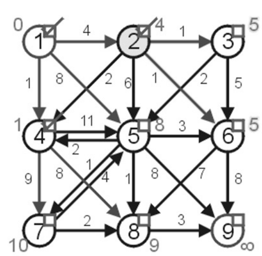


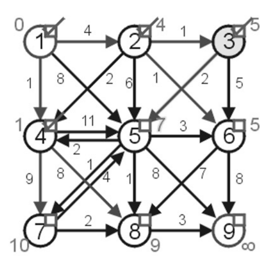
Example

Next, we have a choice of either 3 or 6

We will choose to visit 3

- vertex 5 $5 + 2 < 8$ update
- vertex 6 $5 + 5 \geq 5$ don't update

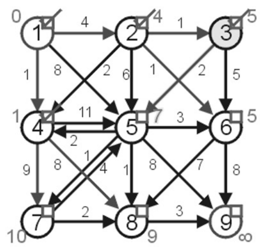


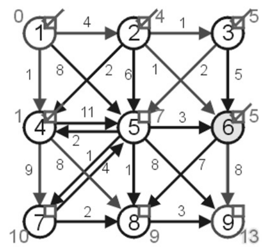


Example

We then visit 6

- vertex 8 $5 + 7 \geq 9$ don't update
- vertex 9 $5 + 8 < \infty$ update



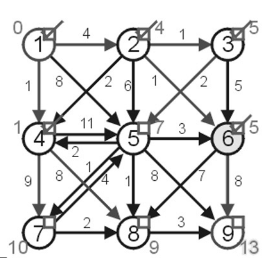


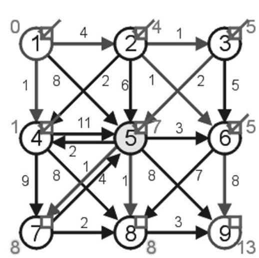
47

Example

Next, we finally visit vertex 5:

- vertices 4 and 6 have already been visited
- vertex 7 $7 + 1 < 10$ update
- vertex 8 $7 + 1 < 9$ update
- vertex 9 $7 + 8 \geq 13$ don't update



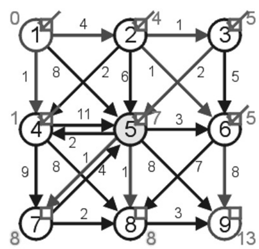


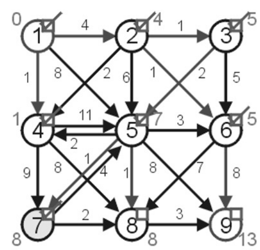
48

Example

Given a choice between vertices 7 and 8, we choose vertex 7

- vertices 5 has already been visited
- vertex 8 $\quad 8 + 2 \geq 8 \quad$ don’t update

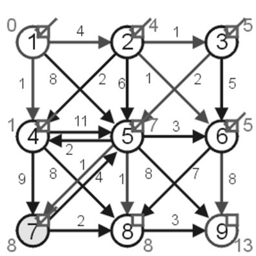


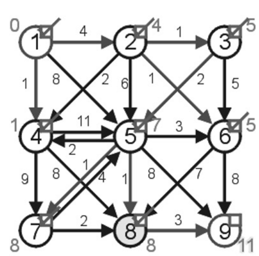


Example

Next, we visit vertex 8:

- vertex 9 $\quad 8 + 3 < 13 \quad$ update

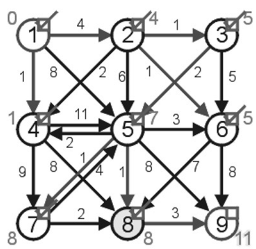


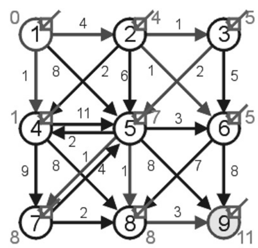


Example

Finally, we visit the end vertex

Therefore, the shortest path from 1 to 9 has length 11



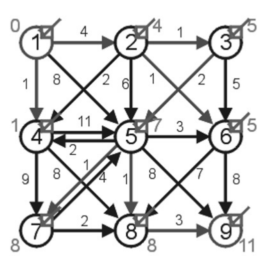


Example

We can find the shortest path by working back from the final vertex:

- 9, 8, 5, 3, 2, 1

Thus, the shortest path is (1, 2, 3, 5, 8, 9)



Example

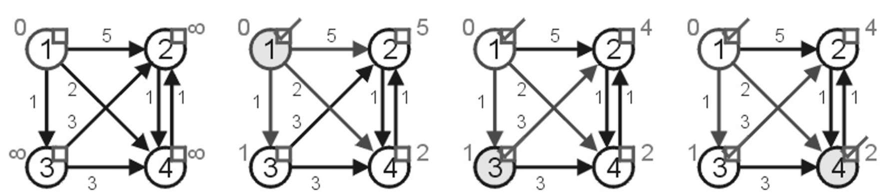
In the example, we visited all vertices in the graph before we finished

This is not always the case, consider the next example

Example

Find the shortest path from 1 to 4:

- the shortest path is found after only three vertices are visited
- we terminated the algorithm as soon as we reached vertex 4
- we only have useful information about 1, 3, 4
- we don't have the shortest path to vertex 2



Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \emptyset$
$Q \leftarrow V$ ⊳ Q is a priority queue maintaining $V - S$
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$ **do**
      if $d[v] > d[u] + w(u, v)$ **then**
        $d[v] \leftarrow d[u] + w(u, v)$
        $p[v] \leftarrow u$

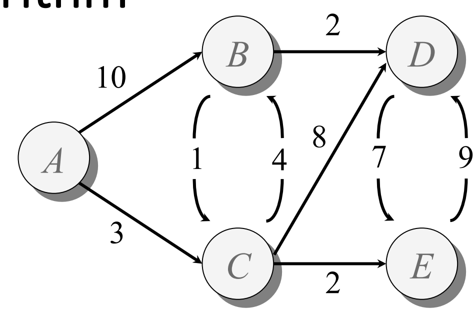
Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \emptyset$
$Q \leftarrow V$ ⊳ Q is a priority queue maintaining $V - S$
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$ **do**
      if $d[v] > d[u] + w(u, v)$ **then**
        $d[v] \leftarrow d[u] + w(u, v)$ ***relaxation step***
        $p[v] \leftarrow u$

Implicit DECREASE-KEY

Example of Dijkstra's algorithm

Graph with nonnegative edge weights:



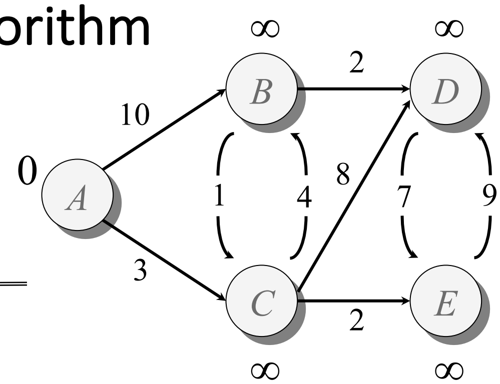
Example of Dijkstra's algorithm

Initialize

S: {}

Q:

A d(A), P(A)	*B* d(B), P(B)	*C* d(C), P(C)	*D* d(D), P(D)	*E* d(E), P(E)
0, *A*	∞, –	∞, –	∞, –	∞, –

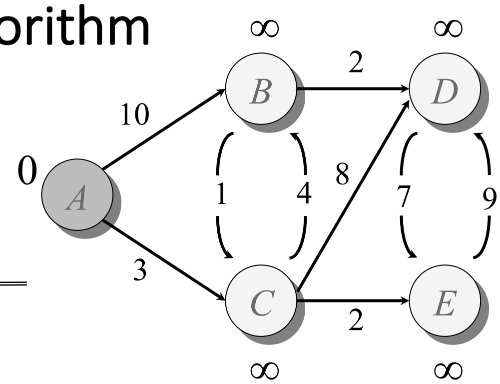


Example of Dijkstra's algorithm

"A" ← EXTRACT-MIN(Q):

S: $\{A\}$

Q:	A d(A), P(A)	B d(B), P(B)	C d(C), P(C)	D d(D), P(D)	E d(E), P(E)
	0, A	∞, –	∞, –	∞, –	∞, –



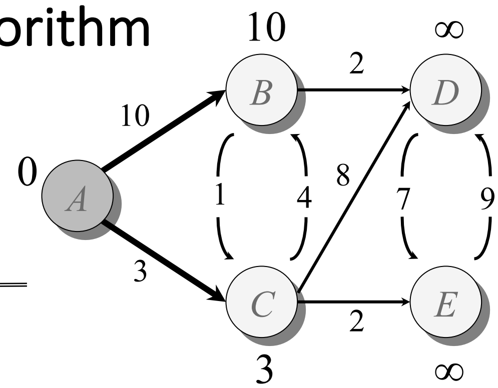
59

Example of Dijkstra's algorithm

Relax all edges leaving A:

S: $\{A\}$

Q:	A d(A), P(A)	B d(B), P(B)	C d(C), P(C)	D d(D), P(D)	E d(E), P(E)
	0, A	∞, –	∞, –	∞, –	∞, –
		10, A	3, A	∞, –	∞, –



60

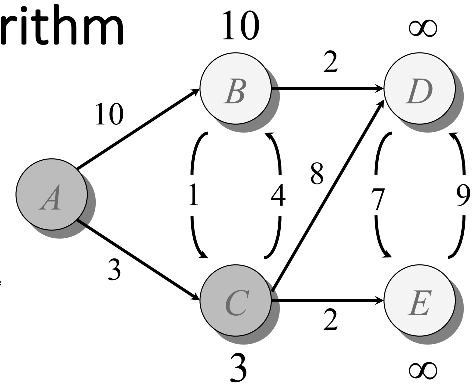
Example of Dijkstra's algorithm

***"C"* ← EXTRACT-MIN(*Q*):**

S: {*A*, *C*}

Q:

A d(A), P(A)	*B* d(B), P(B)	*C* d(C), P(C)	*D* d(D), P(D)	*E* d(E), P(E)
0	∞, –	∞, –	∞, –	∞, –
	10, *A*	3, *A*	∞, –	∞, –



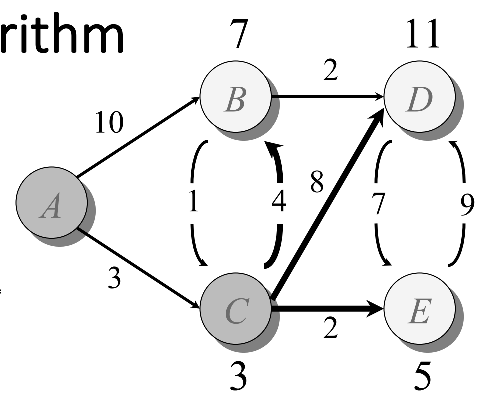
Example of Dijkstra's algorithm

Relax all edges leaving *C*:

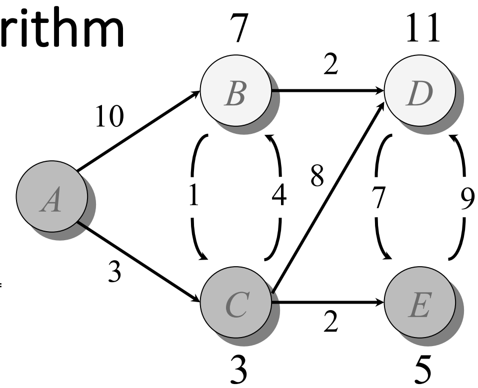
S: {*A*, *C*}

Q:

A d(A), P(A)	*B* d(B), P(B)	*C* d(C), P(C)	*D* d(D), P(D)	*E* d(E), P(E)
0	∞, –	∞, –	∞, –	∞, –
	10, *A*	3, *A*	∞, –	∞, –
	7, *C*		11, *C*	5, *C*



Example of Dijkstra's algorithm

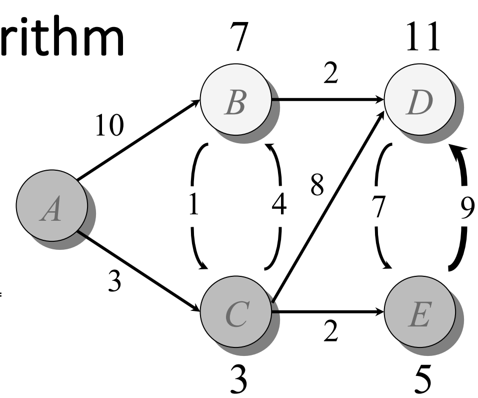


"E" ← EXTRACT-MIN(Q):

S: {A, C, E}

Q: A d(A), P(A)	B d(B), P(B)	C d(C), P(C)	D d(D), P(D)	E d(E), P(E)
0	∞, –	∞, –	∞, –	∞, –
	10, A	3, A	∞, –	∞, –
	7, C		11, C	5, C

Example of Dijkstra's algorithm



Relax all edges leaving E:

S: {A, C, E}

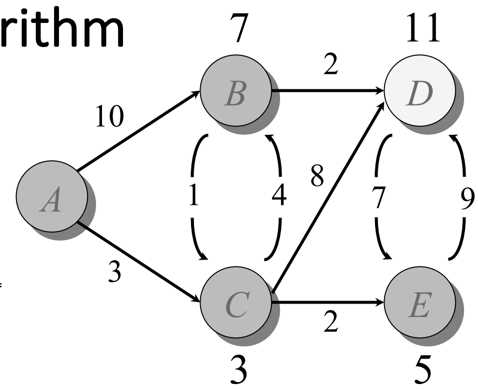
Q: A d(A), P(A)	B d(B), P(B)	C d(C), P(C)	D d(D), P(D)	E d(E), P(E)
0	∞, –	∞, –	∞, –	∞, –
	10, A	3, A	∞, –	∞, –
	7, C		11, C	5, C
	7, C		11, C	

Example of Dijkstra's algorithm

"*B*" ← EXTRACT-MIN(*Q*):

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

Q: *A* d(A), P(A)	*B* d(B), P(B)	*C* d(C), P(C)	*D* d(D), P(D)	*E* d(E), P(E)
0	∞, –	∞, –	∞, –	∞, –
	10, *A*	3, *A*	∞, –	∞, –
	7, *C*		11, *C*	5, *C*
	7, *C*		11, *C*	



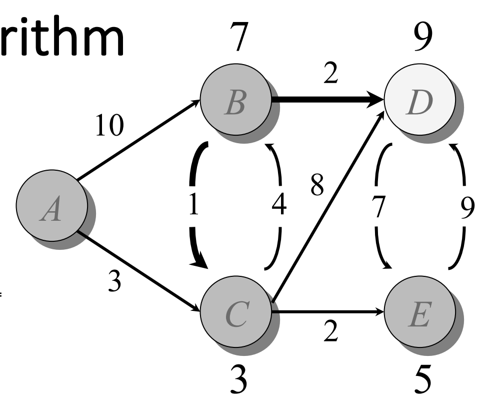
UMass Boston

Example of Dijkstra's algorithm

Relax all edges leaving *B*:

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

Q: *A* d(A), P(A)	*B* d(B), P(B)	*C* d(C), P(C)	*D* d(D), P(D)	*E* d(E), P(E)
0	∞, –	∞, –	∞, –	∞, –
	10, *A*	3, *A*	∞, –	∞, –
	7, *C*		11, *C*	5, *C*
	7, *C*		11, *C*	
			9, *B*	



UMass Boston

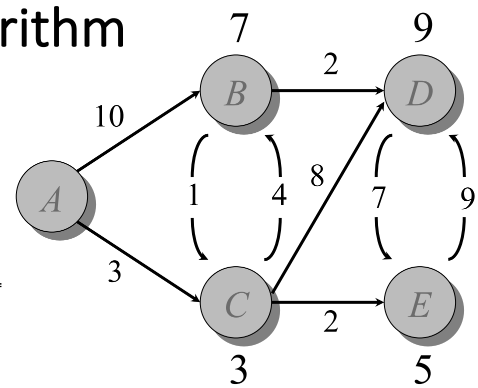
Example of Dijkstra's algorithm

***"D"* ← EXTRACT-MIN(*Q*):**

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

Q:

A d(A), P(A)	*B* d(B), P(B)	*C* d(C), P(C)	*D* d(D), P(D)	*E* d(E), P(E)
0	∞, –	∞, –	∞, –	∞, –
	10, *A*	3, *A*	∞, –	∞, –
	7, *C*		11, *C*	5, *C*
	7, *C*		11, *C*	
			9, *B*	

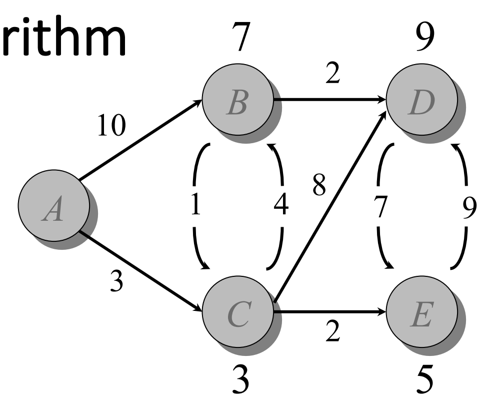


UMass Boston

67

Example of Dijkstra's algorithm

A d(A), P(A)	*B* d(B), P(B)	*C* d(C), P(C)	*D* d(D), P(D)	*E* d(E), P(E)
0	∞, –	∞, –	∞, –	∞, –
	10, *A*	3, *A*	∞, –	∞, –
	7, *C*		11, *C*	5, *C*
	7, *C*		11, *C*	
			9, *B*	



From A

Destination	Length	Path
B	7	A-C-B
C	3	A-C
D	9	A-C-B-D
E	5	A-C-E

UMass Boston

68

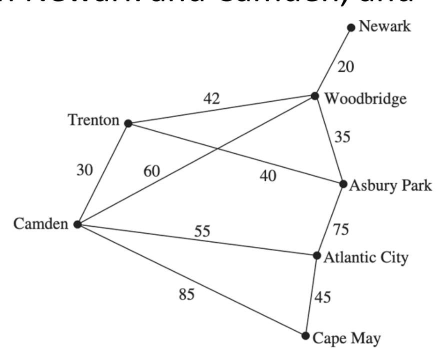
Summary

Given a weighted directed graph, we can find the shortest distance between two vertices by:

- starting with a trivial path containing the initial vertex
- growing this path by always going to the next vertex which has the shortest current path

In-class exercise

The weighted graphs shows the distances between cities in New Jersey. Find a shortest route in distance between Newark and Camden, and between Newark and Cape May



Solution

N $d(N), P(N)$	W $d(W), P(W)$	As $d(As), P(As)$	At $d(At), P(At)$	CM $d(CM), P(CM)$	CA $d(CA), P(CA)$	T $d(T), P(T)$
0, N	∞, –	∞, –	∞, –	∞, –	∞, –	∞, –
	20, N	∞, –	∞, –	∞, –	∞, –	∞, –
		55, W	∞, –	∞, –	80, W	62, W
			130, As	∞, –	80, W	62, W
			130, As	∞, –	80, W	
			130, As	165, CA		
				165, CA		

N : Newark
W : wood bridge
As : Asbury park
At : Atlantic City
CM : Cape May
CA : Camden
T : Trenton

from Newark (N) to :

W	20	N – W
As	55	N – W – As
At	130	N – W – AS – At
CM	165	N – W – CA – CM
CA	80	N – W – CA
T	62	N – W – T

length

UMass Boston