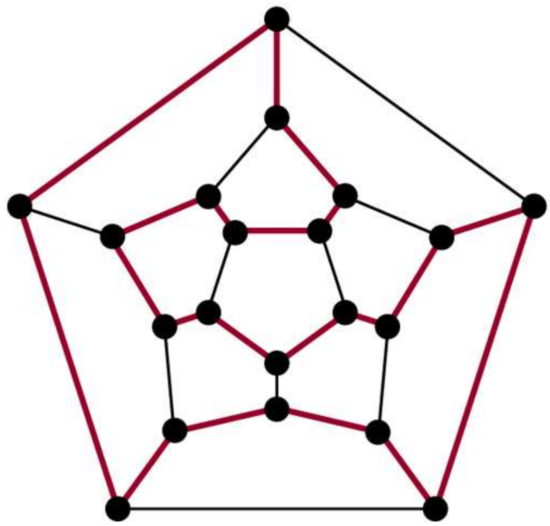
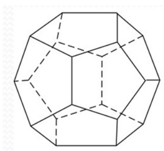
10.5.2 Hamilton paths and circuit

Hamilton's puzzle

The object of the puzzle was to start at a city and travel along the edges of the dodecahedron, visiting each of the other 19 cities exactly once, and end back at the first city.

The equivalent question: Is there a circuit in the graph shown in b that passes through each vertex exactly once?





A ***Hamilton path*** in a graph *G* is a path which visits every vertex in *G* exactly once.

A ***Hamilton circuit*** (or ***Hamilton cycle***) is a cycle which visits every vertex exactly once, ***except for the first vertex***, which is also visited at the end of the cycle.

If a connected graph *G* has a Hamilton circuit, then *G* is called a ***Hamilton graph***.

Question:

1) H path is a simple path?

1. The sufficient condition for the existence of Hamilton path and Hamilton circuit

【Theorem 3】 DIRAC'THEOREM

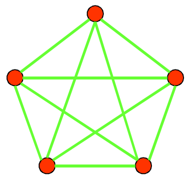
If G is a simple graph with n vertices with n>=3 such that the degree of every vertex in G is at least $n/2$, then G has a Hamilton circuit.

【Theorem 4】ORE'THEOREM

If G is a simple graph with n vertices with n>=3 such that deg(u)+deg(v) >= n for every pair of nonadjacent vertices u and v in G , then G has a Hamilton circuit.

〖Example 4〗 Show that K_n has a Hamilton circuit whenever $n>=3$?

For example,



Proof:

We can form a Hamilton circuit in K_n beginning at any vertex. Such a circuit can be built by visiting vertices in any order we choose, as long as the path begins and ends at the same vertex and visits each other vertex exactly once. This is possible since there are edges in K_n between any two vertices.

2. The necessary condition for Hamilton path and Hamilton circuit

For undirected graph:

The necessary condition for the existence of Hamilton path:

- G is connected;
- There are at most two vertices which degree are less than 2.

2. The necessary condition for Hamilton path and Hamilton circuit

The necessary condition for the existence of Hamilton circuit:

- **The degree of each vertex is larger than 1.**

Some properties:

- **If a vertex in the graph has degree two, then both edges that are incident with this vertex must be part of any Hamilton circuit.**
- **When a Hamilton circuit is being constructed and this circuit has passed through a vertex, then all remaining edges incident with this vertex, other than the two used in the circuit , can be removed from consideration.**

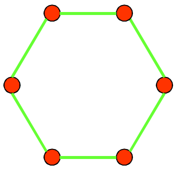
Another important necessary condition

- **G is a Hamilton graph, for any nonempty subset *S* of set *V*, the number of connected components in *G-S* <=|*S*|.**

Note:

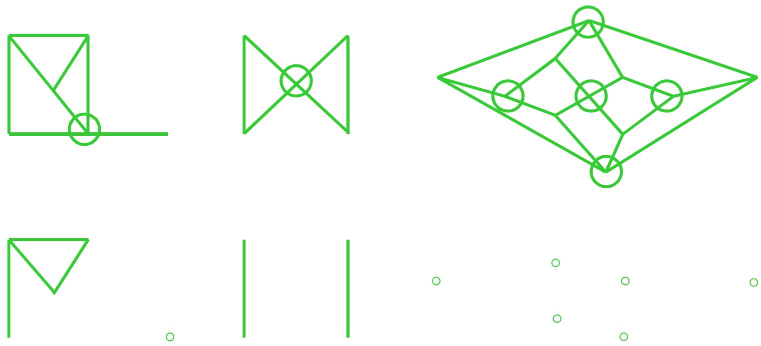
(1)* *G-S* is a subgraph of *G

(2) Suppose that *C* is a H circuit of *G*. For any nonempty subset *S* of set *V*, the number of connected components in *C-S* <=|*S*|.



(3) the number of connected components in *G-S* <= the number of connected components in *C-S*

〖Example 5〗 Does the following graphs have a Hamilton circuit?



3. Applications

Hamilton path or circuit can be used to solve many practical problems also.

For example,

1) Find a path or circuit that visits each road intersection in a city, or each node in a communication network exactly once.

2) The famous traveling salesman problem (TSP)

3) ……

〖Example 6〗 There are seven people denoted by A, B, C, D, E, F, G. Suppose that the following facts are known.

A--English (A can speak English.)

B--English, Chinese

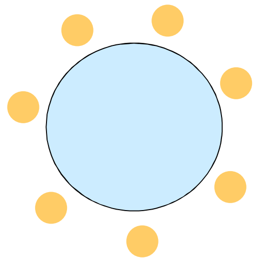
C--English, Italian, Russian

D--Japanese, Chinese

E--German, Italia

F--French, Japanese, Russian

G--French, German

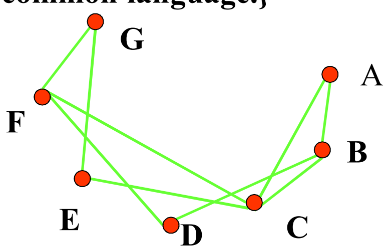


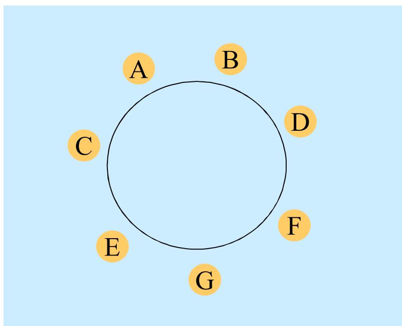
How to arrange seat for the round desk such that the seven people can talk each other?

Solution:

(1) Construct graph

V={A,B,C,D,E,F,G}, E={(u,v)|u,v can speak at least one common language.}





(2) If there is a H circuit, then we can arrange seat for the round desk such that the seven people can talk each other.

H circuit: A,B,D,F,G,E,C,A

Homework:

Sec. 10.5 4, 6, 31, 34, 38, 41

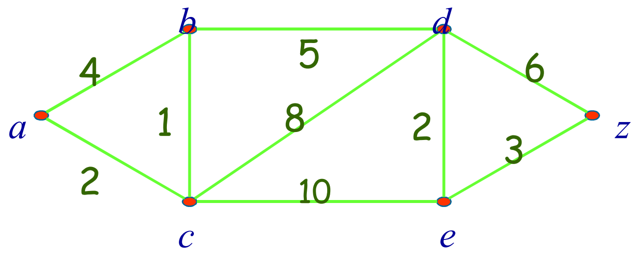
CHAPTER 10 Graphs

1. Introduction

- *Weighted graph* $G = (V, E, W)$

  We can assign weights to the edges of graphs.

For example,



10.6 Shortest Path Problems

1. Introduction

- *Weighted graph* $G = (V, E, W)$

  We can assign weights to the edges of graphs.

- *The length of a path in a weighted graph*

  -- the sum of the weights of the edges of this path.

- *Shortest Path Problems*

  $G=(V,E,W)$ is a weighted graph, where $w(x,y)$ is the weight of edge (x, y) (if $(x,y) \notin E, w(x,y) = \infty$). $a, z \in V$, find the shortest path between a and z.

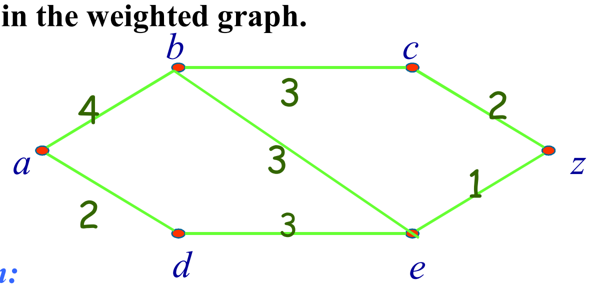
2. A shortest path algorithm

There are several different algorithms that find the shortest path between two vertices in a weighted graph.

Dijkstra's Algorithm (undirected graph with positive weights)

- An algorithm discovered by the Dutch mathematician E. Dijkstra in 1959.
- An iterative procedure.
- Proceed by finding the length of the shortest path from *a* to a first vertex, the length of the shortest path from *a* to a second vertex, and so on, until the length of the shortest path from *a* to *z* is found.

〖Example 1〗 What is the length of the shortest path between *a* and *z* in the weighted graph.



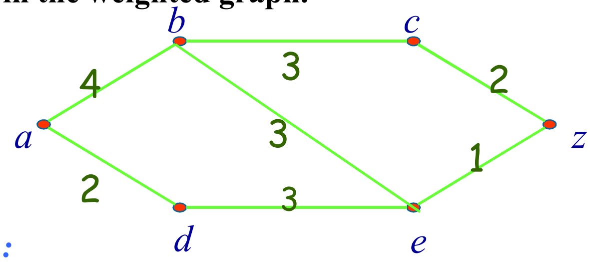
Solution:

We will solve this problem by finding the length of a shortest path from *a* to successive vertices, until *z* is reached.

1) The first closest vertex: *d*

The only paths starting at *a* that contain no vertex other than *a* are *a*, *b* and *a*, *d*.

〖Example 1〗 What is the length of the shortest path between *a* and *z* in the weighted graph.



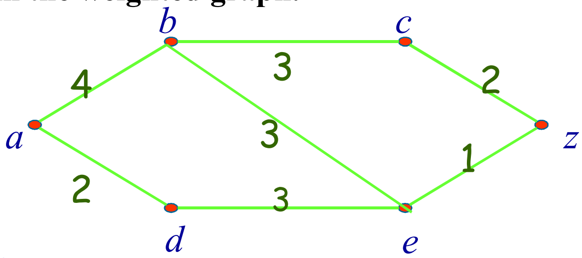
Solution:

1) The first closest vertex: *d*

2) The second closest vertex: *b*

Looking at all paths that go through only *a* and *d*.

〖Example 1〗 What is the length of the shortest path between *a* and *z* in the weighted graph.



Solution:

1) The first closest vertex: *d*

2) The second closest vertex: *b*

3) The third closest vertex: *e*

Examine only paths go through only *a*, *d* and *b*.

4) The forth closest vertex: *z*

The details of Dijkstra's algorithm

Dijkstra's algorithm proceeds by forming a distinguished set of vertices iteratively. Let S_k denote this set of vertices after k iterations of labeling procedure.

Step1 **Label a with 0 and other with ∞, i.e. $L_0(a)=0$, and $L_0(v)=\infty$ and $S_0=\phi$.**

Step 2 **The set S_k is formed from S_{k-1} by adding a vertex u not in S_{k-1} with the smallest label. Once u is added to S_k, we update the labels of all vertices not in S_k, so that $L_k(v)$, the label of the vertex v at the kth stage, is the length of the shortest path from a to v that containing vertices only in S_k.**

10.6 Shortest Path Problems

Let v be a vertex not in S_k. To update the label of v, note that $L_k(v)$ is the shortest path from a to v containing only vertices in S_k,

1. **the shortest path from a to v containing only elements of S_{k-1}**

   Or

2. **it is the shortest path from a to u at the (k-1)st stage with the edge (u,v) added.**

In other words,

$$L_k(v)=\min\{L_{k-1}(v), L_{k-1}(u)+w(u,v)\}$$

Algorithm 1 Dijkstra's Algorithm.

Procedure Dijkstra(G: weighted connected simple graph, with all weights positive)

{G has vertices $a = v_0, v_1, \cdots, v_n = z$ and weights $w(v_i, v_j)$

where $w(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ is not an edge in G}

For i:= 1 to n

$L(v_i) := \infty$

$L(a)$:=0

$S := \emptyset$

{the labels are now initialized so that the label of a is zero and all other labels are ∞,and S is the empty set }

While $z \notin S$

Begin

u :=a vertex not in S with $L(u)$ minimal

$S := S \cup \{u\}$

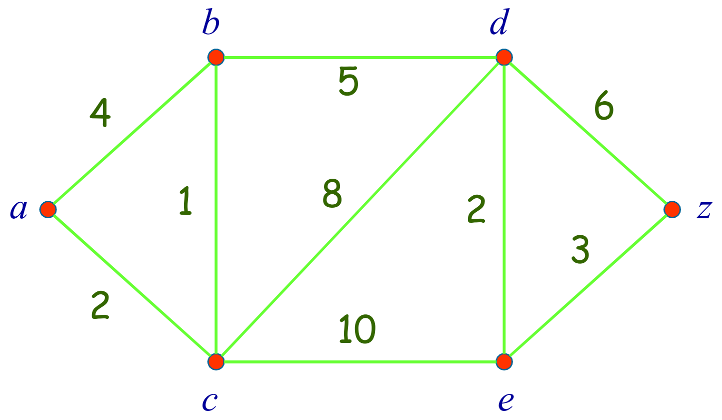
for all vertices v not in S

if $L(u) + w(u,v) < L(v)$ $L(v) := L(u) + w(u,v)$

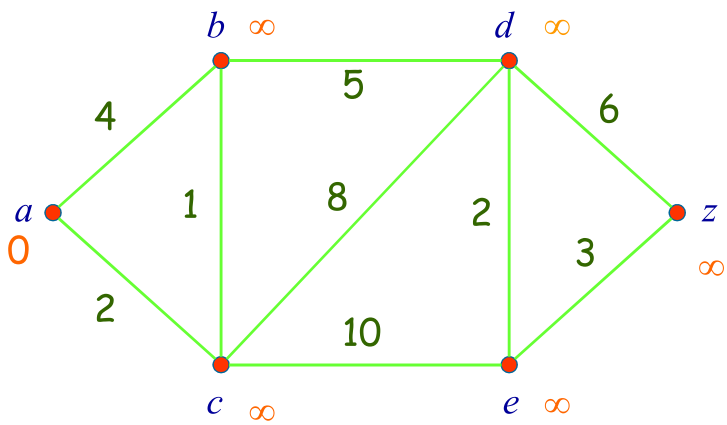
{this adds a vertex to S with minimal label and updates the labels of vertices not in S}

End {$L(z)$=length of shortest path from a to z }

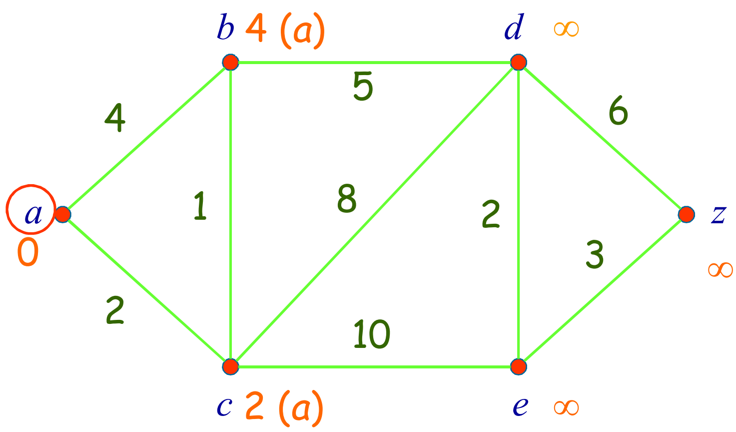
〖Example 1〗 Find the length of the shortest path between *a* and *z* in the given weighted graph.



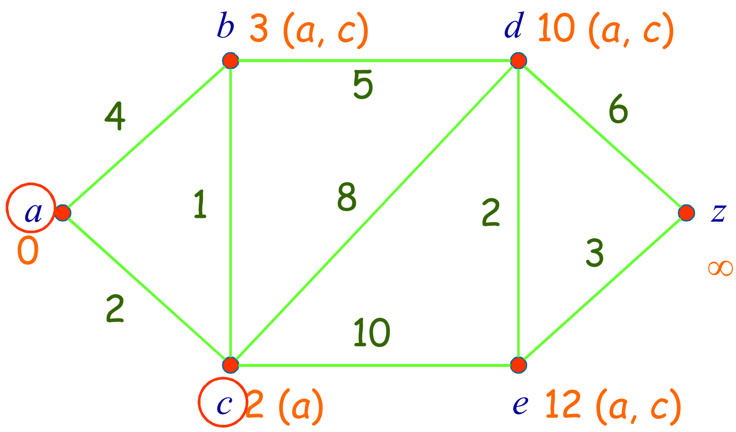
Step 0



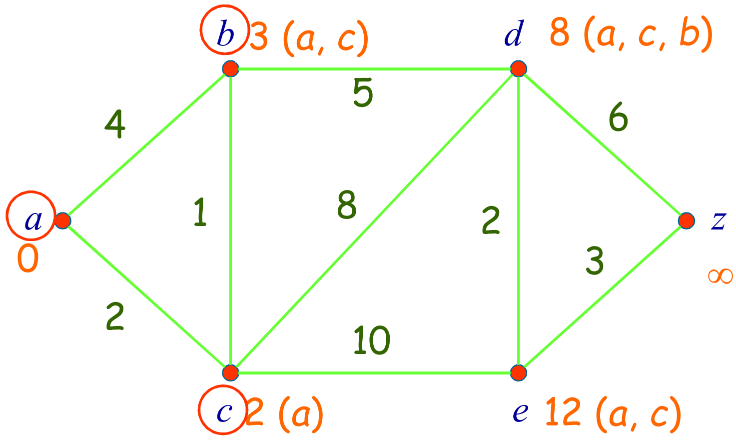
Step 1


b 4 (a)
d ∞
5
4
6
a
0
1
8
2
z
3
∞
2
10
c 2 (a)
e ∞

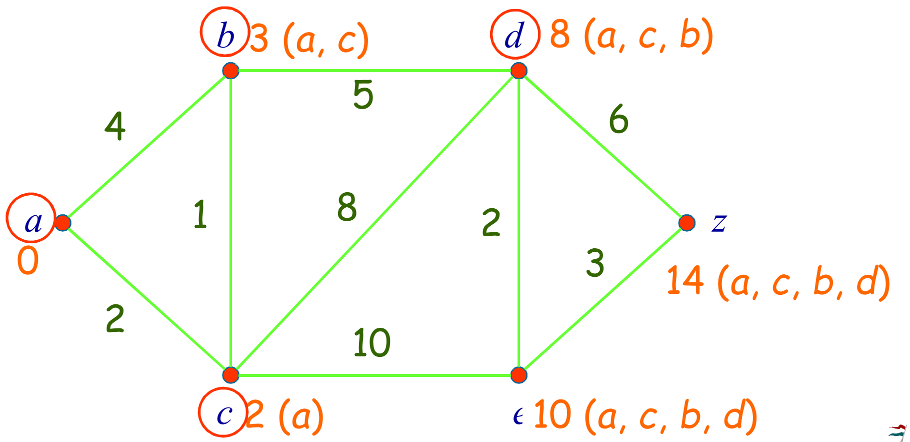
Step 2


b 3 (a, c)
d 10 (a, c)
5
4
6
a
0
1
8
2
z
3
∞
2
10
c 2 (a)
e 12 (a, c)

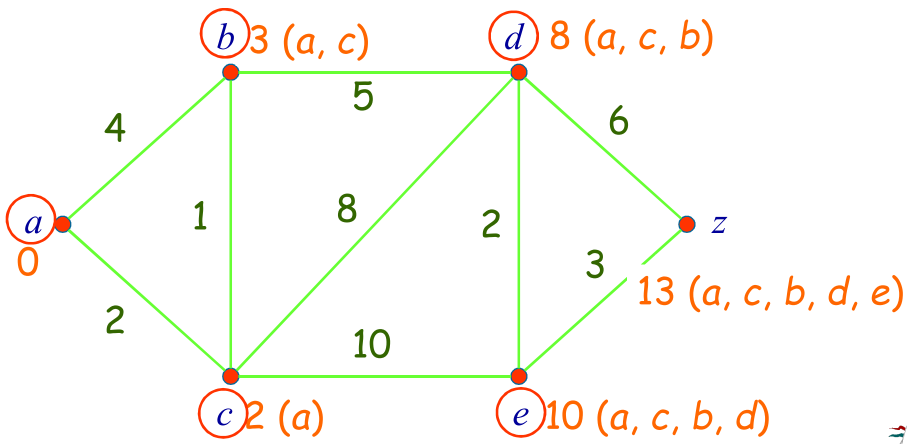
Step 3


b 3 (a, c)
d 8 (a, c, b)
a
0
z
∞
c 2 (a)
e 12 (a, c)
4
5
6
1
8
2
3
2
10

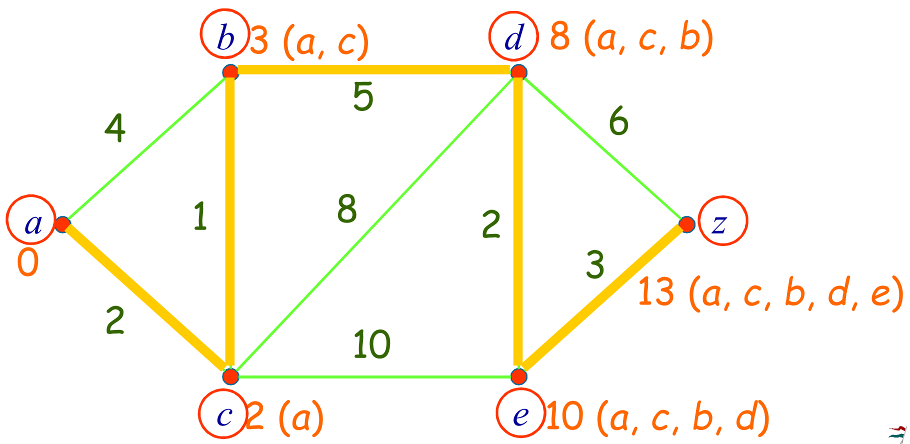
Step 4


b 3 (a, c)
d 8 (a, c, b)
a
0
z
14 (a, c, b, d)
c 2 (a)
10 (a, c, b, d)
4
5
6
1
8
2
3
2
10

Step 5

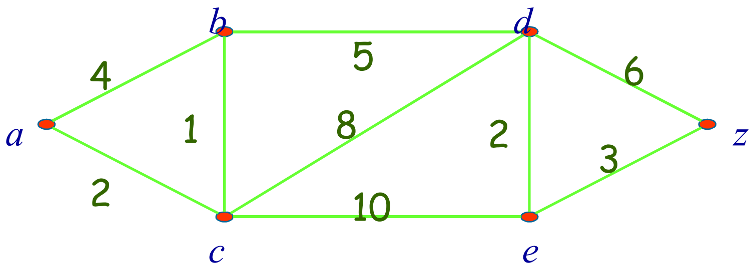

b 3 (a, c)
d 8 (a, c, b)
4
5
6
a
0
1
8
2
z
3
13 (a, c, b, d, e)
2
10
c 2 (a)
e 10 (a, c, b, d)

Step 6

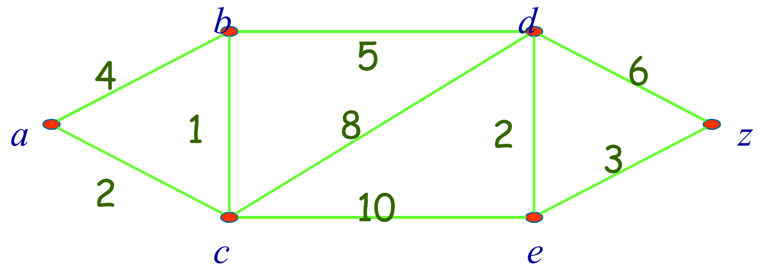

b 3 (a, c)
d 8 (a, c, b)
4
5
6
a
0
1
8
2
z
3
13 (a, c, b, d, e)
2
10
c 2 (a)
e 10 (a, c, b, d)

Remark:

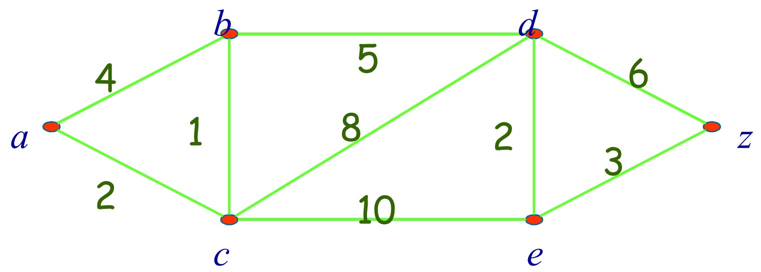
In performing Dijkstra's algorithm it is sometimes more convenient to keep track of labels of vertices in each step using a table.



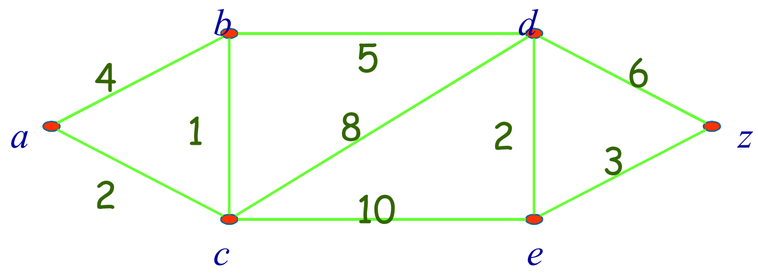
Vertex	S	Link						
a								
b								
c								
d								
e								
z								



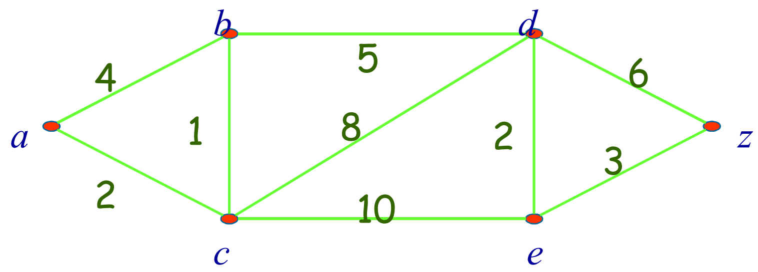
Vertex	S	Link	L_0					
a			0					
b			∞					
c			∞					
d			∞					
e			∞					
z			∞					



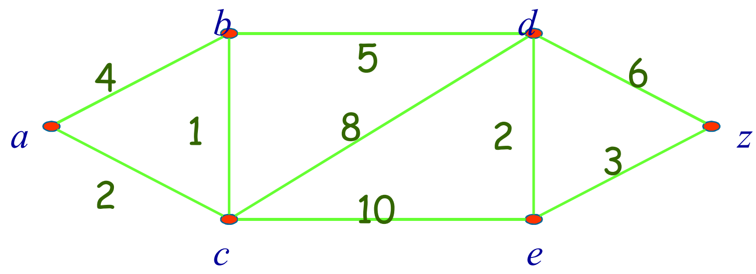
Vertex	S	Link	L_0	L_1				
a	1		0					
b		*a*	∞	4				
c		*a*	∞	2				
d			∞	∞				
e			∞	∞				
z			∞	∞				



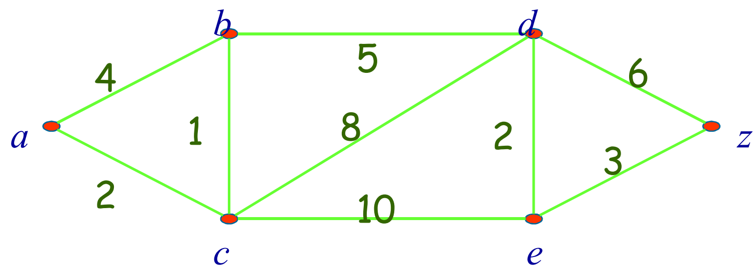
Vertex	S	Link	L_0	L_1	L_2			
a	1		0					
b		*a*→*c*	∞	4	3			
c	1	*a*	∞	2				
d		*a*→*c*	∞	∞	10			
e		*a*→*c*	∞	∞	12			
z			∞	∞	∞			



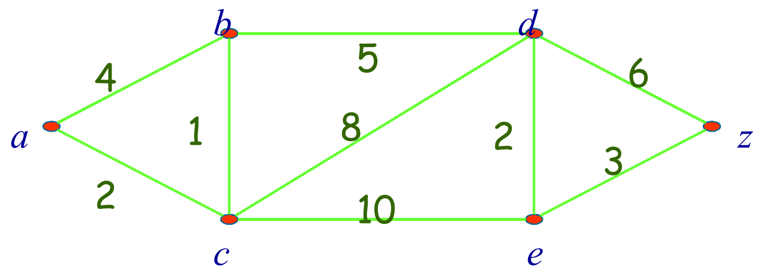
Vertex	S	Link	L_0	L_1	L_2	L_3		
a	1		0					
b	1	*a*→*c*	∞	4	3			
c	1	*a*	∞	2				
d		*a*→*c*→*b*	∞	∞	10	8		
e		*a*→*c*	∞	∞	12	12		
z			∞	∞	∞	∞		



Vertex	S	Link	L_0	L_1	L_2	L_3	L_4	
a	1		0					
b	1	*a*→*c*	∞	4	3			
c	1	*a*	∞	2				
d	1	*a*→*c*→*b*	∞	∞	10	8		
e		*a*→*c*→*b*→*d*	∞	∞	12	12	10	
z		*a*→*c*→*b*→*d*	∞	∞	∞	∞	14	



Vertex	S	Link	L_0	L_1	L_2	L_3	L_4	L_5
a	1		0					
b	1	*a*→*c*	∞	4	3			
c	1	*a*	∞	2				
d	1	*a*→*c*→*b*	∞	∞	10	8		
e	1	*a*→*c*→*b*→*d*	∞	∞	12	12	10	
z		*a*→*c*→*b*→*d* → *e*	∞	∞	∞	∞	14	13



Vertex	S	Link	L_0	L_1	L_2	L_3	L_4	L_5
a	1		0					
b	1	*a*→*c*	∞	4	3			
c	1	*a*	∞	2				
d	1	*a*→*c*→*b*	∞	∞	10	8		
e	1	*a*→*c*→*b*→*d*	∞	∞	12	12	10	
z	1	*a*→*c*→*b*→*d* → *e*	∞	∞	∞	∞	14	13

【 Theorem 1】 Dijkstra's algorithm finds the length of a shortest path between two vertices in a connected simple undirected weighted graph. （只能正权重）

Proof:

Take as the induction hypothesis the following assertion: At the *k*th iteration

I. the label of every vertex *v* in *S* is the length of the shortest path from *a* to this vertex, and

II. the label of every vertex not in *S* is the length of the shortest path from *a* to this vertex that contains only (besides the vertex itself) vertices in *S*.

(1) k=0

$S=\phi$, $L_0(a)=0$, $L_0(v)=\infty$

(2) Assume that the inductive hypothesis holds for the kth iteration.

Let v be the vertex added to S at the (k+1)st iteration so that v is a vertex not in S at the end of the kth iteration with the smallest label.

- (I) holds at the end of the (k+1)st iteration
  - ✓ The vertices in S before the (k+1)st iteration are labeled with the length of the shortest path from a.
  - ✓ v must be labeled with the length of the shortest path to it from a.

    If this were not the case, at the end of the kth iteration there would be a path of length less than $L_k(v)$ containing a vertex not in S.

Let u be *the first* vertex not in S in such a path. There is a path with length less than $L_k(v)$ from a to u containing only vertices of S. This contradicts the choice of v.

- (II) is true.

Let u be a vertex not in S after k+1 iteration.

A shortest path from a to u containing only elements of S either contains v or it does not.

- ➢ If it does not contain v, then by the inductive hypothesis its length is $L_k(u)$.
- ➢ If it does contain v, then it must be made up of a path from a to v of the shortest possible length containing elements of S other than v, followed by the edge from v to u. In this case its length would be $L_k(v)+w(v,u)$.

【 Theorem 2】 Dijkstra's algorithm uses $O(n^2)$ operations (additions and comparisons) to find the length of the shortest path between two vertices in a connected simple undirected weighted graph.

Analysis:

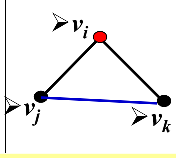
- Use no more than n-1 iteration
- Each iteration,

  using no more than n-1 comparisons to determine the vertex not in S_k with the smallest label

  no more than 2(n-1) operations are used to update no more than n-1 labels

Floyd's Algorithm (find the distance $d(a, b)$ $\forall a, b$)

Procedure *Floyd*(G: weighted simple graph)
{G has vertices $v_1, \ldots, v_n$ and weights $w(v_i, v_j)$ with
  $w(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ is not an edge}
for i := 1 **to** n
  for j := 1 **to** n
    $d(v_i, v_j) := w(v_i, v_j)$
for i := 1 **to** n
  for j := 1 **to** n
    for k := 1 **to** n
      if $d(v_j, v_i) + d(v_i, v_k) < d(v_j, v_k)$
        then $d(v_j, v_k) := d(v_j, v_i) + d(v_i, v_k)$
{$d(v_i, v_j)$ is the length of a shortest path between v_i and v_j}



➢对每个v_i，检查任何一对点目前的距离会不会因为改走 v_i 而变小。

➢This algorithm cannot be used to construct shortest paths.

可以有负权边但不能有负权回路

3. The Traveling Salesman Problem

❈ **The traveling salesman problem**

The traveling salesman problem asks for the circuit of minimum total weight in a weighted, complete, undirected graph that visits each vertex exactly once and returns to its starting point.

The equivalent problem:

Find a Hamilton circuit with minimum total weight in the complete graph.

❈ **Solution of the traveling salesman problem**

(1) A straightforward method

Examine all possible Hamilton circuits and select one of minimum total length.

The number of Hamilton circuits: $(n-1)!$

Since a Hamilton circuits can be traveled in reverse order, we need only examine $(n-1)!/2$ **circuits to find the answer.**

The time complexity: $n!$

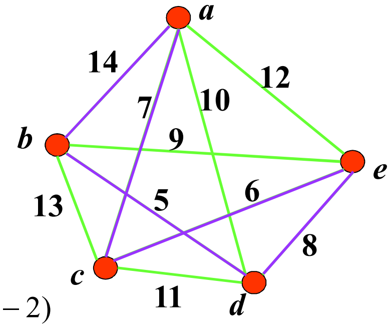
(2) Approximation algorithm

These are algorithms that do not necessary produce the exact solution to the problem but instead are guaranteed to produce a solution that is close to an exact solution.

For example,

The length of this path: 40

The exact solution (the length is 37):*a,c,e,b,d,a*



The time complexity:

$$1+2+3+\cdots+(n-2)=\frac{1}{2}(n-1)(n-2)$$

Homework:

Sec. 10.6 3, 17a), 26