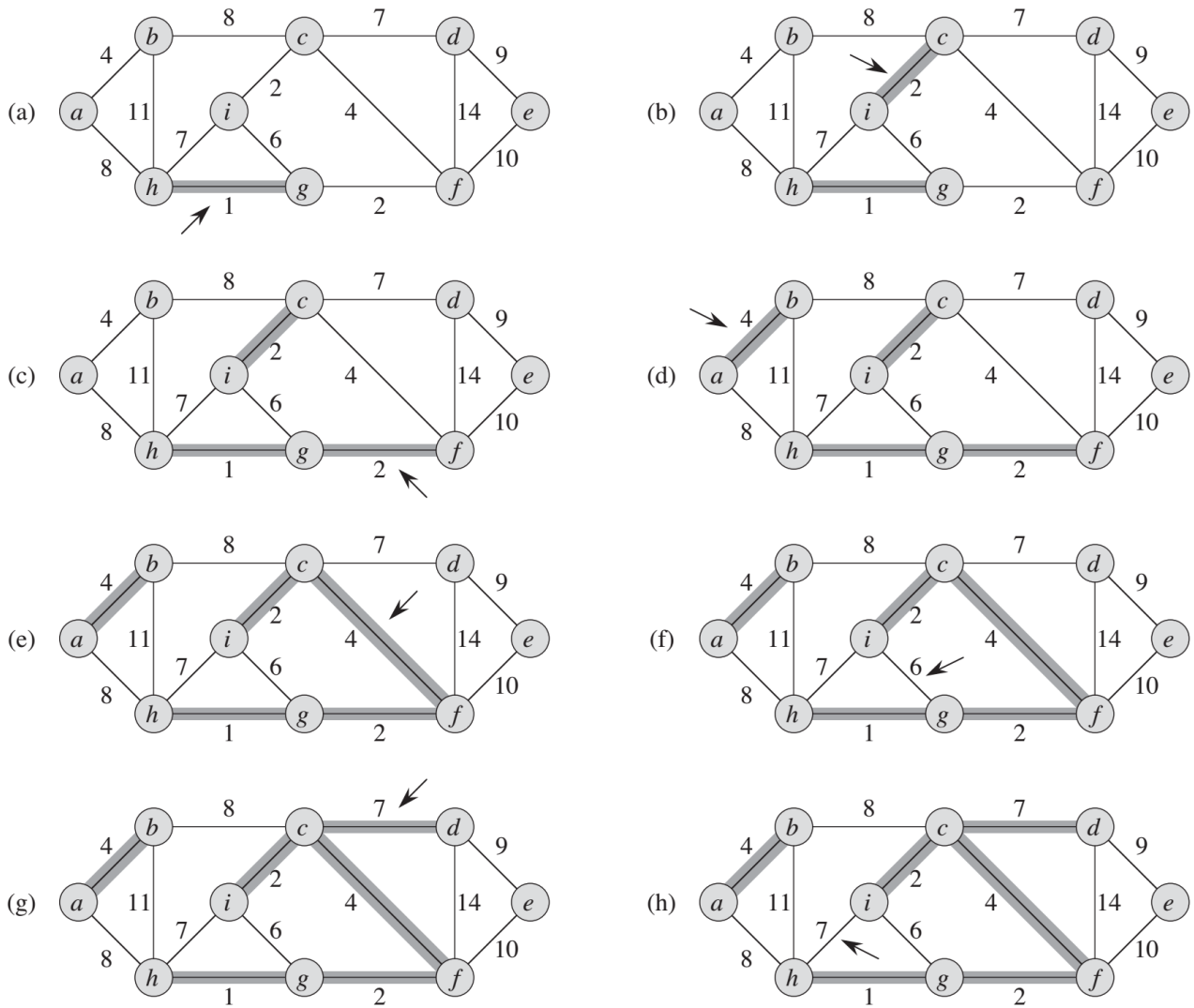
بسم اللّه الرّحمن الرّحیم

دانشگاه صنعتی اصفهان ـ دانشکدهٔ مهندسی برق و کامپیوتر
(نیم‌سال تحصیلی ۴۰۰۱)

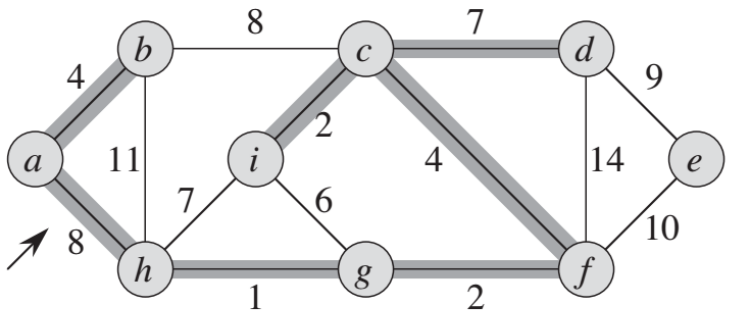
طراحی الگوریتم‌ها

حسین فلسفین

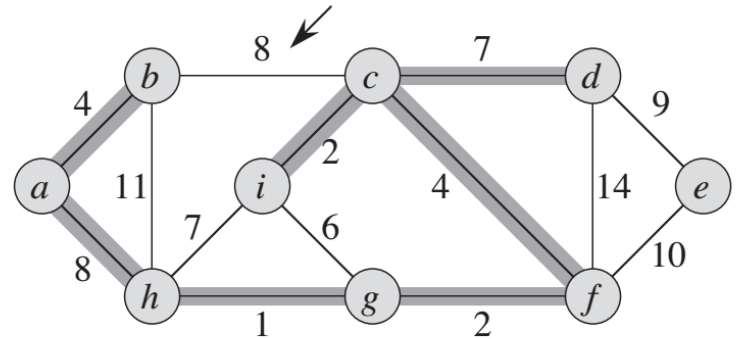
یک مثال دیگر از بکارگیری الگوریتم کراسکال:



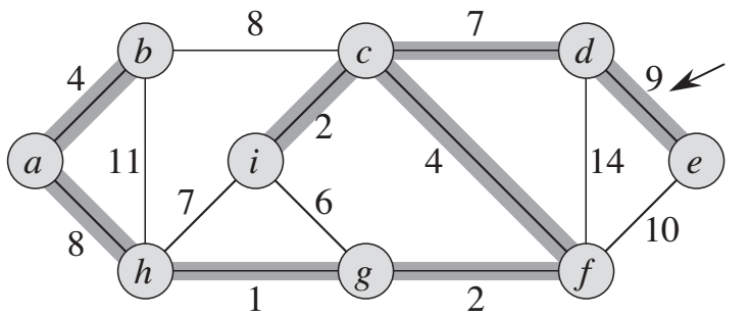
(i)


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
7
6
14
9
10
1
2

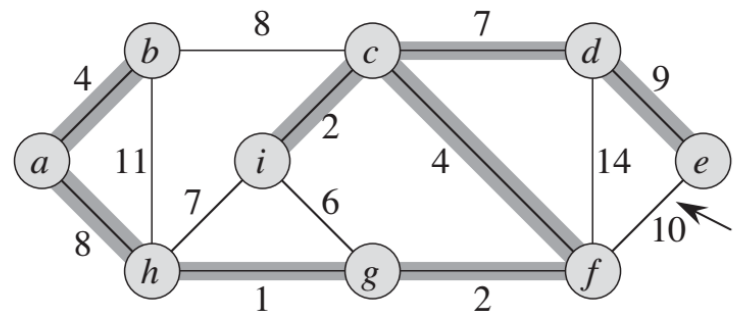
(j)


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
7
6
14
9
10
1
2

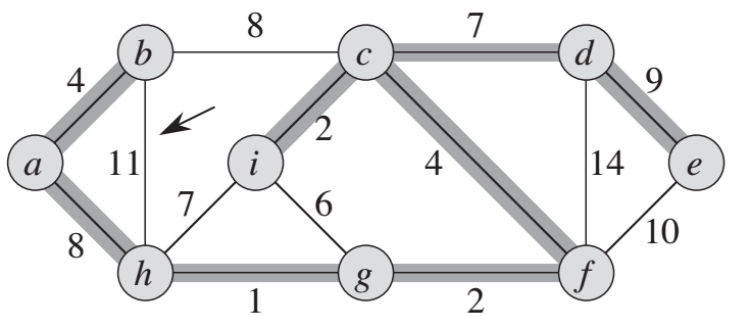
(k)


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
7
6
14
9
10
1
2

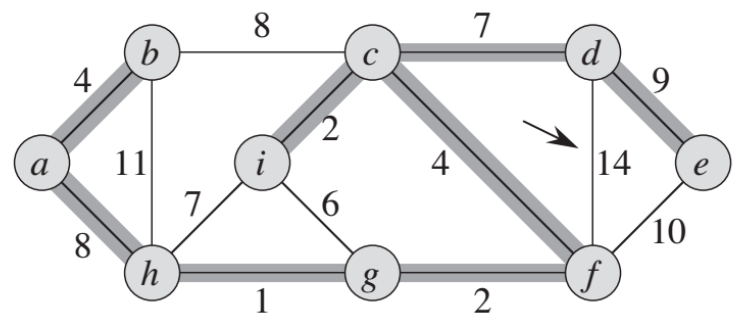
(l)


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
7
6
14
9
10
1
2

(m)


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
7
6
14
9
10
1
2

(n)


a
b
c
d
e
f
g
h
i
4
8
11
8
7
2
4
7
6
14
9
10
1
2

نحوهٔ ادغام حین اجرای الگوریتم کراسکال در مثال فوق:

1. $\{\{a\},\{b\},\{c\},\{d\},\{e\},\{f\},\{g\},\{h\},\{i\}\}$
2. $\{\{a\},\{b\},\{c\},\{d\},\{e\},\{f\},\{g,h\},\{i\}\}$
3. $\{\{a\},\{b\},\{c,i\},\{d\},\{e\},\{f\},\{g,h\}\}$
4. $\{\{a\},\{b\},\{c,i\},\{d\},\{e\},\{f,g,h\}\}$
5. $\{\{a,b\},\{c,i\},\{d\},\{e\},\{f,g,h\}\}$
6. $\{\{a,b\},\{d\},\{e\},\{c,f,g,h,i\}\}$
7. $\{\{a,b\},\{e\},\{c,d,f,g,h,i\}\}$
8. $\{\{e\},\{a,b,c,d,f,g,h,i\}\}$
9. $\{\{a,b,c,d,e,f,g,h,i\}\}$

$$G = (V, E),\ |V| = n,\ |E| = m$$

G is connected, therefore we have $n - 1 \leq m \leq \binom{n}{2} = \frac{n(n-1)}{2}$.

☞ *The time complexity of an algorithm sometimes depends on the data structure used to implement it.*

☞ *The running time of Kruskal's algorithm for a graph $G = (V, E)$ depends on how we implement the disjoint-set data structure. We assume that we use the disjoint-set-forest implementation of [CLRS, Section 21.3] with the union-by-rank and path-compression heuristics, since it is the asymptotically fastest implementation known. The running time of Kruskal's algorithm is $O(m \log(m))$.*

☞ *Using the Fibonacci heap, Fredman and Tarjan (1987) developed the fastest implementation of Prim's algorithm. Their implementation is $\Theta(m + n \log(n))$. For a sparse graph, this is $\Theta(n \log(n))$, and for a dense graph it is $\Theta(n^2)$.*

ما تاکنون، مسئلهٔ یافتن درخت فراگیر کمینه ***(MST)*** را مطرح، و دو الگوریتم معروف مبتنی بر راهبرد حریصانه برای حل این مسئله، یعنی الگوریتم پریم و الگوریتم کراسکال، را معرفی کردیم. علی‌رغم ذات حریصانه، هردو الگوریتم قادر به یافتن جواب بهینه هستند. (این باید رسماً اثبات شود. ما از بیان اثبات صرف‌نظر کرده‌ایم. اثبات هم در کتاب نیپلیتن و هم در کتاب ***CLRS*** مطرح شده است.)

حال در این جلسه یک الگوریتم تقریبی معروف برای مسئلهٔ ***TSP*** که از دو الگوریتم قبلی، یعنی الگوریتم نزدیک‌ترین همسایه و الگوریتم ***Multifrag-ment***، عملکرد مطلوب‌تری دارد را معرفی خواهیم کرد. این الگوریتم از یک درخت فراگیر کمینه برای ساخت تور بهره می‌گیرد.

Minimum-Spanning-Tree-Based Algorithms

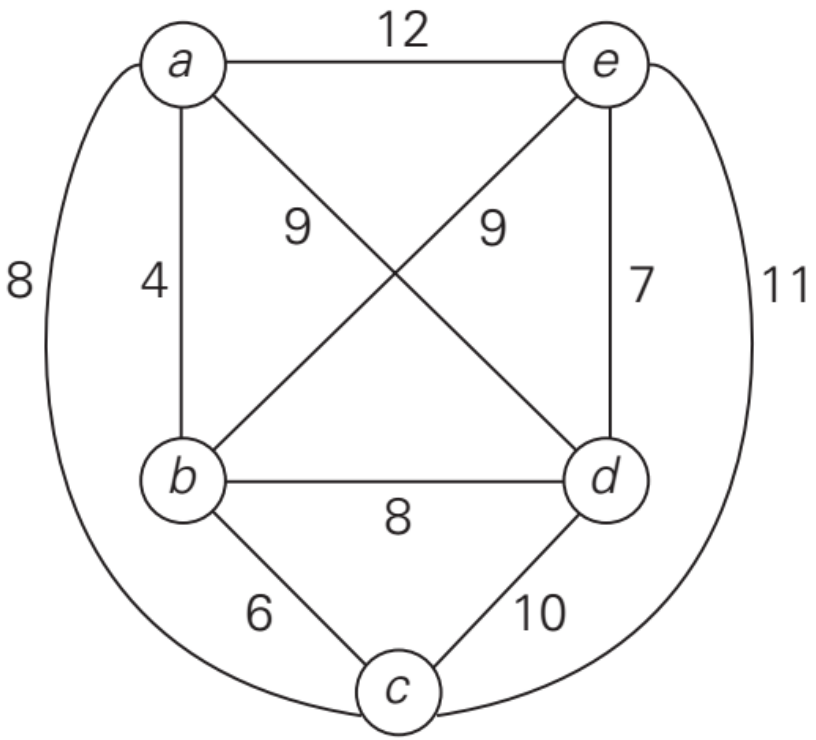
There are approximation algorithms for the traveling salesman problem that exploit a connection between Hamiltonian circuits and spanning trees of the same graph. Since removing an edge from a Hamiltonian circuit yields a spanning tree, we can expect that the structure of a minimum spanning tree provides a good basis for constructing a shortest tour approximation. Here is an algorithm that implements this idea in a rather straightforward fashion.

Twice-around-the-tree algorithm (double-tree algorithm)

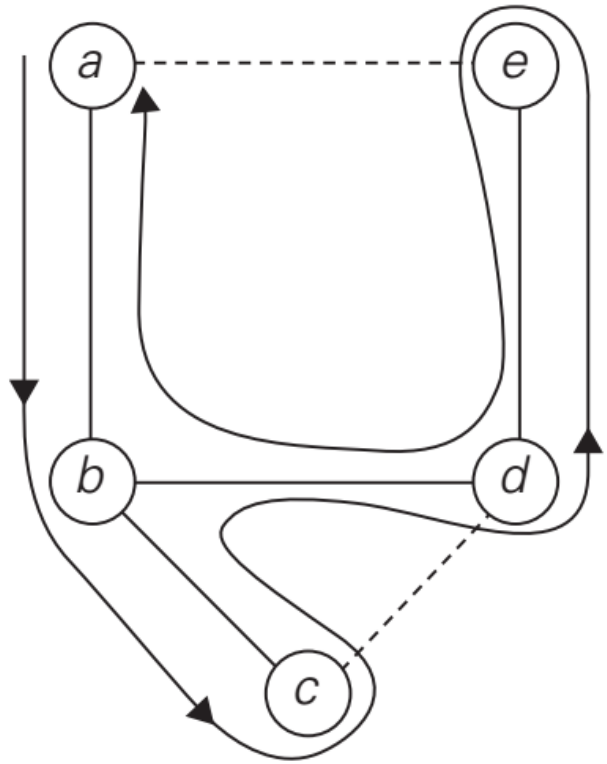
☞ *Step 1 Construct a minimum spanning tree of the graph corresponding to a given instance of the traveling salesman problem.*

☞ *Step 2 Starting at an arbitrary vertex, perform a walk around the minimum spanning tree recording all the vertices passed by. (This can be done by a DFS traversal.)*

☞ *Step 3 Scan the vertex list obtained in Step 2 and eliminate from it all repeated occurrences of the same vertex except the starting one at the end of the list. (This step is equivalent to making shortcuts in the walk.) The vertices remaining on the list will form a Hamiltonian circuit, which is the output of the algorithm.*



(a)

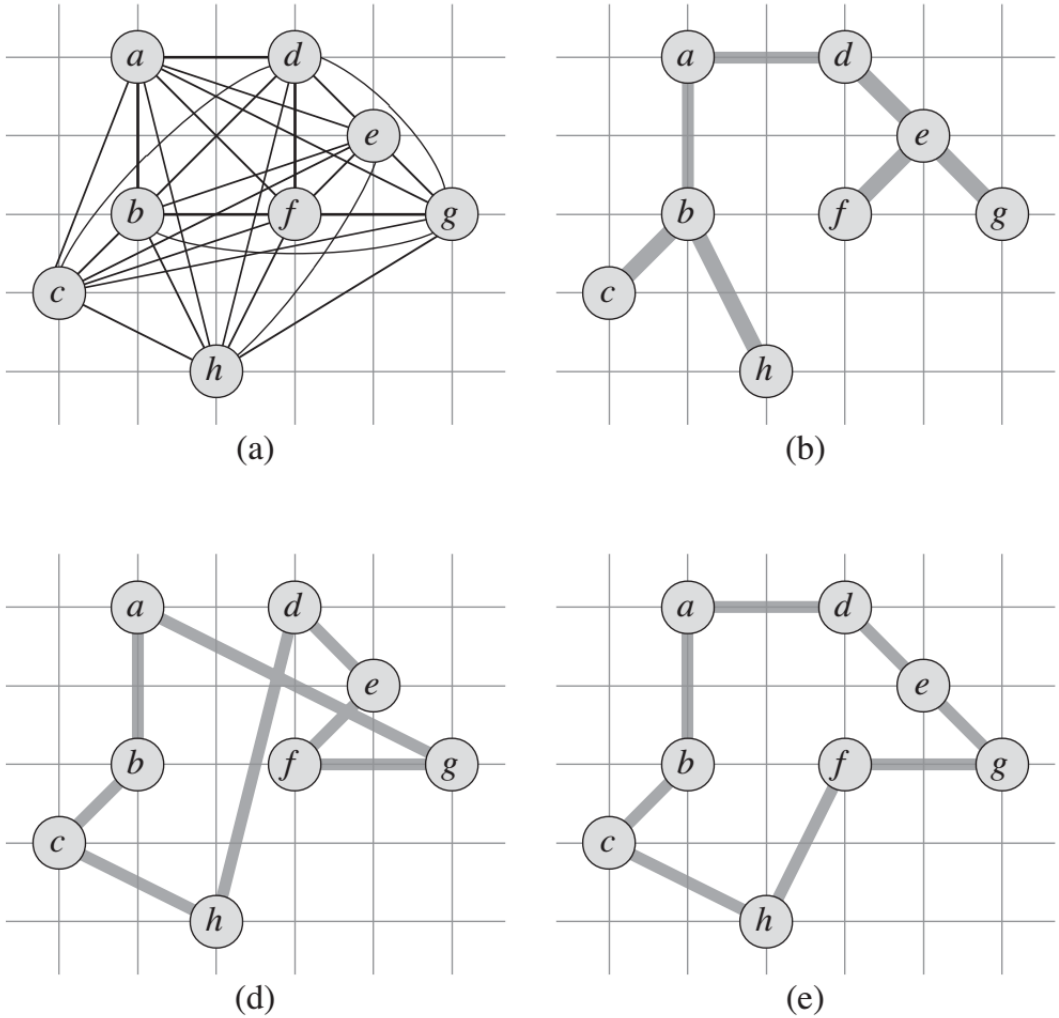


(b)

$$a, b, c, b, d, e, d, b, a \text{ ☞ } a, b, c, d, e, a$$

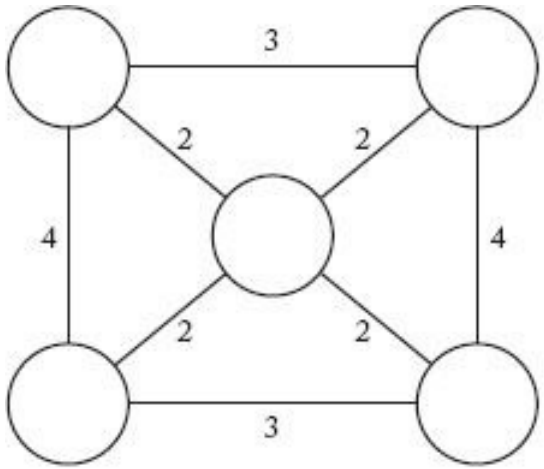
Length: 39 (The tour is not optimal.)

For the twice-around-the-tree algorithm, we can at least estimate the accuracy ratio $f(s_a)/f(s^*)$, provided the graph is Euclidean.

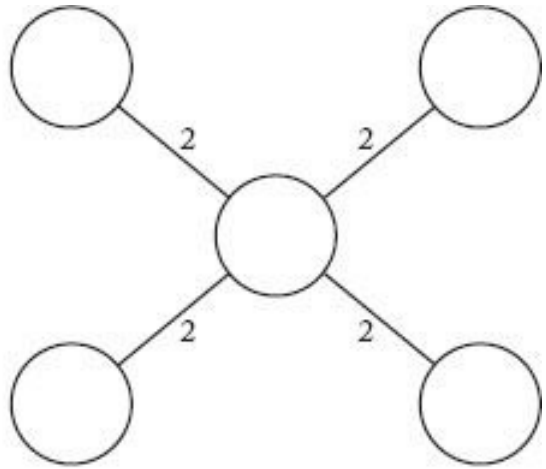


$a, b, c, b, h, b, a, d, e, f, e, g, e, d, a$ ☞ a, b, c, h, d, e, f, g

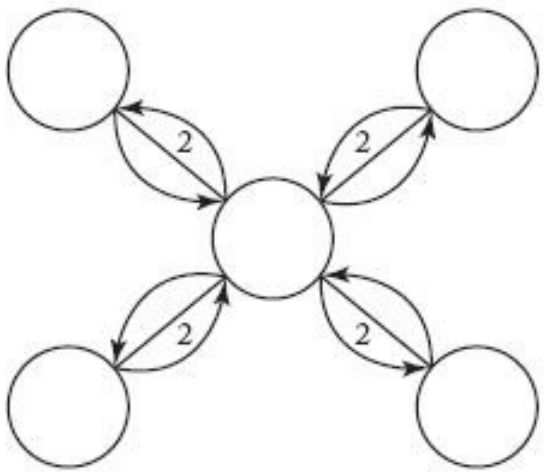
$(f(s_a) = 19.074$ ***and*** $f(s^*) = 14.715)$



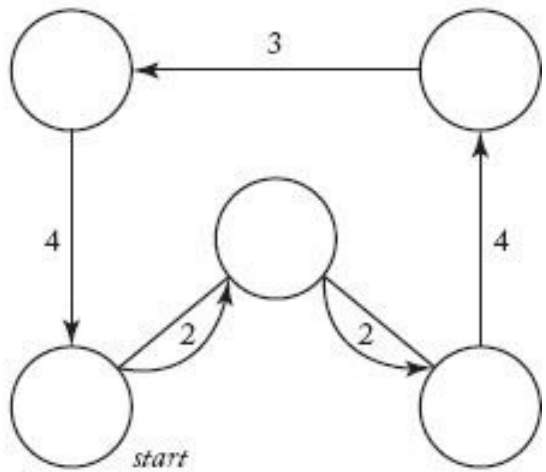
(a) A weighted, undirected graph



(b) A minimum spanning tree



(c) Twice around the tree



(d) A circuit with shortcuts

توجه کنید که بنا با قضیه ای که قبلاً مطرح کردیم، نمی‌توان یک الگوریتم c-***approximation*** برای مسئلهٔ ***TSP*** ارائه کرد. اما اگر خود را محدود به نمونه‌های اقلیدسی کنیم اوضاع فرق می‌کند.

(قضیهٔ زیر و اثباتش مهم هستند.)

THEOREM *The twice-around-the-tree algorithm is a 2-approximation algorithm for the traveling salesman problem with Euclidean distances.*

PROOF *Obviously, the twice-around-the-tree algorithm is polynomial-time if we use a reasonable algorithm such as Prim's or Kruskal's in Step 1. We need to show that for any Euclidean instance of the traveling salesman problem, the length of a tour s_a obtained by the twice-around-the-tree algorithm is at most twice the length of the optimal tour s^*, i.e.,*

$$f(s_a) \leq 2f(s^*).$$

Since removing any edge from s^* yields a spanning tree T of weight $w(T)$, which must be greater than or equal to the weight of the graph's minimum spanning tree $w(T^*)$, we get the inequality

$$f(s^*) > w(T) \geq w(T^*).$$

This inequality implies that

$$2f(s^*) > 2w(T^*) =$$

the length of the walk obtained in Step 2 of the algorithm.

The possible shortcuts outlined in Step 3 of the algorithm to obtain s_a *cannot increase the total length of the walk in a Euclidean graph, i.e.,*

the length of the walk obtained in Step 2 $\geq$ *the length of the tour* s_a.

REMARK Our walk is generally not a tour, since it visits some vertices more than once. By the triangle inequality, however, we can delete a visit to any vertex from the walk and the cost does not increase. (If we delete a vertex y *from between visits to* x *and* z*, the resulting ordering specifies going directly from* x *to* z*.) By repeatedly applying this operation, we can remove from the tour all but the first visit to each vertex.*

Combining the last two inequalities, we get the inequality

$$2f(s^*) > f(s_a),$$

which is, in fact, a slightly stronger assertion than the one we needed to prove. □

در اثبات فوق، در چه مواضعی از اقلیدسی بودن استفاده شده است؟

این الگوریتم مفصلاً در زیربخش ۱.۵.۹ از کتاب نیپلیتن، بخش ۲.۳۵ از کتاب ***CLRS***، و بخش ۳.۱۲ از کتاب ***Levitin*** تشریح شده است.

The general traveling-salesman problem

قضیه زیر و اثباتش خیلی مهم هستند. (بخش ۳.۱۲ از کتاب **Levitin** و بخش قضیهٔ ۳.۳۵ از **CLRS** را ببینید.)

Theorem ***If $P \neq NP$, there exists no c-approximation algorithm for the traveling salesman problem, i.e., there exists no polynomial-time approximation algorithm for this problem so that for all instances***

$$f(s_a)/f(s^*) \leq c$$

for some constant c.

قبل از بیان اثبات، باید مسئلهٔ ***Hamiltonian circuit problem*** ***(Hamiltonian-cycle problem)*** را معرفی کنیم.

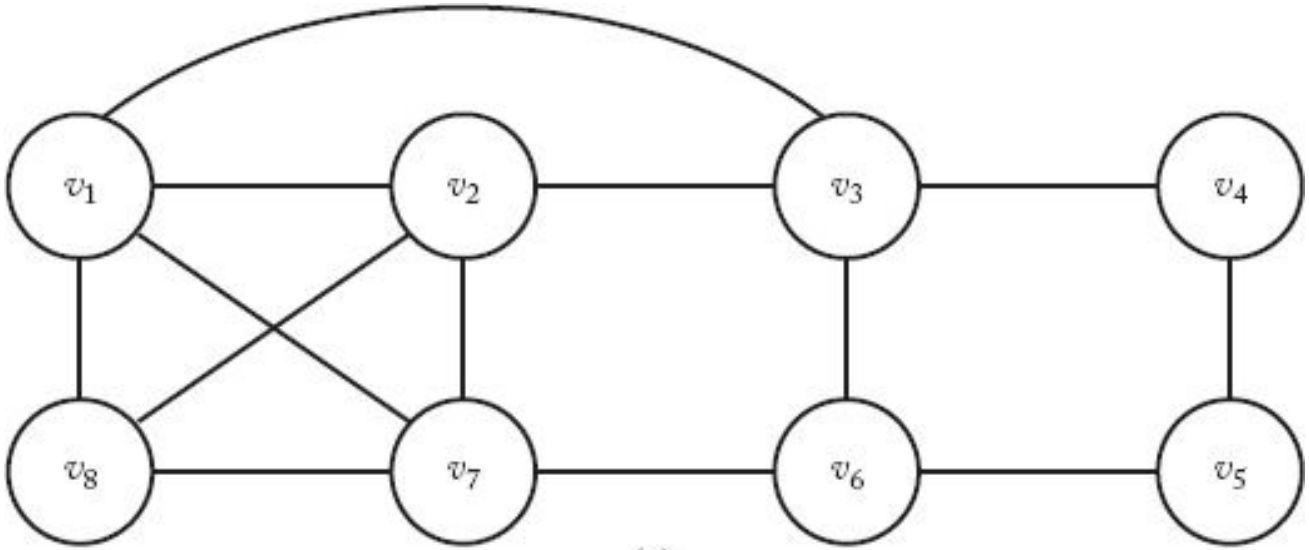
Hamiltonian cycles

☞ *The problem of finding a hamiltonian cycle in an undirected graph has been studied for over a hundred years. Formally, a hamiltonian cycle of an undirected graph $G = (V, E)$ is a simple cycle that contains each vertex in V. A graph that contains a hamiltonian cycle is said to be hamiltonian; otherwise, it is nonhamiltonian.*

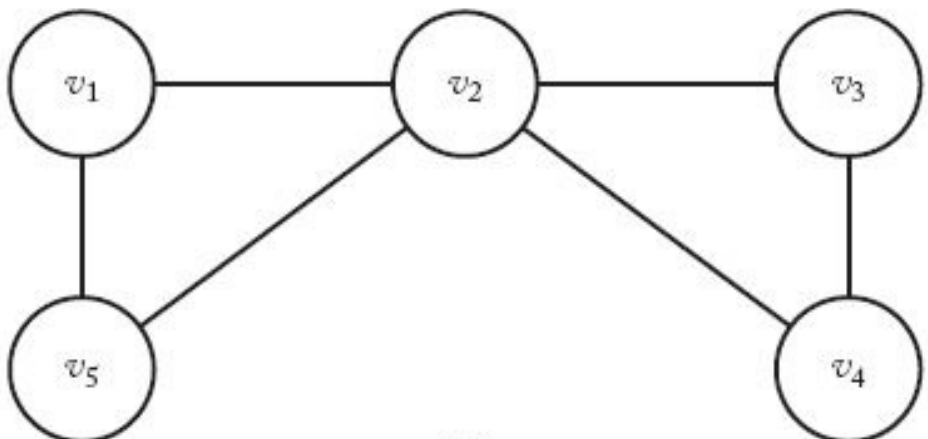
☞ *hamiltonian-cycle problem: "Does a graph G have a hamiltonian cycle?"*

☞ *Given a connected, undirected graph, a Hamiltonian Circuit (also called a tour) is a path that starts at a given vertex, visits each vertex in the graph exactly once, and ends at the starting vertex.*

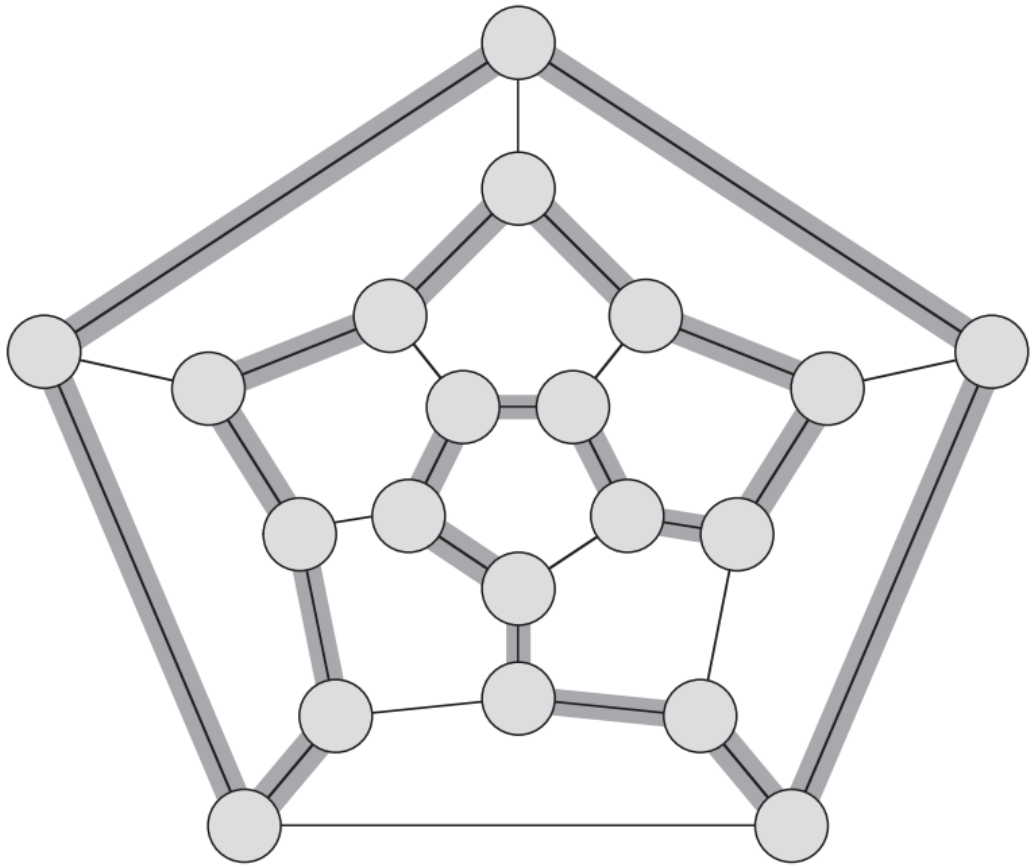
☞ *The Hamiltonian Circuits problem determines the Hamiltonian Circuits in a connected, undirected graph.*



(a)



(b)



The hamiltonian cycle problem is NP-complete.

(توجه کنید که این مسئله یک مسئلهٔ تصمیم گیری است نه بهینه سازی.)

حال به سراغ اثبات میرویم:

PROOF *By way of contradiction, suppose that such an approximation algorithm A and a constant c exist. (Without loss of generality, we can assume that c is a positive integer.) We will show that this algorithm could then be used for solving the Hamiltonian circuit problem in polynomial time. We reduce the Hamiltonian circuit problem to the traveling salesman problem. Let G be an arbitrary graph with n vertices. We map G to a complete weighted graph G' by assigning weight 1 to each edge in G and adding an edge of weight $cn+1$ between each pair of vertices not adjacent in G. If G has a Hamiltonian circuit, its length in G' is n; hence, it is the exact solution s^* to the traveling salesman problem for G'. Note that if s_a is an approximate solution obtained for G' by algorithm A, then $f(s_a) \leq cn$ by the assumption. If G does not have a Hamiltonian circuit in G, the shortest tour in G' will contain at least one edge of weight $cn+1$, and hence*

$$f(s_a) \geq f(s^*) > cn.$$

Taking into account the two derived inequalities, we could solve the Hamiltonian circuit problem for graph G in polynomial time by mapping G to G', applying algorithm A to get tour s_a in G', and comparing its length with cn. Since the Hamiltonian circuit problem is NP-complete, we have a contradiction unless $P = NP$. $\square$