# Hamilton Circuit

Topics in Contemporary Mathematics

MA 103

Summer II, 2013

# Hamilton Paths and Hamilton Circuits

In **Euler paths and Euler circuits**, the game was to find paths or circuits that include **every edge** of the graph once (and only once).

In **Hamilton paths and Hamilton circuits,** the game is to find paths and circuits that include **every vertex** of the graph once and only once.

# HAMILTON PATHS & CIRCUITS

- A *Hamilton path* in a graph is a path that includes each vertex of the graph once and only once.
- A *Hamilton circuit* is a circuit that includes each vertex of the graph once and only once. (At the end, of course, the circuit must return to the starting vertex.)

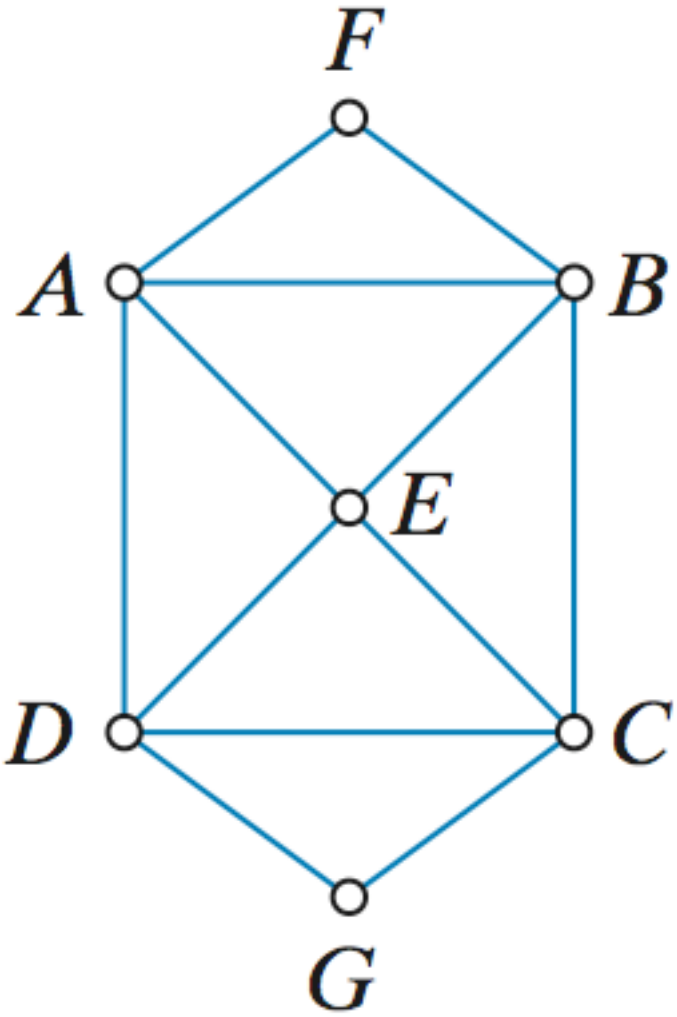
# Example 1 Hamilton versus Euler

The figure shows a graph that

(1) *has Euler circuits (the vertices are all even) and*

(2) *has Hamilton circuits.*

One such *Hamilton circuit is A, F, B, C, G, D, E, A* – there are plenty more.



# Example 1 Hamilton versus Euler

Note that *if a graph has a Hamilton circuit, then it automatically has a Hamilton path*–(the Hamilton circuit can always be truncated into a Hamilton path by dropping the last vertex of the circuit.)

*For example, the Hamilton circuit A, F, B, C, G, D, E, A can be truncated into the Hamilton path A, F, B, C, G, D, E.*

Contrast this with the mutually exclusive relationship between Euler circuits and paths: *If a graph has an Euler circuit it cannot have an Euler path and vice versa.*

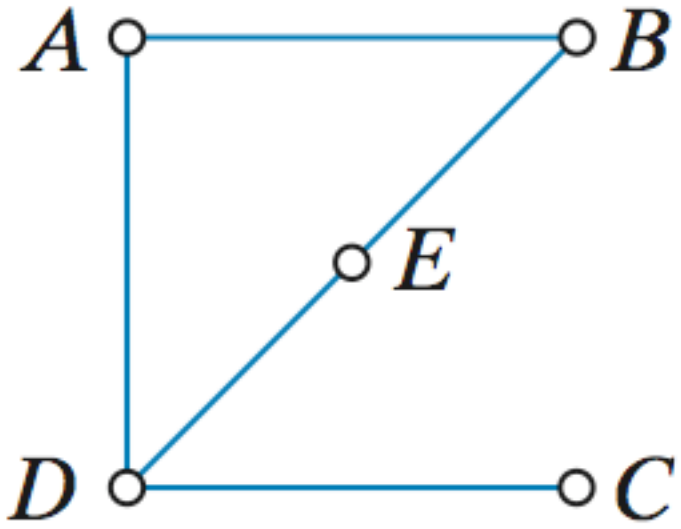
# Example 1 Hamilton versus Euler

This figure shows a graph that

*(1)has no Euler circuits but does have Euler paths (for example C, D, E, B, A, D) and*

*(2)has no Hamilton circuits (sooner or later you have to go to C, and then you are stuck) but does have Hamilton paths (for example, A, B, E, D, C).*

*This illustrates that a graph can have a Hamilton path but no Hamilton circuit!*



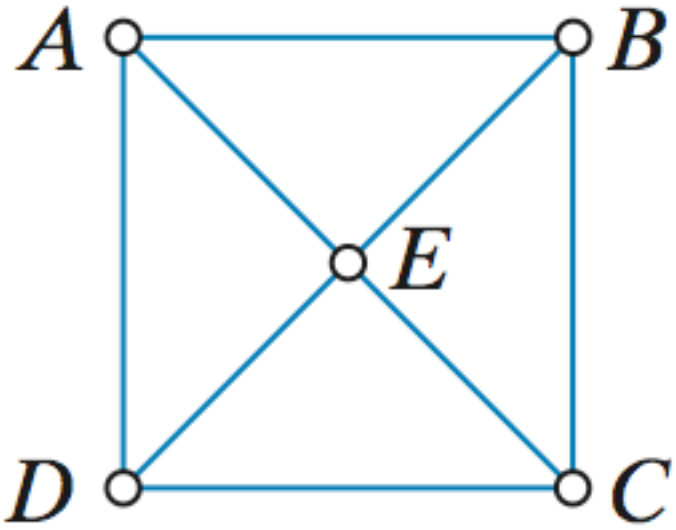
# Example 1 Hamilton versus Euler

This figure shows a graph that

*(1)has neither Euler circuits nor paths (it has four odd vertices) and*

*(2) has Hamilton circuits (for example A, B, C, D, E, A – there are plenty more)*

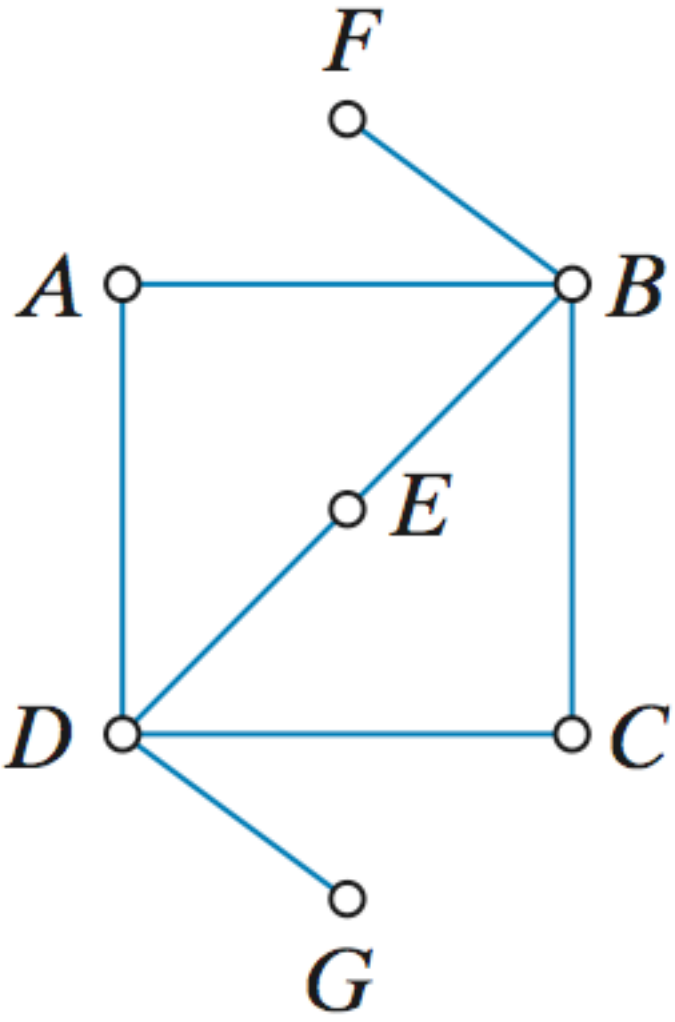
and consequently has *Hamilton paths (for example, A, B, C, D, E).*



# Example 1 Hamilton versus Euler

This figure shows a graph that *(1) has no Euler circuits but has Euler paths (F and G are the two odd vertices) and*

*(2) has neither Hamilton circuits nor Hamilton paths.*

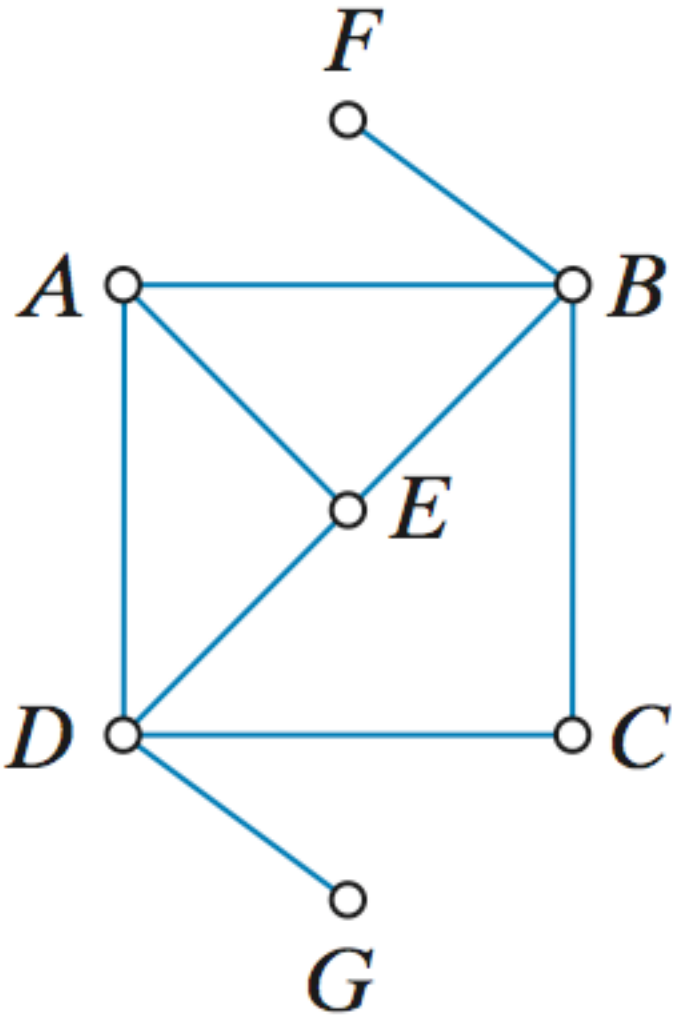


# Example 1 Hamilton versus Euler

This figure shows a graph that

*(1) has neither Euler circuits nor Euler paths (too many odd vertices) and*

*(2) has neither Hamilton circuits nor Hamilton paths.*



# Euler versus Hamilton

The lesson of Example 1 is that the existence of an Euler path or circuit in a graph tells us nothing about the existence of a Hamilton path or circuit in that graph.

# Euler versus Hamilton

*Unlike Euler circuit and path, there exist no "Hamilton circuit and path theorems" for determining if a graph has a Hamilton circuit, a Hamilton path, or neither.*

Determining when a given graph does or does not have a Hamilton circuit or path can be very easy, but it also can be very hard–it all depends on the graph.

# How Many Hamilton Circuits?

Sometimes the question, *Does the graph have a Hamilton circuit?* has an obvious *yes* answer, and the more relevant question turns out to be, *How many different Hamilton circuits does it have?*

In this section we will answer this question for an important **family of graphs called complete graphs**.

# Complete Graph

A graph with **N** vertices in which every pair of distinct vertices is joined by an edge is called *a complete graph on N vertices* and is denoted by the symbol $K_N$.

# How Many Hamilton Circuits?

One of the *key properties of* $K_N$ *is that every vertex has degree* $N - 1$.

This implies that the sum of the degrees of all the vertices is $N(N - 1)$, and it follows from Euler's sum of degrees theorem that the number of edges in $K_N$ is $N(N - 1)/2$.

For a graph with $N$ vertices and no multiple edges or loops, $N(N - 1)/2$ is the maximum number of edges possible, and this maximum can only occur when the graph is $K_N$.

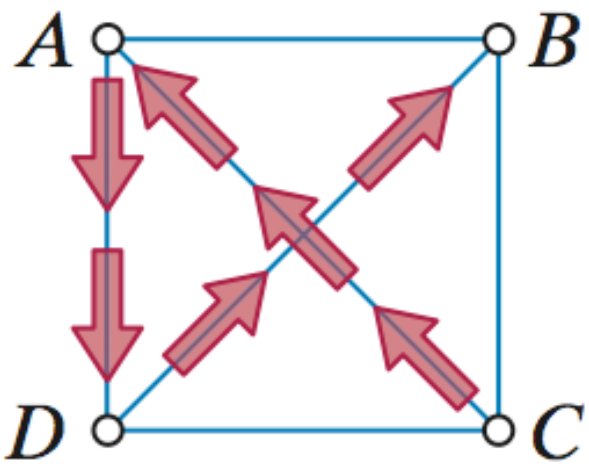
## NUMBER OF EDGES IN $K_N$

- $K_N$ has $N(N - 1)/2$ edges.
- Of all graphs with $N$ vertices and no multiple edges or loops, $K_N$ has the most edges.

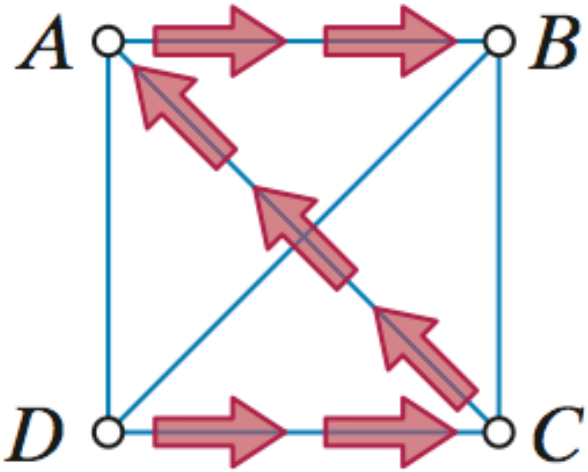
Because $K_N$ *has a complete set of edges* (every vertex is connected to every other vertex), *it also has a complete set of Hamilton circuits* – you can travel the vertices in any sequence you choose and you will not get stuck.

# Example 2 Hamilton Circuits in $K_4$

If we travel the four vertices of $K_4$ in an arbitrary order, we get a Hamilton path. *For example, C, A, D, B is a Hamilton path [Fig. (a)]; D, C, A, B is another one [Fig. (b)]; and so on.*



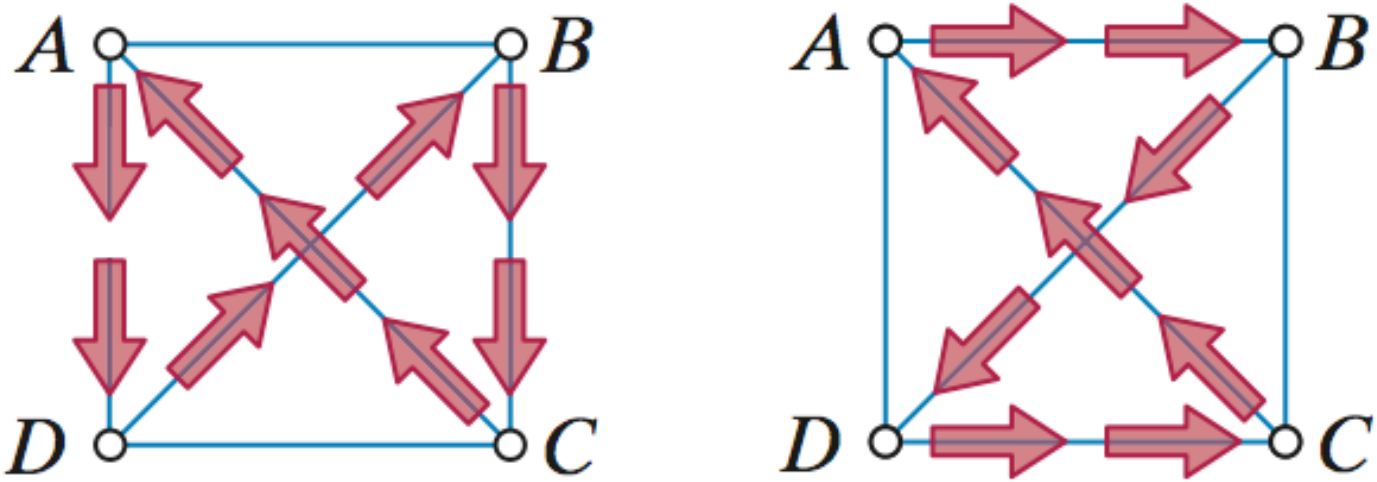

(a)




(b)

# Example 2 Hamilton Circuits in $K_4$

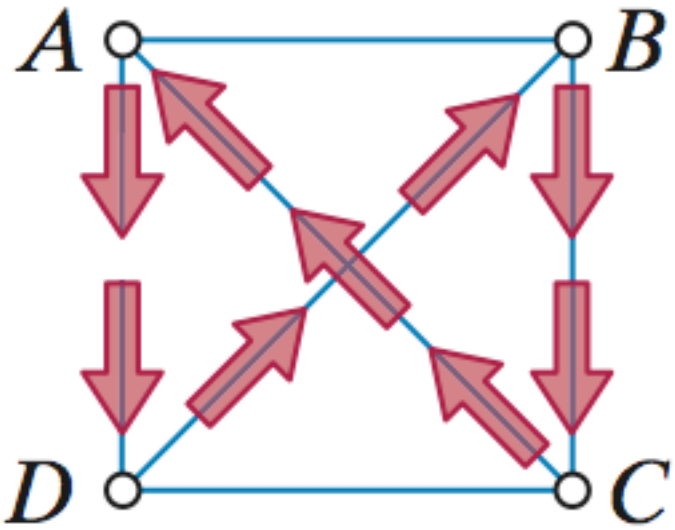
Each of these Hamilton paths can be closed into a Hamilton circuit–*the path C, A, D, B begets the circuit C, A, D, B, C [Fig. (c)]; the path D, C, A, B begets the circuit D, C, A, B, D [Fig. (d)]; and so on.*




(c) (d)

# Example 2 Hamilton Circuits in $K_4$

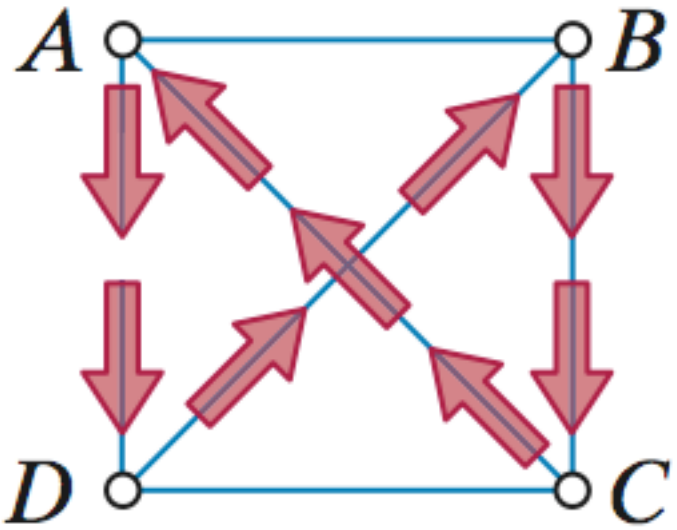
It looks like we have an abundance of Hamilton circuits, but it is important to remember that the same Hamilton circuit can be written in many ways. *For example, C, A, D, B, C is the same circuit as A, D, B, C, A* –the figure describes either one– *the only difference is that in the first case we used C as the reference point; in the second case we used A as the reference point.*



# Example 2 Hamilton Circuits in $K_4$

There are two additional sequences that describe this same Hamilton circuit: *D, B, C, A, D (with reference point D) and B, C, A, D, B (with reference point B).* Taking all this into account, there are six different Hamilton

circuits in $K_4$, as shown in the Table on the next slide (the table also shows the four different ways each circuit can be written).



# Example 2 Hamilton Circuits in $K_4$

The Six Hamilton Circuits in $K_4$

| | Reference point is *A* | Reference point is *B* | Reference point is *C* | Reference point is *D* |
|---|---|---|---|---|
| 1 | $A, B, C, D, A$ | $B, C, D, A, B$ | $C, D, A, B, C$ | $D, A, B, C, D$ |
| 2 | $A, B, D, C, A$ | $B, D, C, A, B$ | $C, A, B, D, C$ | $D, C, A, B, D$ |
| 3 | $A, C, B, D, A$ | $B, D, A, C, B$ | $C, B, D, A, C$ | $D, A, C, B, D$ |
| 4 | $A, C, D, B, A$ | $B, A, C, D, B$ | $C, D, B, A, C$ | $D, B, A, C, D$ |
| 5 | $A, D, B, C, A$ | $B, C, A, D, B$ | $C, A, D, B, C$ | $D, B, C, A, D$ |
| 6 | $A, D, C, B, A$ | $B, A, D, C, B$ | $C, B, A, D, C$ | $D, C, B, A, D$ |

# Example 2 Hamilton Circuits in $K_5$

Let's try to list all the Hamilton circuits in $K_5$.

For simplicity, we will write each circuit just once, using a common reference point – say *A*. (As long as we are consistent, it doesn't really matter which reference point we pick.)

Each of the Hamilton circuits will be described by a sequence that starts and ends with *A*, with the letters *B*, *C*, *D*, and *E* sandwiched in between in some order.

There are $4 \times 3 \times 2 \times 1 = 24$ different ways to shuffle the letters *B*, *C*, *D*, and *E*, each producing a different Hamilton circuit.

# Example 2 Hamilton Circuits in $K_5$

The complete list of the 24 Hamilton circuits in $K_5$ is shown in the table on the next slide. The table is laid out so that each of the circuits in the table is directly opposite its mirror-image circuit (the circuit with vertices listed in reverse order).

Although they are close relatives, a circuit and its mirror image are not considered the same circuit.

# Example 2 Hamilton Circuits in $K_5$

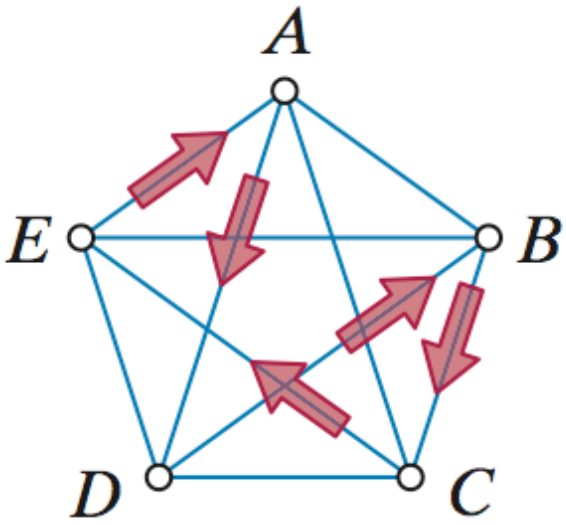
The complete list of the 24 Hamilton circuits in $K_5$.

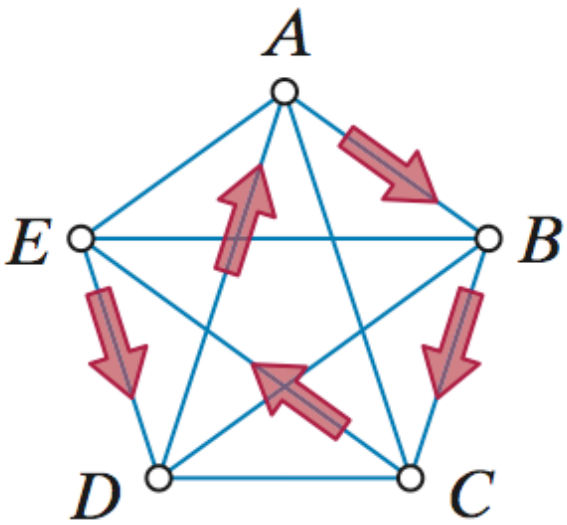
**The 24 Hamilton Circuits in $K_5$**

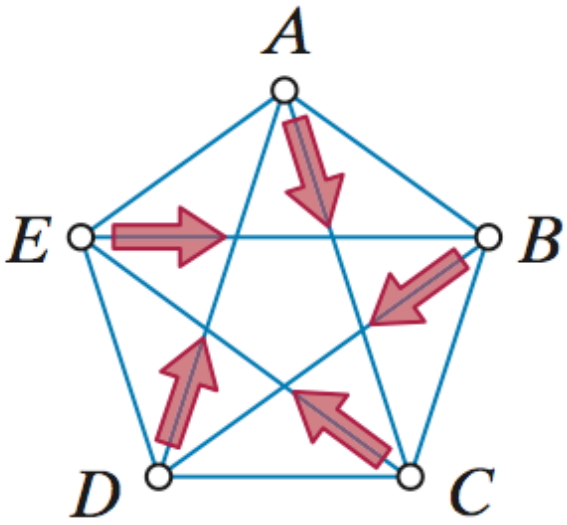
| | | | |
|---|---|---|---|
| 1 | $A, B, C, D, E, A$ | 13 | $A, E, D, C, B, A$ |
| 2 | $A, B, C, E, D, A$ | 14 | $A, D, E, C, B, A$ |
| 3 | $A, B, D, C, E, A$ | 15 | $A, E, C, D, B, A$ |
| 4 | $A, B, D, E, C, A$ | 16 | $A, C, E, D, B, A$ |
| 5 | $A, B, E, C, D, A$ | 17 | $A, D, C, E, B, A$ |
| 6 | $A, B, E, D, C, A$ | 18 | $A, C, D, E, B, A$ |
| 7 | $A, C, B, D, E, A$ | 19 | $A, E, D, B, C, A$ |
| 8 | $A, C, B, E, D, A$ | 20 | $A, D, E, B, C, A$ |
| 9 | $A, C, D, B, E, A$ | 21 | $A, E, B, D, C, A$ |
| 10 | $A, C, E, B, D, A$ | 22 | $A, D, B, E, C, A$ |
| 11 | $A, D, B, C, E, A$ | 23 | $A, E, C, B, D, A$ |
| 12 | $A, D, C, B, E, A$ | 24 | $A, E, B, C, D, A$ |

# Example 2 Hamilton Circuits in $K_5$

Here are three of the 24 Hamilton circuits in $K_5$.



# Number of Hamilton Circuits

*"What is the number of Hamilton circuits in $K_N$?"* boils down to the equivalent question,

*"How many different ways are there to rearrange the (N – 1) vertices?*

*The answer is given by the number*

*$1 \times 2 \times 3 \times \ldots \times (N - 1)$, called the **factorial** of (N – 1) and written as (N – 1)! for short.*

# NUMBER OF HAMILTON CIRCUITS IN $K_N$

- $K_N$ has $N(N - 1)/2$ edges.
- Of all graphs with $N$ vertices and no multiple edges or loops, $K_N$ has the most edges.

# Number of Hamilton Circuits

The table on the next slide shows the number of Hamilton circuits in complete graphs with up to $N = 20$ vertices.

Notice that as we increase the number of vertices, the number of Hamilton circuits in the complete graph goes through the roof.

*Even a relatively small graph such as has $K_8$ more than five thousand Hamilton circuits. Double the number of vertices to $K_{16}$ and the number of Hamilton circuits exceeds 1.3 trillion.*

## Number of Distinct Hamilton Circuits in $K_N$

| $N$ | $(N-1)!$ | $N$ | $(N-1)!$ |
|---|---|---|---|
| 3 | 2 | 12 | 39,916,800 |
| 4 | 6 | 13 | 479,001,600 |
| 5 | 24 | 14 | 6,227,020,800 |
| 6 | 120 | 15 | 87,178,291,200 |
| 7 | 720 | 16 | 1,307,674,368,000 |
| 8 | 5040 | 17 | 20,922,789,888,000 |
| 9 | 40,320 | 18 | 355,687,428,096,000 |
| 10 | 362,880 | 19 | 6,402,373,705,728,000 |
| 11 | 3,628,800 | 20 | 121,645,100,408,832,000 |

# Traveling Salesman Problem

Any problem that shares these common elements:

1) a *traveler*,
2) a set of *sites*,
3) a *cost* function for travel between pairs of sites,
4) a need to *tour all* the sites, and
5) a desire to *minimize* the total cost of the tour

is known as a **traveling salesman problem**, or **TSP**.

# Routing School Buses

A school bus *(the traveler)* picks up children in the morning and drops them off at the end of the day at designated stops *(the sites)*.

On a typical school bus route there may be 20 to 30 such stops. With school buses, total time on the bus is always the most important variable (students have to get to school on time), and there is a known time of travel *(the cost)* between any two bus stops.

Since children must be picked up at every bus stop, a tour of all the sites *(starting and ending at the school)* is required. Since the bus repeats its route every day during the school year, *finding an optimal tour is crucial*.

# Delivering Packages

Package delivery companies such as UPS and FedEx deal with TSPs on a daily basis.

Each truck is a *traveler* that must deliver packages to a specific list of delivery destinations *(the sites)*. The travel time between any two delivery sites *(the cost)* is known or can be estimated. Each day the truck must *deliver to all the sites on its list*, so a *tour* is an implied part of the requirements. Since one can assume that the driver would rather be home than out delivering packages, an optimal tour is a highly desirable goal.

# Modeling a TSP

Every TSP can be modeled by a *weighted graph*, *that is, a graph such that there is a number associated with each edge (called the weight of the edge)*.

The beauty of this approach is that the model always has the same structure: *The vertices of the graph are the sites of the TSP, and there is an edge between X and Y if there is a direct link for the traveler to travel from site X to site Y.* Moreover, the weight of the edge *XY* is the cost of travel between *X* and *Y*.

# Modeling a TSP

In this setting a tour is a Hamilton circuit of the graph, and an optimal tour is the Hamilton circuit of least total weight.

In all the applications and examples we will be considering, *we will make the assumption that there is an edge connecting every pair of sites, which implies that the underlying graph model is always a complete weighted graph.* The following is a summary of the preceding observations.

# GRAPH MODEL OF A TRAVELING SALESMAN PROBLEM

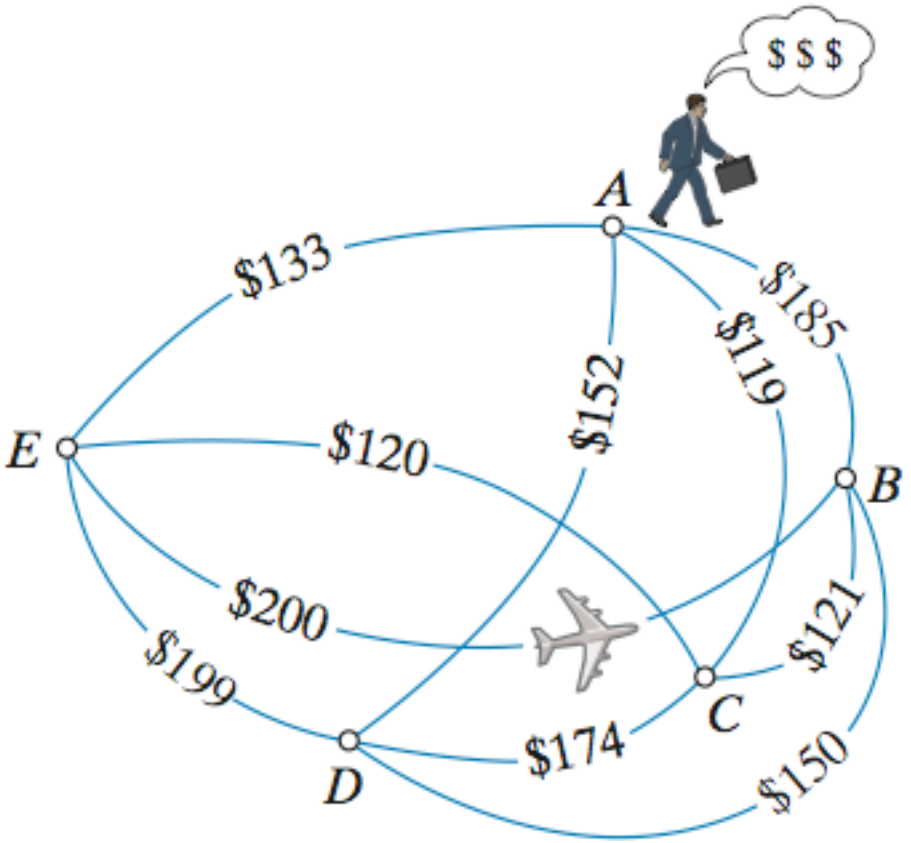
- Sites → vertices of the graph.
- Costs → weights of the edges.
- Tour → Hamilton circuit.
- Optimal tour → Hamilton circuit of least total cost

# Example 3 A Tour of Five Cities

Willy a traveling salesman is pondering his upcoming sales trip. He wants to tour five cities; A, B, C, D and E.

The one-way airfares between any two cities are shown on the graph.

Wiley would like to find an optimal tour.



# Traveling Salesman Problems

We will explore two strategies for solving traveling salesman problems:

1. ***Exhaustive Search***

2. ***Go Cheap***

# STRATEGY 1 (EXHAUSTIVE SEARCH)

Make a list of all possible Hamilton circuits. For each circuit in the list, calculate the weight of the circuit. From all the circuits, choose a circuit with least total weight. This is your optimal tour.

# Example 4 A Tour of Five Cities: Part 2 - STRATEGY 1

The table on the next slide shows a detailed implementation of this strategy.

The 24 Hamilton circuits are split into two columns consisting of circuits and their mirror images.

The total costs for the circuits are shown in the middle column of the table. *There are two optimal tours, with a total cost of $676 – A, D, B, C ,E, A and its mirror image A, E, C, B, D, A.*

# Example 4 A Tour of Five Cities: Part 2 - STRATEGY 1

The first group of 12 Hamilton circuits:

**Willy's Possible Tours and Their Costs**

| | Hamilton circuit | Total cost | Mirror-image circuit |
|---|---|---|---|
| 1 | $A, B, C, D, E, A$ | $185 + 121 + 174 + 199 + 133 = 812$ | $A, E, D, C, B, A$ |
| 2 | $A, B, C, E, D, A$ | $185 + 121 + 120 + 199 + 152 = 777$ | $A, D, E, C, B, A$ |
| 3 | $A, B, D, C, E, A$ | $185 + 150 + 174 + 120 + 133 = 762$ | $A, E, C, D, B, A$ |
| 4 | $A, B, D, E, C, A$ | $185 + 150 + 199 + 120 + 119 = 773$ | $A, C, E, D, B, A$ |
| 5 | $A, B, E, C, D, A$ | $185 + 200 + 120 + 174 + 152 = 831$ | $A, D, C, E, B, A$ |
| 6 | $A, B, E, D, C, A$ | $185 + 200 + 199 + 174 + 119 = 877$ | $A, C, D, E, B, A$ |

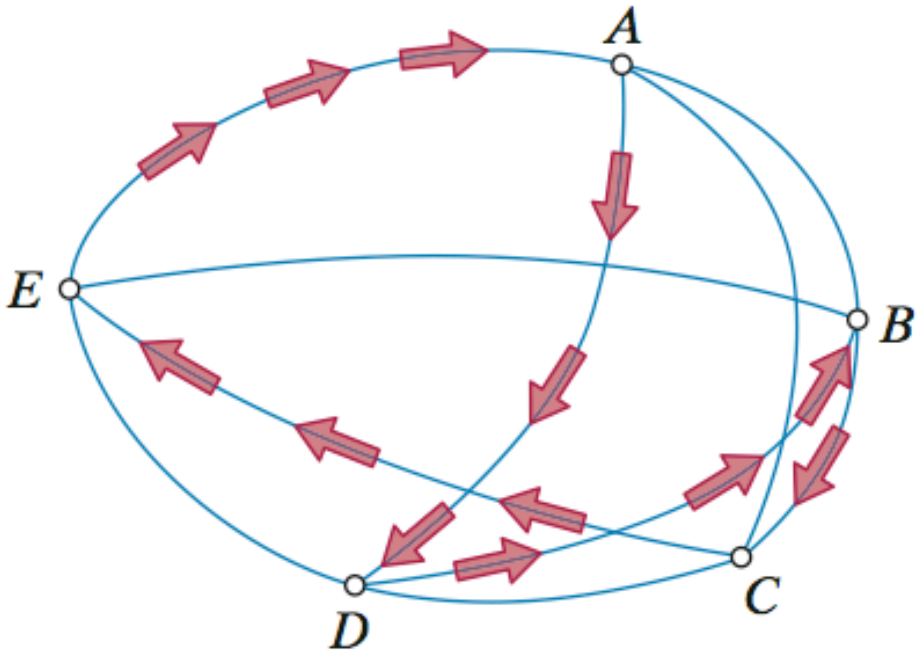
# Example 4 A Tour of Five Cities: Part 2 - STRATEGY 1

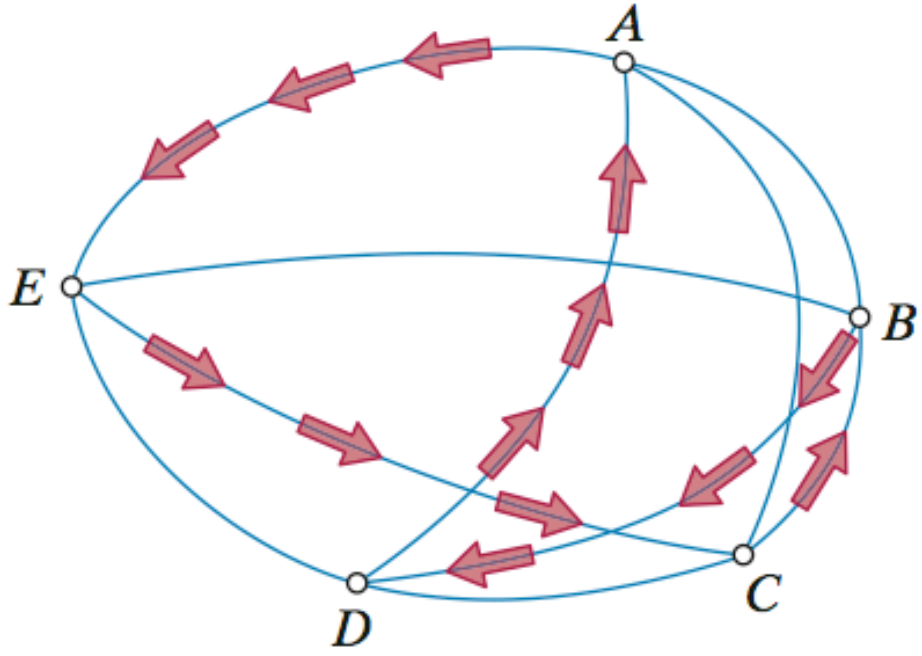
The second group of 12 Hamilton circuits:

| | | | |
|---|---|---|---|
| 7 | $A, C, B, D, E, A$ | $119 + 121 + 150 + 199 + 133 = 722$ | $A, E, D, B, C, A$ |
| 8 | $A, C, B, E, D, A$ | $119 + 121 + 200 + 199 + 152 = 791$ | $A, D, E, B, C, A$ |
| 9 | $A, C, D, B, E, A$ | $119 + 174 + 150 + 200 + 133 = 776$ | $A, E, B, D, C, A$ |
| 10 | $A, C, E, B, D, A$ | $119 + 120 + 200 + 150 + 152 = 741$ | $A, D, B, E, C, A$ |
| 11 | $A, D, B, C, E, A$ | $152 + 150 + 121 + 120 + 133 = 676$ | $A, E, C, B, D, A$ |
| 12 | $A, D, C, B, E, A$ | $152 + 174 + 121 + 200 + 133 = 780$ | $A, E, B, C, D, A$ |

# Example 4 A Tour of Five Cities: Part 2 - STRATEGY 1

There are always going to be at least two optimal tours, *since the mirror image of an optimal tour is also optimal. Either one of them can be used for a solution.*



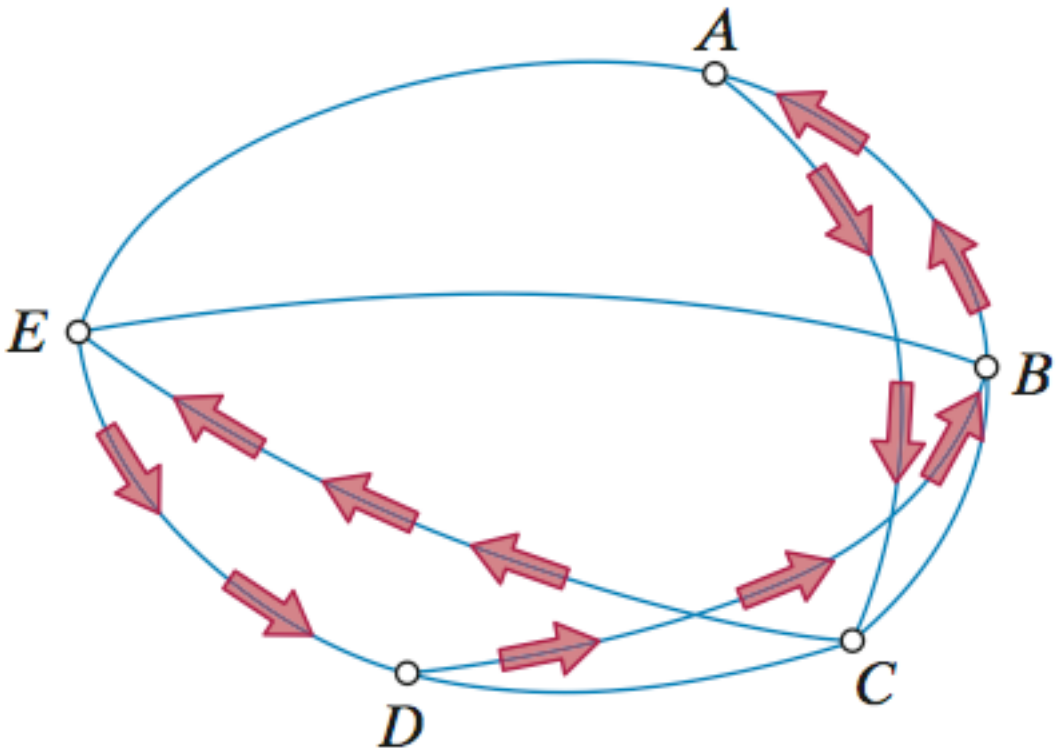
# STRATEGY 2 (GO CHEAP)

- Start from the home city. From there go to the city that is the cheapest to get to. From each new city go to the next new city that is cheapest to get to. When there are no more new cities to go to, go back home.

# Example 5 A Tour of Five Cities: Part 2 - Strategy 2

We assume that Willy lives in city A. In Willy case, the strategy works like this: Start at *A*. From *A* Willy goes to *C* (the cheapest place he can fly to from *A*).

*From C to E, E to D, and from D the last new city to go to is B. From B Willy has to return home to A. The total cost of this tour is $773.*



# Example 5 A Tour of Five Cities: Part 2 - Strategy 2

The "go cheap" strategy takes a lot less work than the "exhaustive search" strategy, but there is a hitch – *the cost of the tour we get is $773, which is $97 more than the optimal tour found under the exhaustive search strategy.*

# Algorithms for Solving TSP's

In this section we will look at the two strategies we informally developed in connection with Willy's sales trips and recast them in the language of algorithms.

The "exhaustive search" strategy can be formalized into an algorithm generally known as the **brute-force algorithm**;

the "go cheap" strategy can be formalized into an algorithm known as the **nearest-neighbor algorithm**.

# ALGORITM 1: THE BRUTE FORCE ALGORITHM

- **Step 1.** Make a list of *all* the possible Hamilton circuits of the graph. Each of these circuits represents a tour of the vertices of the graph.
- **Step 2.** For each tour calculate its weight (i.e., add the weights of all the edges in the circuit).
- **Step 3.** Choose an *optimal* tour (there is always more than one optimal tour to choose from!).

# ALGORITM 2: THE NEAREST NEIGHBOR ALGORITHM

- **Start:** Start at the designated starting vertex.If there is no designated starting vertex pick any vertex.

- **First step:** From the starting vertex go to its nearest neighbor(i.e.,the vertex for which the corresponding edge has the smallest weight).

# ALGORITM 2: THE NEAREST NEIGHBOR ALGORITHM

- **Middle steps:** From each vertex go to its nearest neighbor, choosing only among the vertices that haven't been yet visited. (If there is more than one nearest neighbor choose among them at random.) Keep doing this until all the vertices have been visited.
- **Last step:** From the last vertex return to the starting vertex.

# Pros and Cons

*The positive aspect of the brute-force algorithm is that it is an optimal algorithm.*

(An **optimal algorithm** is an algorithm that, when correctly implemented, is guaranteed to produce an optimal solution.)

In the case of the brute-force algorithm, we know we are getting an optimal solution because we are choosing from among all possible tours.

# Pros and Cons

The negative aspect of the brute-force algorithm is the amount of effort that goes into implementing the algorithm, which is (roughly) proportional to the number of Hamilton circuits in the graph.

# Practical Terms of the Pros and Cons

Let's start with human computation. *To solve a TSP for a graph with 10 vertices , the brute-force algorithm requires checking 362,880 Hamilton circuits.*

To do this by hand, even for a fast and clever human, it would take over 1000 hours. Thus, at *N* =10 we are already beyond the limit of what can be considered reasonable human effort.

# Using a Computer

Even with the world's best technology on our side, *we very quickly reach the point beyond which using the brute-force algorithm is completely unrealistic.*

*Superhero is the fastest supercomputer on the planet.*

| *N* | SUPERHERO computation time |
|---|---|
| 20 | 2 minutes |
| 21 | 40 minutes |
| 22 | 14 hours |
| 23 | 13 days |
| 24 | 10 months |
| 25 | 20 years |
| 26 | 500 years |
| 27 | 13,000 years |
| 28 | 350,000 years |
| 29 | 9.8 million years |
| 30 | 284 million years |

# Using a Computer

The brute-force algorithm is a classic example of what is formally known as *an **inefficient algorithm***–

*an algorithm for which the number of steps needed to carry it out grows disproportionately with the size of the problem.*

The trouble with inefficient algorithms is that they are of limited practical use–they can realistically be carried out only when the problem is small.

# Pros and Cons - Nearest Neighbor

We hop from vertex to vertex using a simple criterion: Choose the next available "nearest" vertex and go for it. ***Let's call the process of checking among the available vertices and finding the nearest one a single computation.***

Then, for a TSP with $N = 5$ we need to perform 5 computations. What happens when we double, the number of vertices to

$N = 10$? We now have to perform 10 computations. For $N = 30$, we perform 30 computations.

# Pros and Cons - Nearest Neighbor

We can summarize the above observations by saying *that the nearest-neighbor algorithm is an efficient algorithm*.

Roughly speaking, an **efficient algorithm** is an algorithm for which the amount of computational effort required to implement the algorithm grows in some reasonable proportion with the size of the input to the problem.

# Pros and Cons - Nearest Neighbor

*The main problem with the nearest-neighbor algorithm is that it is not an optimal algorithm*. In Example 5, the **nearest-neighbor** tour had a cost of $773, whereas the *optimal tour* had a cost of $676.

In absolute terms the nearest-neighbor tour is off by $97 (this is called the absolute error). A better way to describe how far "off" this tour is from the optimal tour is to use the concept of *relative error*. In this example the absolute error is $97 out of $676, giving a relative error of $97/$676 = 0.1434911 ≈ 14.35%.

In general, for any tour that might be proposed as a "solution" to a TSP, *we can find its* ***relative error*** $\varepsilon$ *as follows:*

## RELATIVE ERROR OF A TOUR (e)

$$\varepsilon = \frac{\left(\text{cost of tour} - \text{cost of optimal tour}\right)}{\text{cost of optimal tour}}$$

# Relative Error - Good or Not Good

It is customary to express the relative error as a percentage (usually rounded to two decimal places).

By using the notion of relative error, we can characterize the optimal tours as those with relative error of 0%.

All other tours give "approximate solutions," the relative merits of which we can judge by their respective relative errors: tours with "small" relative errors are good, and tours with "large" relative errors are not good.

# Repetitive Nearest-Neighbor Algorithm

As one might guess, the repetitive nearest-neighbor algorithm is a variation of the nearest-neighbor algorithm in which we repeat several times the entire nearest- neighbor circuit-building process.

Why would we want to do this? The reason is that the nearest-neighbor tour depends on the choice of the starting vertex. If we change the starting vertex, the nearest-neighbor tour is likely to be different, and, if we are lucky, better.

# Repetitive Nearest-Neighbor Algorithm

Since finding a nearest-neighbor tour is an efficient process, it is not unreasonable to repeat the process several times, each time starting at a different vertex of the graph.

In this way, we can obtain several different "nearest-neighbor solutions," from which we can then pick the best.

# Repetitive Nearest-Neighbor Algorithm

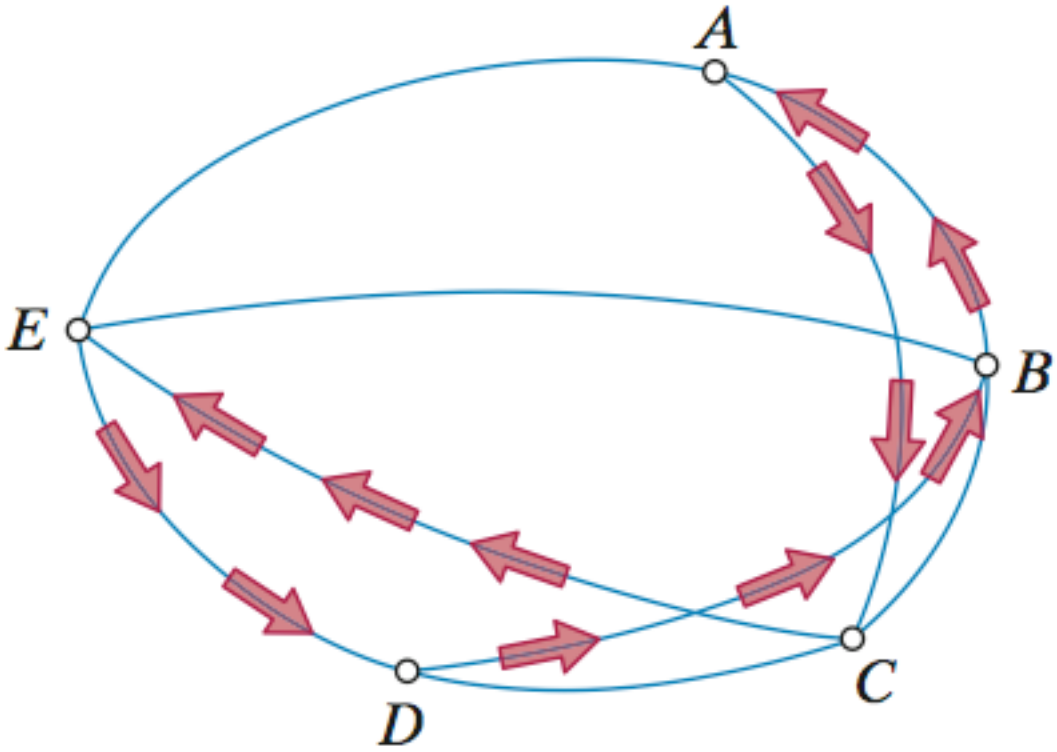
But what do we do with a tour that starts somewhere other than the vertex we really want to start at?

That's not a problem. Remember that once we have a circuit, we can start the circuit anywhere we want.

In fact, in an abstract sense, a circuit has no starting or ending point.

# Example 6 A Tour of Five Cities: Part 3

In Example 5, we computed the nearest-neighbor tour *with A as the starting vertex, and we got A, C, E, D, B, A with a total cost of $773.*

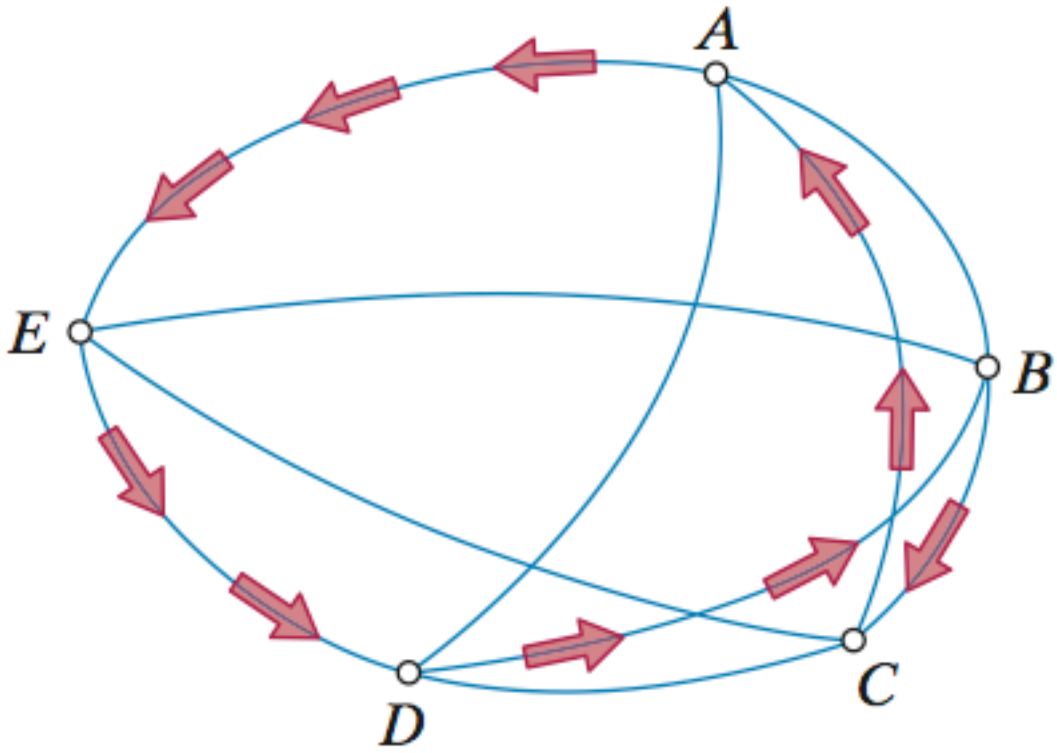


# Example 6 A Tour of Five Cities: Part 3

But if we use *B* as the starting vertex, the nearest-neighbor tour takes us from *B* to *C*, then to *A*, *E*, *D*, and back to *B*, with a total cost of $722. Well, that is certainly an improvement!

As for Willy–who must start and end his trip at *A*–this very same tour would take the form

*A*, *E*, *D*, *B*, *C*, *A*.



# Example 6 A Tour of Five Cities: Part 3

*The process is once again repeated using C, D, and E as the starting vertices.*

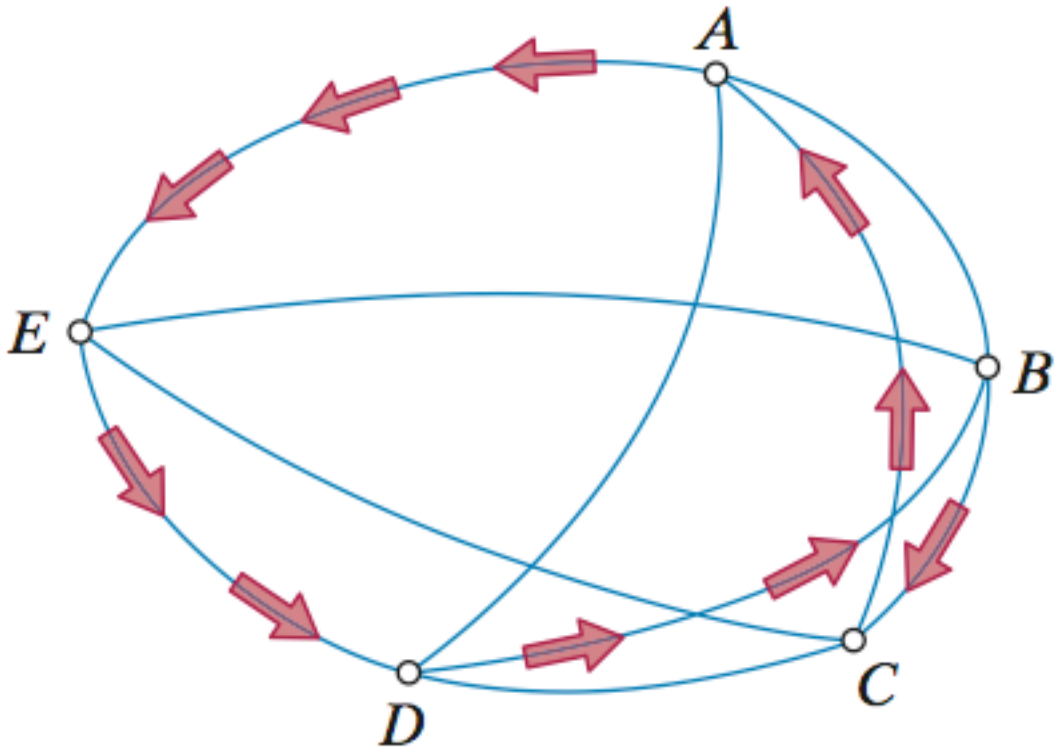
When the starting point is *C*, we get the nearest-neighbor tour *C*, *A*, *E*, *D*, *B*, *C* (total cost is $722);

when the starting point is *D*, we get the nearest-neighbor tour *D*, *B*, *C*, *A*, *E*, *D* (total cost is $722); and

when the starting point is *E*, we get the nearest-neighbor tour *E*, *C*, *A*, *D*, *B*, *E* (total cost is $741).

# Example 6 A Tour of Five Cities: Part 3

Among all the nearest-neighbor tours we pick the cheapest one – *A, E, D, B, C, A – with a cost of $722*.



# ALGORITM 3: THE REPETITIVE NEAREST NEIGHBOR ALGORITHM

- Let *X* be any vertex. Find the nearest-neighbor tour with *X* as the starting vertex, and calculate the cost of this tour.
- Repeat the process with each of the other vertices of the graph as the starting vertex.
- Of the nearest-neighbor tours thus obtained, choose one with least cost. If necessary, rewrite the tour so that it starts at the designated starting vertex. We will call this tour the *repetitive nearest-neighbor tour.*

# Cheapest-Link Algorithm

*The idea behind the **cheapest-link algorithm** is to piece together a tour by picking the separate "links" (i.e., legs) of the tour on the basis of cost.*

It doesn't matter if in the intermediate stages the "links" are all over the place – if we are careful at the end, they will all come together and form a tour.

This is how it works: *Look at the entire graph and choose the cheapest edge of the graph, wherever that edge may be. Once this is done, choose the next cheapest edge of the graph, wherever that edge may be.*

# Cheapest-Link Algorithm

Don't worry if that edge is not adjacent to the first edge. Continue this way, each time choosing the cheapest edge available but following two rules:

(1) Do not allow a circuit to form except at the very end, and

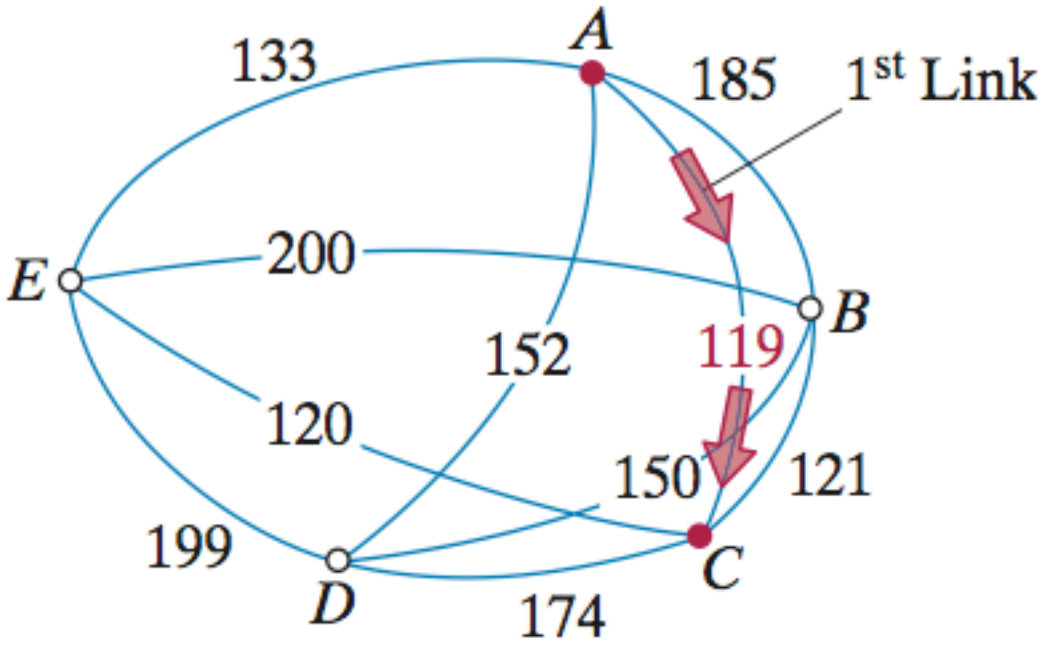
(2) do not allow three edges to come together at a vertex.

A violation of either of these two rules will prevent forming a Hamilton circuit. Conversely, following these two rules guarantees that the end result will be a Hamilton circuit.

# Example 7 A Tour of Five Cities: Part 4

Among all the edges of the graph, the "cheapest link" is edge *AC*, with a cost of $119. For the purposes of recordkeeping,

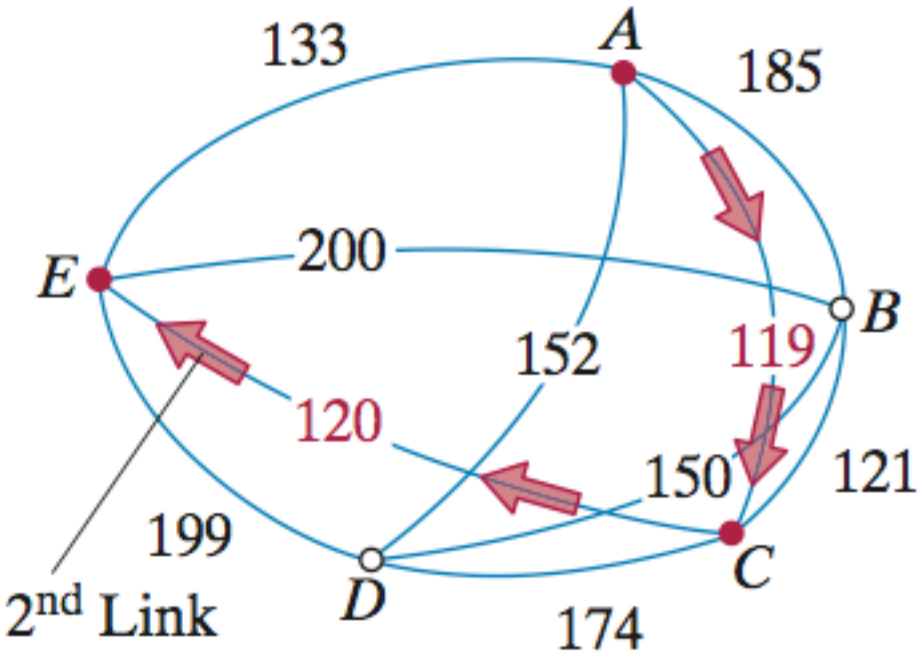
we will tag this edge as taken by marking it in red as shown.



# Example 7 A Tour of Five Cities: Part 4

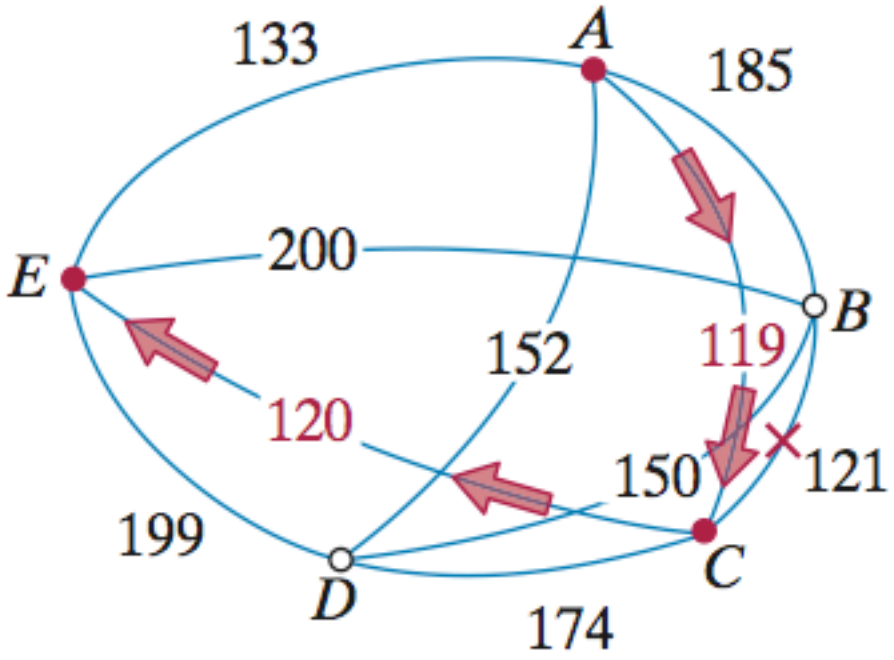
The next step is to scan the entire graph again and choose the cheapest link available, in this case edge *CE* with a cost of $120.

Again, we mark it in red, to indicate it is taken.



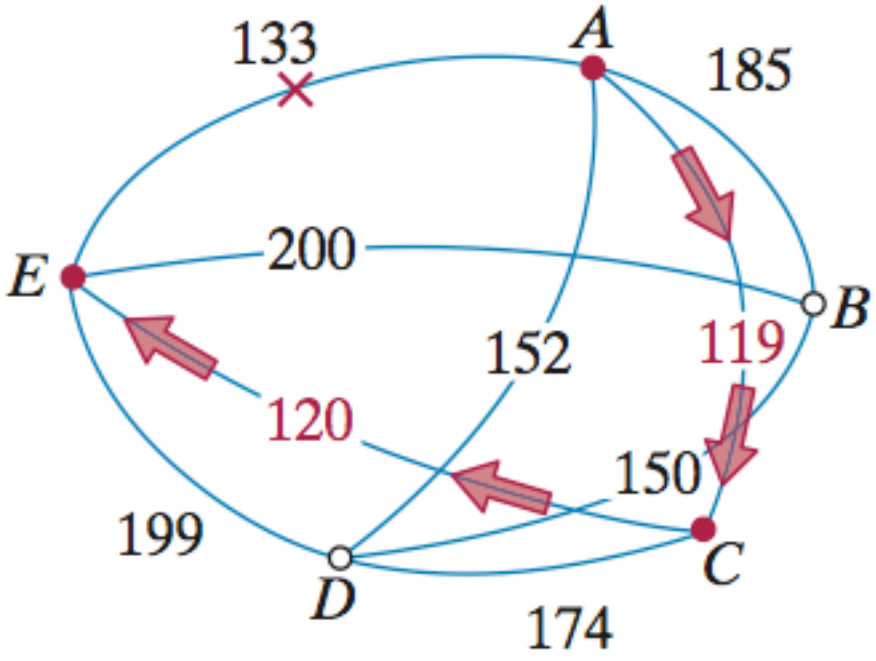
# Example 7 A Tour of Five Cities: Part 4

The next cheapest link available is edge *BC* ($121), but we should not choose *BC*–we would have three edges coming out of vertex *C* and this would prevent us from forming a circuit. This time we put a "do not use" mark on edge *BC*.



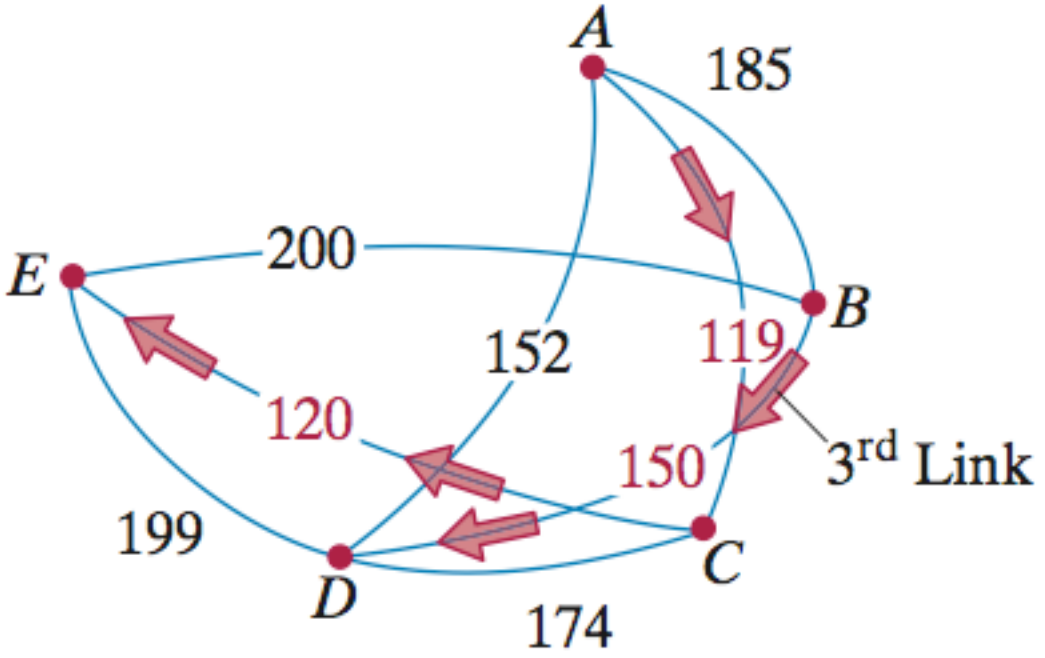
# Example 7 A Tour of Five Cities: Part 4

The next cheapest link available is *AE* ($133). But we can't take *AE* either–the vertices *A*, *C*, and *E* would form a small circuit, making it impossible to form a Hamilton circuit at the end. So we put a "do not use" mark on *AE*.



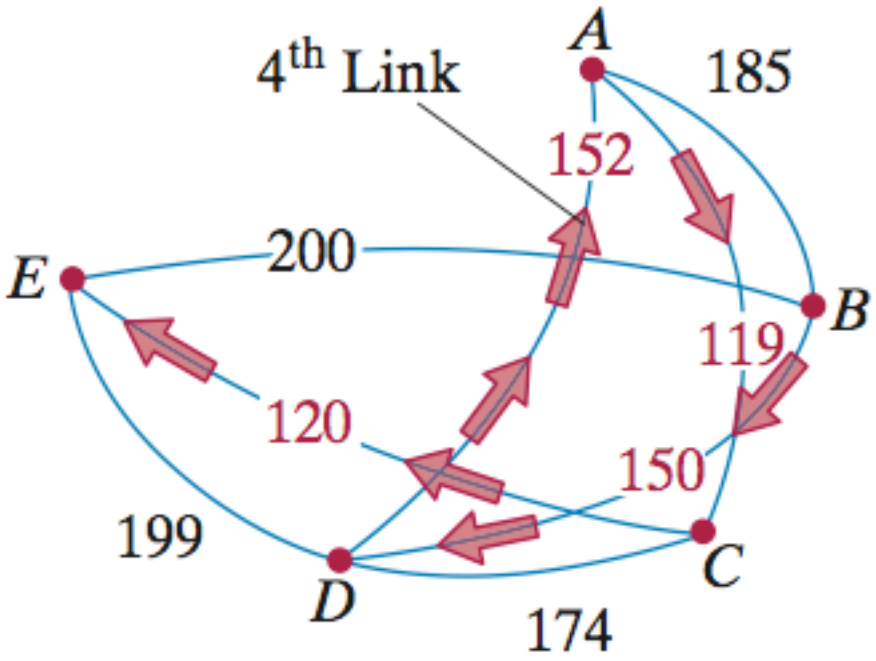
# Example 7 A Tour of Five Cities: Part 4

The next cheapest link available is *BD* ($150). Choosing *BD* would not violate either of the two rules, so we can add it to our budding circuit and mark it in red.



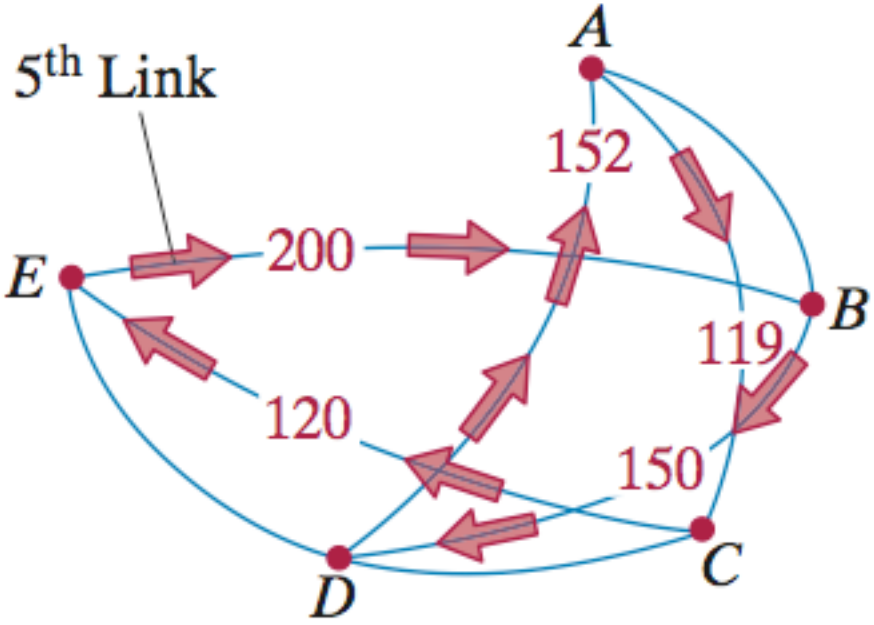
# Example 7 A Tour of Five Cities: Part 4

The next cheapest link available is *AD* ($152), and it works just fine.



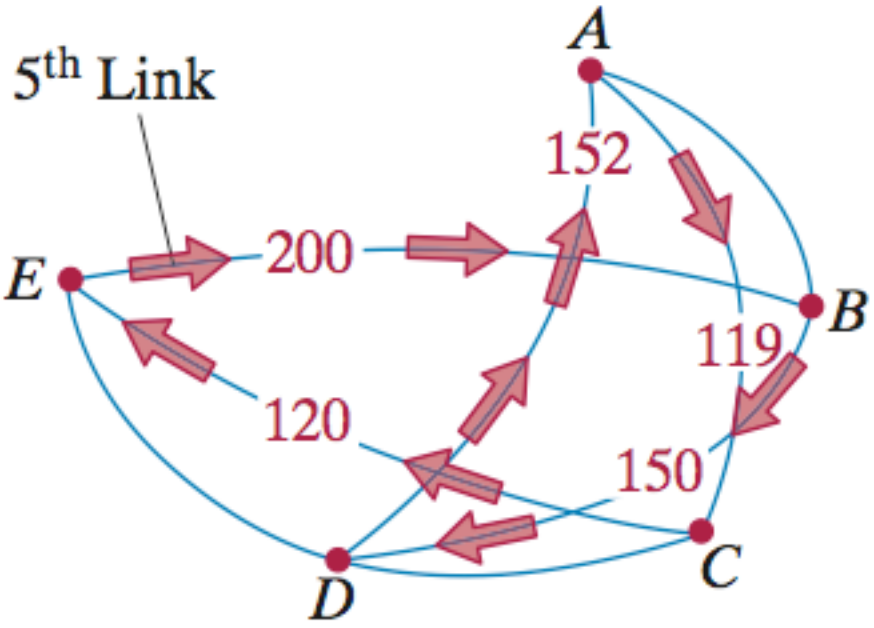
# Example 7 A Tour of Five Cities: Part 4

At this point, we have only one way to close up the Hamilton circuit, edge *BE*, as shown.



# Example 7 A Tour of Five Cities: Part 4

The cheapest-link tour shown could have any starting vertex we want. Since Willy lives at *A*, we describe it as *A*, *C*, *E*, *B*, *D*, *A*. The total cost of this tour is $741, which is a little better than the nearest-neighbor tour but not as good as the repetitive nearest-neighbor tour.



# ALGORITM 4: THE CHEAPEST LINK ALGORITHM

- **Step 1.** Pick the *cheapest link* (i.e., edge with smallest weight) available. (In case of a tie, pick one at random.) Mark it (say in red).

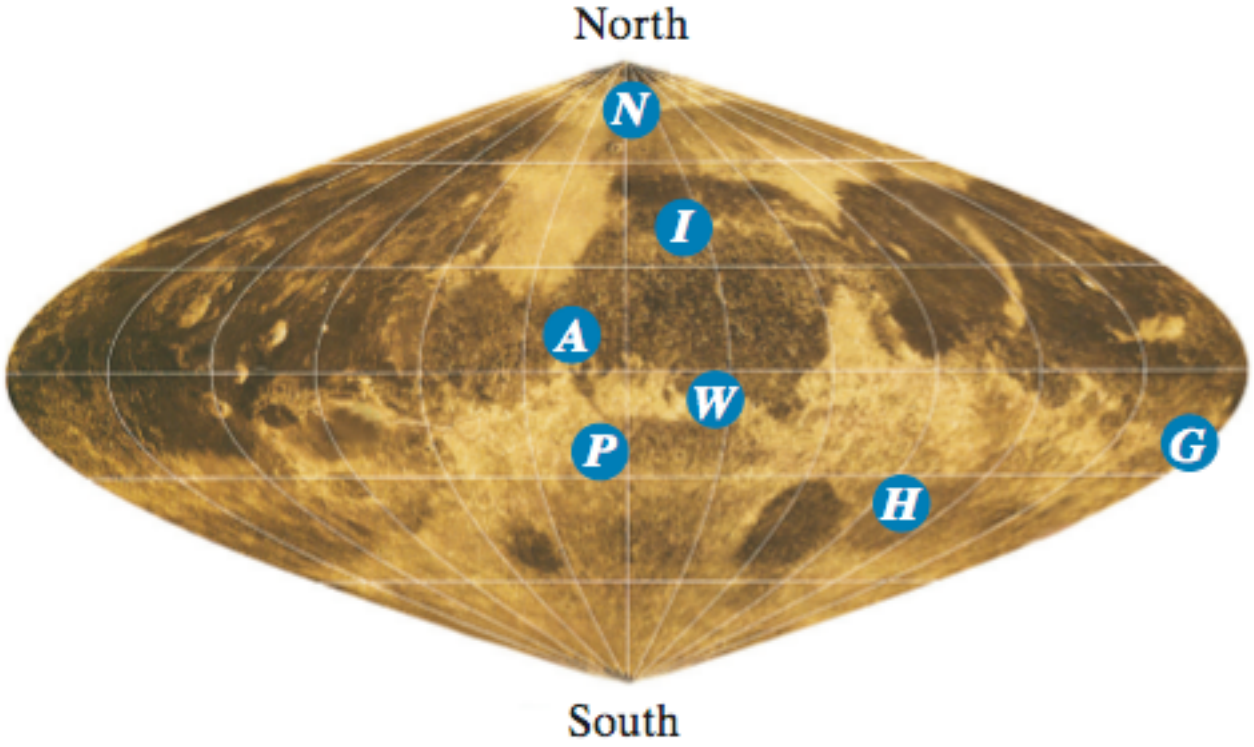
- **Step 2.** Pick the next cheapest link available and mark it.

# ALGORITM 4: THE CHEAPEST LINK ALGORITHM

- **Step 3, 4, …, *N* – 1.** Continue picking and marking the cheapest unmarked link available that does not
  (a) close a circuit or
  (b) create three edges coming out of a single vertex
- **Step *N*.** Connect the last two vertices to close the red circuit. This circuit gives us the cheapest-link tour.

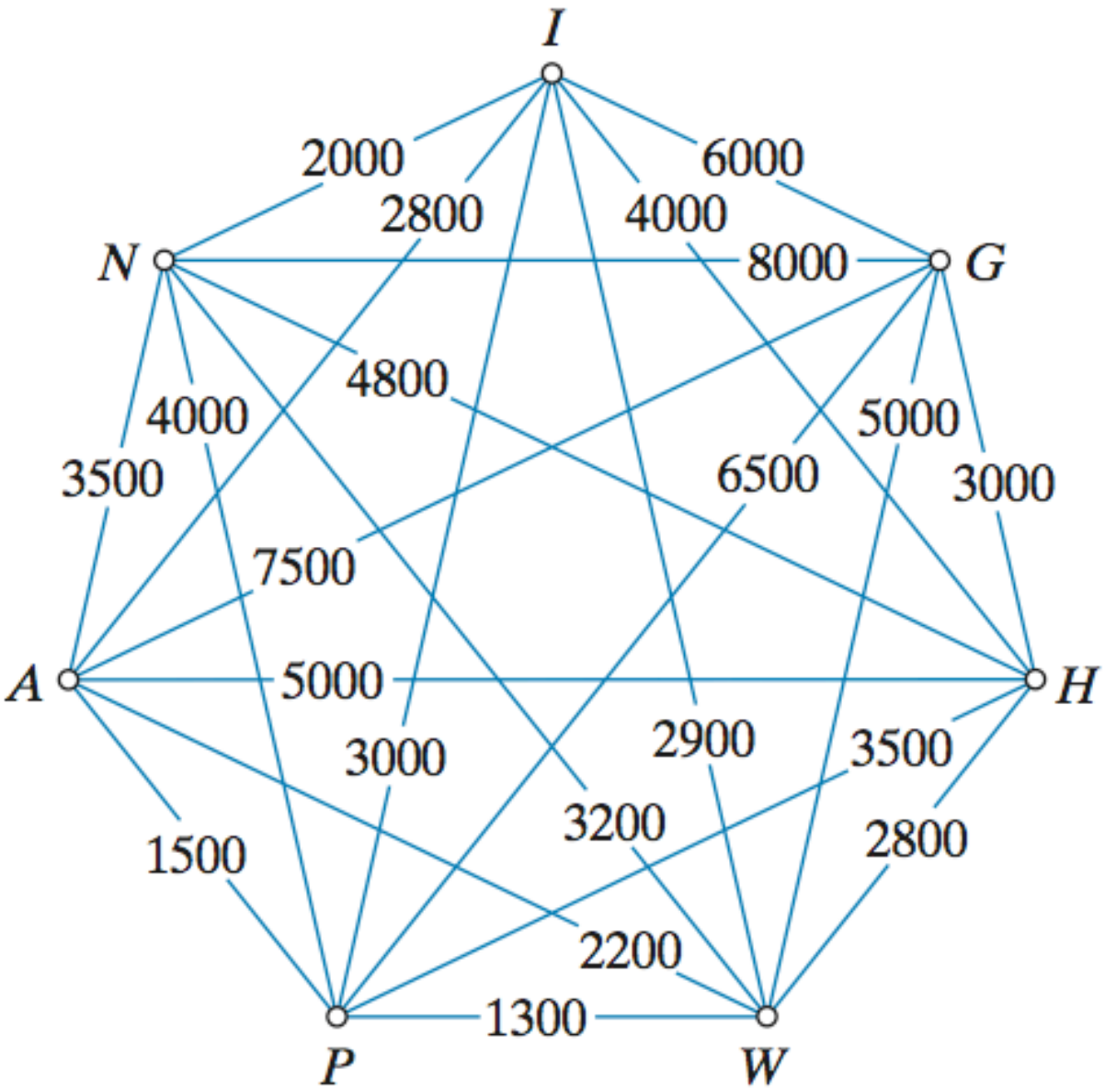
# Example 8 Roving the Red Planet

The figure shows seven sites on Mars identified as particularly interesting sites for geological exploration. Our job is to find an optimal tour for a rover that will land at *A*, visit all the sites to collect rock samples, and at the end return to *A*.



# Example 8 Roving the Red Planet

The distances between sites (in miles) are given in the graph shown.

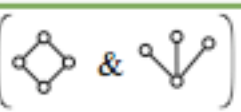
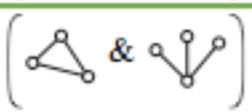


# Example 8 Roving the Red Planet

Let's look at some of the approaches one might use to tackle this problem.

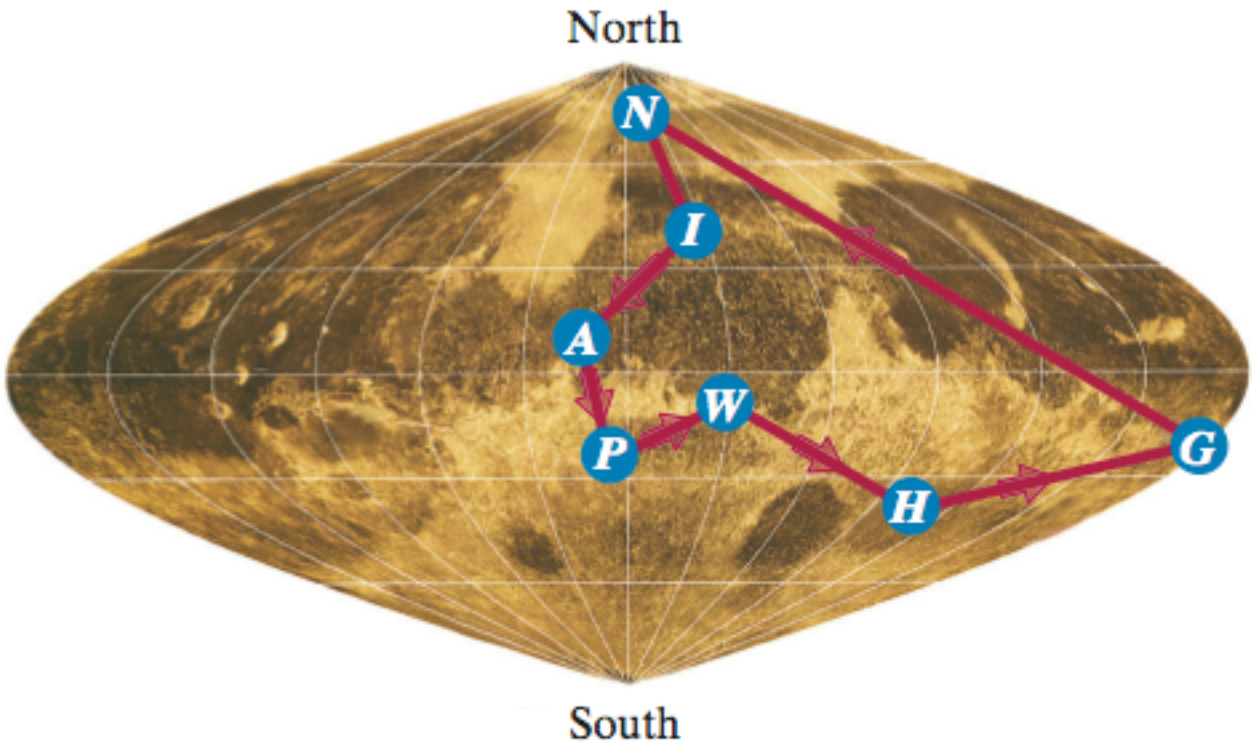
***Brute force**: The brute-force algorithm would require us to check and compute 6! = 720 different tours. We will pass on that idea for now.*

***Cheapest link**: The cheapest-link algorithm is a reasonable algorithm to use – not trivial but not too hard either. A summary of the steps is shown in the table on the next slide.*

| Step | Cheapest edge available | Weight | Add to circuit? |
|---|---|---|---|
| 1 | *PW* | 1300 | Yes |
| 2 | *AP* | 1500 | Yes |
| 3 | *IN* | 2000 | Yes |
| 4 | *AW* | 2200 | No  |
| 5 | *HW* | 2800 | Yes |
| 6 | *AI* | 2800 | Yes |
| | tie | | |
| 7 | *IW* | 2900 | No  |
| 8 | *IP* | 3000 | No  |
| 9 | *GH* | 3000 | Yes |
| | tie | | |
| Last | *GN* only way to close circuit | 8000 | Yes |

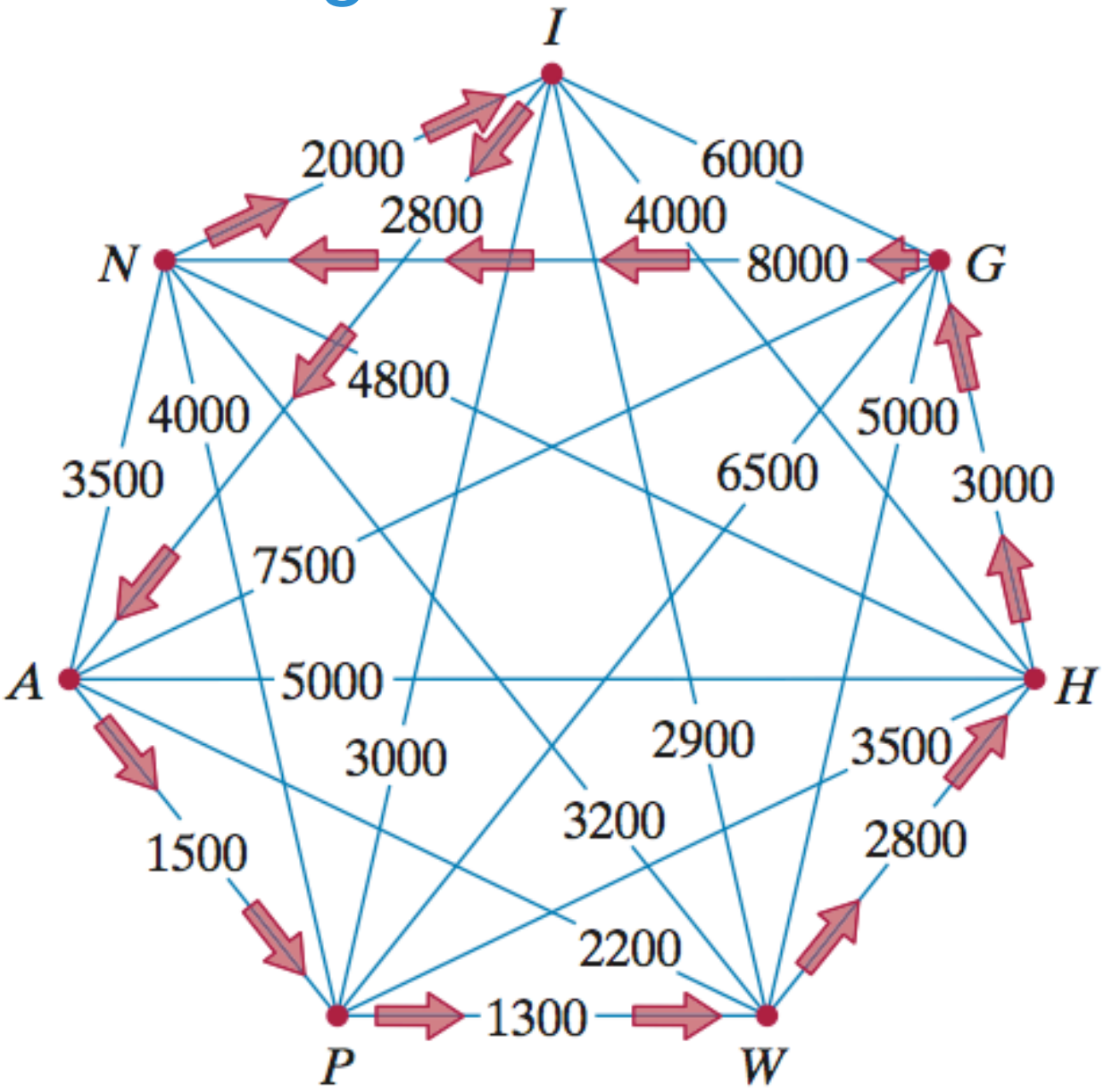
# Example 8 Roving the Red Planet

*The cheapest-link tour (A, P, W, H, G, N, I, A with a total length of 21,400 miles) is shown.*



# Example Roving the Red Planet

*Here is the cheapest-link tour again (A, P, W, H, G, N, I, A with a total length of 21,400 miles) is shown.*



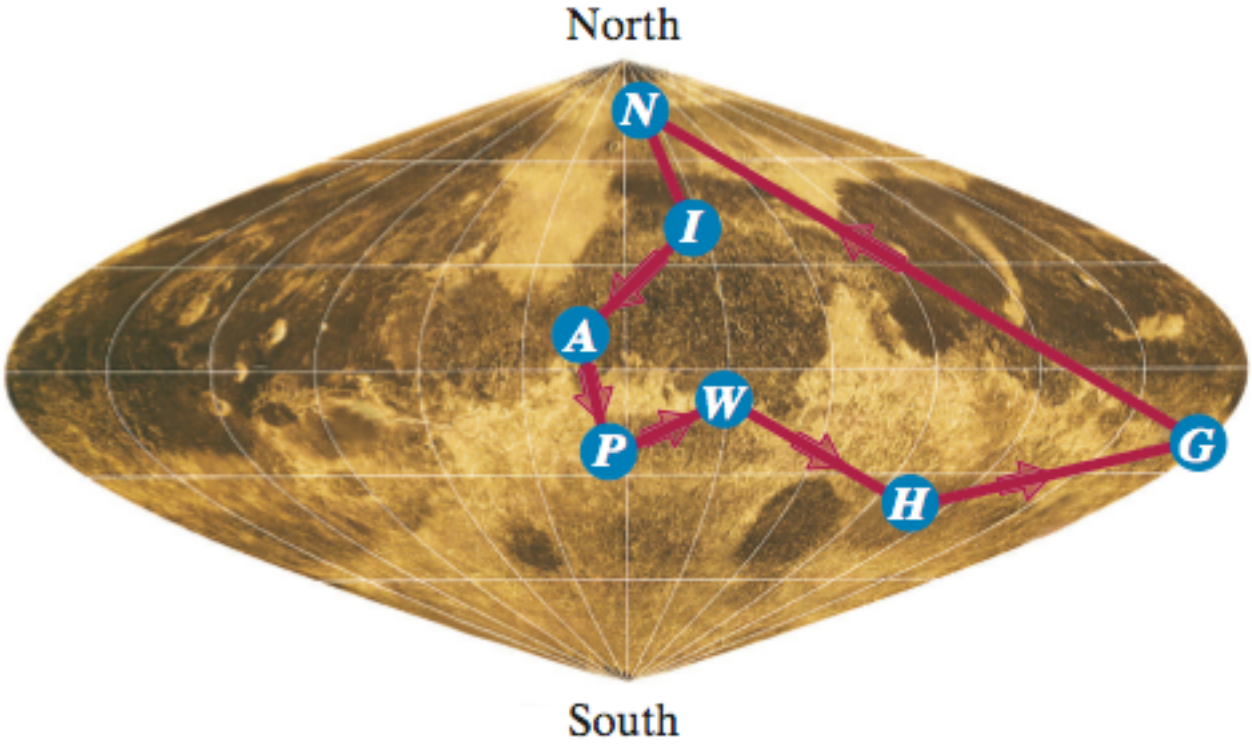
# Example Roving the Red Planet

**Nearest neighbor:** *The nearest-neighbor algorithm is the simplest of all the algorithms we learned. Starting from A we go to P, then to W, then to H, then to G, then to I, then to N, and finally back to A.*

The nearest-neighbor tour (*A*, *P*, *W*, *H*, *G*, *I*, *N*, *A* with a total length of 20,100 miles) is shown on the next two slides. (We know that we can repeat this method using different starting points, but we won't bother with that at this time.)

# Example Roving the Red Planet

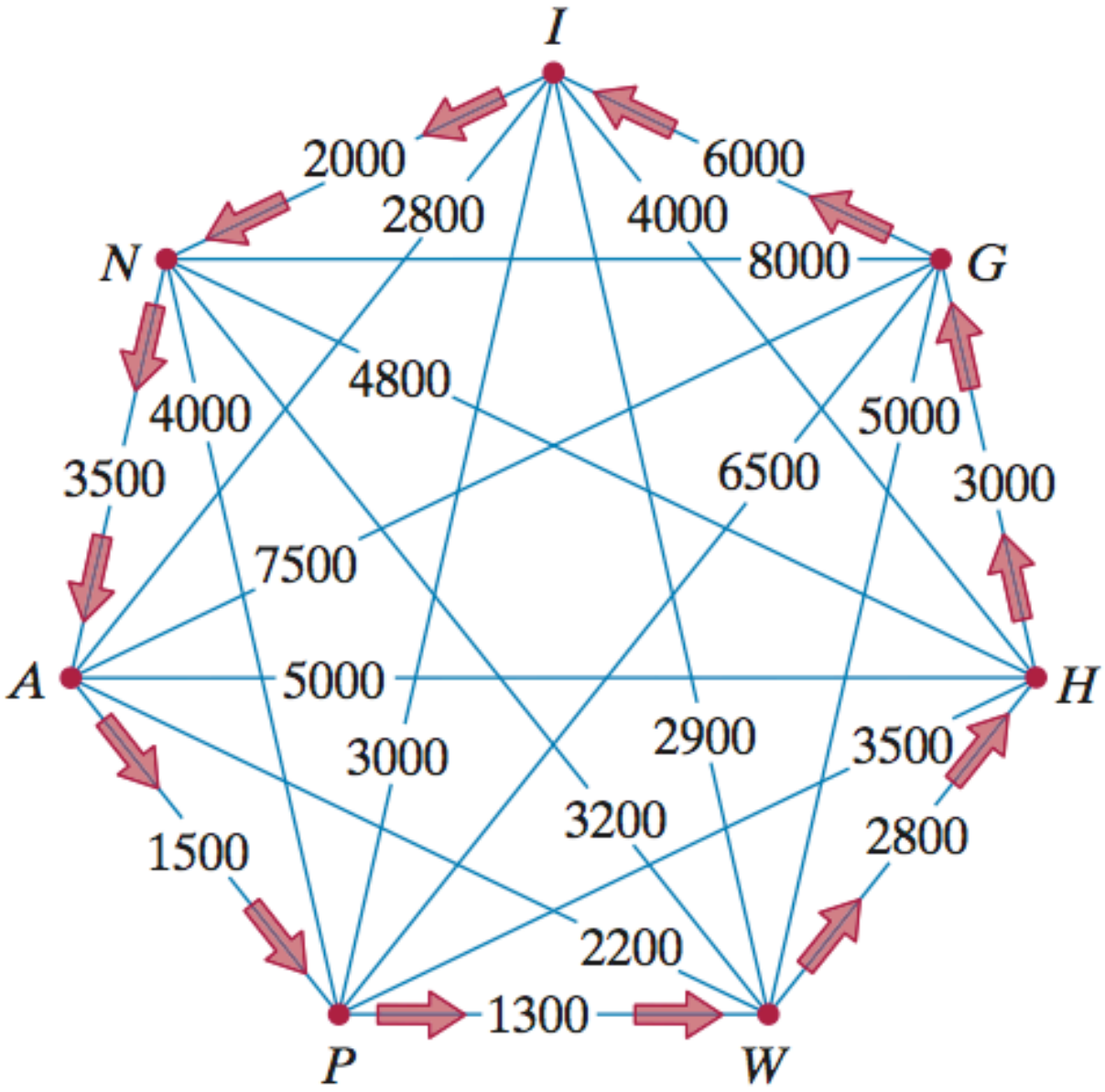
**Nearest neighbor:** *A*, *P*, *W*, *H*, *G*, *I*, *N*, *A* with a total length of 20,100 miles.



# Example Roving the Red Planet

**Nearest neighbor:**

*A*, *P*, *W*, *H*, *G*, *I*, *N*, *A* with a total length of 20,100 miles.



# Surprise One

*The first surprise in in the last example is that the nearest-neighbor algorithm gave us a better tour than the cheapest-link algorithm.*

Sometimes the cheapest-link algorithm produces a better tour than the nearest-neighbor algorithm, but just as often, it's the other way around.

*The two algorithms are different but of equal standing – neither one can be said to be superior to the other one in terms of the quality of the tours it produces.*

# Surprise Two

*The second surprise is that the nearest-neighbor tour A, P, W, H, G, I, N, A turns out to be an optimal tour. (This can be verified using a computer and the brute-force algorithm.)*

Essentially, this means that in this particular example, the simplest of all methods happens to produce the optimal answer–a nice turn of events. *Too bad we can't count on this happening on a more consistent basis!*