Review:

Hamilton path: a path that travels through every **vertex** of a graph **once and only once.**

Hamilton circuit: a circuit that travels through every **vertex** of a graph **once and only once.**

complete graph: there is exactly one edge connecting each pair of vertices; the complete graph with n vertices is written Kn.

Kn has a total of $n(n-1)/2$ edges.

factorials: $n!=1 \times 2 \times 3 \times \ldots \times n$.

Kn has (n-1)! Hamilton circuits.

10!= 3628800
20!= 2432902008176640000
30!= 265252859812191058636308480000000
40!= 815915283247897734345611269596115894272000000000
Notice how quickly these numbers become huge!

Traveling-salesman problems.

We have a weighted complete graph: each edge has a number called the weight attached to it. Each path has a total weight, the sum of the weights of the edges in the path.

So the "traveling-salesman problem", **TSP** for short, is to find the Hamilton circuit with the **smallest total weight.**

The brute-force algorithm:

1. list all Hamilton circuits.
2. calculate the total weight of each one.
3. choose the one which has least total weight.

The nearest-neighbor algorithm:

1. Choose any starting vertex.
2. Whenever you reach any vertex, look at the weights of all the edges that lead to vertices you haven't visited yet. Choose the one of least weight.
3. Once you've reached the last vertex, go back to the starting point.

Advantages and disadvantages of the algorithms

The brute-force algorithm is **optimal**: it gives the best possible answer. The nearest-neighbor algorithm is **approximate**: it gives an answer that is usually pretty good, but is not always the best possible.

The brute-force algorithm is **inefficient**: as the number of vertices increases, the amount of work we have to do to find the answer increases very rapidly (on the order of n!). The nearest-neighbor algorithm is **efficient**: the amount of work we have to do to find the answer increases, but not very rapidly (on the order of n^2— any polynomial rate of increase is considered efficient).

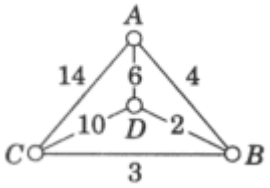
There are other optimal algorithms (which we won't learn), and there are other efficient algorithms (one or two of which we will learn), but no one has yet found an algorithm which is both optimal and efficient. Most experts think it can't be done. It is closely related to a famous problem (the **P=NP** problem) which has a million dollar reward for its solution (see the Millenium Prizes offered by the Clay Foundation at www.claymath.org).

Section 7. The repetitive nearest-neighbor algorithm.

The nearest-neighbor algorithm depends on what vertex you choose to start from. The **repetitive nearest-neighbor algorithm** says to try each vertex as starting point, and then choose the best answer.

Example.

A garbage truck must pick up garbage at four different dump sites (*A, B, C,* and *D*) as shown in the graph below, starting and ending at *A*. The numbers on the edges represent distances (in miles) between locations. The truck driver wants to minimize the total length of the trip.



Start at A: A, B, D, C, A has total length 4+2+10+14=30.

Start at B: B, D, A, C, B has total length 2+6+14+3=25.

Start at C: C, B, D, A, C has total length 3+2+6+14=25.

Start at D: D, B, C, A, D has total length 2+3+14+6=25.

So we can choose any of the solutions which have total length 25 (they all are the same Hamilton circuit, except they have different starting points, and some are mirror-images). You can rewrite the Hamilton circuit to start and end at A if you wish: A, C, B, D, A (or its mirror image A, D, B, C, A).

Note that the Hamilton circuit A, B, C, D, A has total length 4+3+10+6=23. The brute-force algorithm gives this answer.

Section 7. The cheapest link algorithm.

In this method we don't choose a starting vertex.

Instead we choose the “cheapest link” = the edge of smallest weight in the graph.

Then we chooses the edge of second smallest weight .(this edge doesn't need to share a vertex with the previous edge).

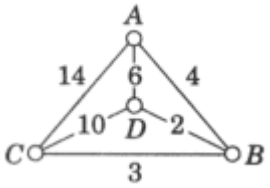
We keep doing this, except we reject any edges which either:

1) form a “short circuit” (a circuit which is not a Hamilton circuit”, or
2) result in 3 of our edges meeting at the same vertex.

Once we have chosen in this way n-1 edges, the edges will form a Hamilton path.

Then for our last edge we choose the one joining the two endpoints.

Consider the previous example:



Edges increasing order of weight: BD, BC, BA(but we reject this edge: it results in 3 edges meeting at B), AD; we have 3 edges so we have a Hamilton path: ADBC; we complete it to a Hamilton circuit: ADBCA, which has total weight 25, not best possible but as good as the last method.

This method is approximate and efficient.