The Traveling Salesperson Problem (Approached by Dynamic Programming)

Now we will return to a famous problem, the traveling salesperson problem (TSP), known in the sexist days when it was first defined as the "traveling salesman problem," and tackle it by using the dynamic programming algorithm design technique.

Here is the problem. Given a directed, weighted graph, find the total weight of the shortest/cheapest/best (actually, if we call the cost of going along an edge the "weight," the correct English term would be "lightest," but nobody says that) path that starts at v_1 and visits each other vertex exactly once, ending up back at v_1.

> The choice of v_1 is arbitrary—any of the vertices could be viewed as the starting point. For example, the path $v_1 \to v_3 \to v_5 \to v_2 \to v_4 \to v_1$ has the same cost as $v_5 \to v_2 \to v_4 \to v_1 \to v_3 \to v_5$.

Such a path is known as a "Hamiltonian circuit" of the graph, or a "tour." The TSP is to find the best Hamiltonian circuit.

> Typically we will behave as if the vertices are named 1, 2, …, n, rather than v_1, v_2, and so on, just to save time writing.

Earlier you solved this problem by brute force. This brute force approach had efficiency in $\Theta((n-1)!)$. The dynamic programming approach will have efficiency in $\Theta(n^2 2^{n-1})$, which is still horrible, but way better than $(n-1)!$.

⇒ Run the program `BruteForceETSP` to see the brute force method in action. Click the mouse at some points to specify a *Euclidean Traveling Salesman Problem* (ETSP) instance, where the weight of the edge between two points is the distance between them, and then press `s` to solve, followed by the space bar to see each permutation, or `o` to automatically find the optimal solution.

> We will often study ETSP instead of TSP. The difference is that for ETSP, we just have to specify points in the plane, and the weights—the distances between the points—are automatically determined. ETSP is still a famously difficult problem, even though it has some properties that don't hold for a general TSP instance given by an arbitrary weights matrix. For example, in ETSP, the weight of the edge from j to k is the same as the weight of the edge from k to j, because they are both just the distance between the two points, but there is nothing that says those two weights have to be the same in the general TSP problem. Also, in ETSP, all the weights are finite, whereas in TSP we could allow (like in Floyd's algorithm) infinite weight edges between vertices, corresponding to there not being an edge between them. In ETSP, there's always an edge between any two vertices.

Now we want to develop a dynamic programming approach to this problem, based on the chart of subproblems defined by:

> Let $D[A][j]$ be the length of a shortest path that starts at v_j, passes through each vertex in A exactly once, and then goes to v_1, where A is a non-empty subset of $\{v_2, \ldots, v_n\} - \{v_j\}$.

Given this definition of D, we need to figure out the answers to all the standard dynamic programming questions to create an algorithm.

First we note that if we have all the cells of this chart filled in, then we can solve TSP by considering each $j > 1$, and computing $w_{1j} + D[\{v_2, v_3, \ldots, v_n\} - \{v_j\}][v_j]$. This quantity is the length of the best path that goes from v_1 to v_j and then passes through all the other vertices exactly once before going back to v_1. So, we can just take the best of these quantities to find the best tour.

The base cases in the chart are obviously the cells for subsets $A = \{v_k\}$ with the best value for such cells being $w_{jk} + w_{k1}$, because the only choice is to go from v_j to v_k and then go from v_k to v_1.

Now we need to figure out the crucial recursive relationship that will allow us to fill a cell assuming that related simpler cells have been filled. To do this, we just need to consider all the possible first edges from v_j to a vertex in A. For each $v_k \in A$, other than for $k = j$, $w_{jk} + D[A - \{v_k\}][j]$ is the cost of the best path that goes from v_j to v_k and then passes through each other vertex in A before going to v_1. So, we simply take the best of these values as the value of $D[A][j]$. Note that this idea is really the same as the idea behind finding the best tour once we have filled the chart.

The efficiency category for this algorithm is $\Theta(n^2 2^{n-1})$ because the chart will have a row for each subset of the $n-1$ vertices other than v_1, will have $n-1$ columns, and computing the value of a cell will take, at worst, $O(n)$ additions.

$\Rightarrow$ Let's work through computing $D[\{2, 3, 5\}][4]$:

> We have three choices for where to go first from vertex 4, namely 2, 3, or 5. For each of these choices, we can find the cost, almost, from cells in the chart that have already been filled in. For example, the best path if we go from 4 to 2 costs
> $$w_{41} + D[\{3, 5\}][2],$$
> because $D[\{3, 5\}][2]$ is the cost of the best way to start at 2, pass through 3 and 5 exactly once, and then go to 1. We can do the same calculation for first going to 3, or first going to 5, and pick the best of these three paths to give the value for $D[\{2, 3, 5\}][4]$.

$\Rightarrow$ For a little more experience with the definition of D, say for $n = 6$, let's state in English the meaning of $D[\{2, 4, 5, 7\}][3]$, and then draw little pictures to justify expressing this cell in terms of other, simpler chart cells.

Example of Dynamic Programming Solution to a TSP Instance

Consider this instance of TSP:

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 8 & 3 & 4 & 7 \\ 2 & 9 & 0 & 2 & 5 & 10 \\ 3 & 3 & 11 & 0 & 4 & 9 \\ 4 & 7 & 8 & 12 & 0 & 5 \\ 5 & 12 & 9 & 6 & 7 & 0 \end{array}$$

D

$D[\{\}][2]$ $\quad 2 \xrightarrow{\vdash c \dashv} 1$

~~$D[\{2\}][2]$~~ 3 $\quad$ ~~$2 \to 2 \to 1$~~

$D[\{2\}][3]$ $\quad 3 \xrightarrow{11} \boxed{2 \xrightarrow{9} 1}$

$D[\{2\}][4]$

$4 \xrightarrow{8} \boxed{2 \to 1}$ $\quad b = D[\{3\}][2] + w_{32}$

$b = 9 + 11$

⇒ Let's fill in this chart (or as much of it as we can stand to) for this instance:

A

A	cost j = 2	3	4	5
{}	9 (0)	3 (0)	7 (0)	12 (0)
{2}	___	20 (2)	17 (2)	12+9 21 (2)
{3, 4, 5}				

$2 \to \{3,4,5\} \to 1$: $2 \to 3 \to \{4,5\} \to 1$

$2 \to 4 \to \{3,5\} \to 1$

$2 \to 5 \to \{3,4\} \to 1$

⇒ Now that we have the chart, let's solve the instance. If we get bored, tired, or confused, we can use `DynProgTSP` to help.

Exercise 22 [10 points] (target due date: Monday, October 19)

Here are the weights for an instance of TSP:

	1	2	3	4	5	6	7
1	0	7	2	11	14	9	5
2	15	0	2	1	8	6	12
3	4	18	0	21	7	13	19
4	6	2	11	9	8	15	29
5	14	17	6	15	10	4	3
6	17	12	23	13	6	12	24
7	7	12	11	6	25	4	3

Here, and continued on the next pages, is the chart for this instance. The contents of six cells (indicated by question marks) have been redacted. Your first job is to compute the scores in each of these cells, along with the vertex to go to first, following the given format. Write out carefully how you computed the score. Your second job is to compute the best tour for this instance. Again, explain your work carefully.

```
            2         3         4         5         6         7
     {} 15.00( 0)  4.00( 0)  6.00( 0)     ?     17.00( 0)  7.00( 0)
    {2} --------- 33.00( 2) 17.00( 2) 32.00( 2) 27.00( 2) 27.00( 2)
    {3}  6.00( 3) --------- 15.00( 3) 10.00( 3) 27.00( 3) 15.00( 3)
  {2,3} --------- ---------  8.00( 2) 23.00( 2) 18.00( 2) 18.00( 2)
    {4}  7.00( 4) 27.00( 4) --------- 21.00( 4) 19.00( 4) 12.00( 4)
  {2,4} --------- 25.00( 2) --------- 24.00( 2) 19.00( 2) 19.00( 2)
  {3,4} 16.00( 4) --------- --------- 30.00( 4) 28.00( 4) 21.00( 4)
{2,3,4} --------- --------- --------- 23.00( 4) 21.00( 4) 14.00( 4)
    {5} 22.00( 5) 21.00( 5) 22.00( 5) --------- 20.00( 5) 39.00( 5)
  {2,5} --------- 39.00( 5) 24.00( 2) ---------     ?     34.00( 2)
  {3,5} 18.00( 5) --------- 18.00( 5) --------- 16.00( 5) 32.00( 3)
{2,3,5} --------- --------- 20.00( 2) ---------     ?     30.00( 2)
  {4,5} 23.00( 4) 28.00( 5) --------- --------- 27.00( 5) 28.00( 4)
{2,4,5} --------- 31.00( 5) --------- --------- 30.00( 5)     ?
{3,4,5} 19.00( 4) --------- --------- --------- 31.00( 4) 24.00( 4)
```

```
    {2,3,4,5} --------- --------- --------- --------- 29.00( 5) 26.00( 4)
          {6} 23.00( 6) 30.00( 6) 32.00( 6) 21.00( 6) --------- 21.00( 6)
        {2,6} --------- 40.00( 6) 25.00( 2) 31.00( 6) --------- 31.00( 6)
        {3,6} 32.00( 3) --------- 41.00( 3) 31.00( 6) --------- 31.00( 6)
      {2,3,6} --------- --------- 33.00( 6) 22.00( 6) --------- 22.00( 6)
        {4,6} 25.00( 6) 32.00( 6) --------- 23.00( 6) --------- 23.00( 6)
      {2,4,6} --------- 32.00( 6) --------- 23.00( 6) --------- 23.00( 6)
      {3,4,6} 34.00( 3) --------- --------- 32.00( 6) --------- 32.00( 6)
    {2,3,4,6} --------- --------- ---------     ?     --------- 25.00( 6)
        {5,6} 26.00( 6) 28.00( 5) 29.00( 5) --------- --------- 24.00( 6)
      {2,5,6} --------- 38.00( 5) 28.00( 2) --------- --------- 38.00( 2)
      {3,5,6} 22.00( 6) --------- 31.00( 6) --------- --------- 20.00( 6)
    {2,3,5,6} --------- --------- 24.00( 2) --------- --------- 33.00( 6)
      {4,5,6} 30.00( 4) 30.00( 5) --------- --------- --------- 31.00( 6)
    {2,4,5,6} --------- 30.00( 5) --------- --------- --------- 34.00( 4)
    {3,4,5,6} 32.00( 3) --------- --------- --------- --------- 35.00( 6)
  {2,3,4,5,6} --------- --------- --------- --------- --------- 30.00( 4)
          {7} 19.00( 7) 26.00( 7) 36.00( 7) 10.00( 7) 31.00( 7) ---------
        {2,7} --------- 37.00( 2) 21.00( 2) 30.00( 7) 31.00( 2) ---------
        {3,7} 27.00( 7) --------- 37.00( 3) 18.00( 7) 39.00( 7) ---------
      {2,3,7} --------- --------- 29.00( 2) 21.00( 7) 39.00( 2) ---------
        {4,7} 24.00( 7) 31.00( 7) --------- 15.00( 7) 36.00( 7) ---------
      {2,4,7} --------- 38.00( 7) --------- 22.00( 7) 34.00( 4) ---------
      {3,4,7} 33.00( 3) --------- --------- 24.00( 7) 45.00( 7) ---------
    {2,3,4,7} --------- --------- --------- 17.00( 7) 38.00( 7) ---------
        {5,7} 18.00( 5) 17.00( 5) 18.00( 5) --------- 16.00( 5) ---------
      {2,5,7} --------- 36.00( 2) 20.00( 2) --------- 30.00( 2) ---------
      {3,5,7} 19.00( 3) --------- 26.00( 5) --------- 24.00( 5) ---------
    {2,3,5,7} --------- --------- 21.00( 2) --------- 27.00( 5) ---------
      {4,5,7} 19.00( 4) 22.00( 5) --------- --------- 21.00( 5) ---------
    {2,4,5,7} --------- 29.00( 5) --------- --------- 28.00( 5) ---------
    {3,4,5,7} 24.00( 3) --------- --------- --------- 30.00( 5) ---------
  {2,3,4,5,7} --------- --------- --------- --------- 23.00( 5) ---------
        {6,7} 33.00( 7) 40.00( 7) 46.00( 6) 24.00( 7) --------- ---------
      {2,6,7} --------- 44.00( 6) 35.00( 2) 34.00( 7) --------- ---------
      {3,6,7} 42.00( 3) --------- 51.00( 3) 34.00( 7) --------- ---------
    {2,3,6,7} --------- --------- 44.00( 2) 25.00( 7) --------- ---------
      {4,6,7} 35.00( 7) 42.00( 7) --------- 26.00( 7) --------- ---------
    {2,4,6,7} --------- 42.00( 7) --------- 26.00( 7) --------- ---------
    {3,4,6,7} 44.00( 3) --------- --------- 35.00( 7) --------- ---------
  {2,3,4,6,7} --------- --------- --------- 28.00( 7) --------- ---------
      {5,6,7} 22.00( 6) 29.00( 6) 31.00( 6) --------- --------- ---------
    {2,5,6,7} --------- 40.00( 2) 24.00( 2) --------- --------- ---------
    {3,5,6,7} 30.00( 6) --------- 39.00( 6) --------- --------- ---------
  {2,3,5,6,7} --------- ---------     ?     --------- --------- ---------
    {4,5,6,7} 27.00( 6) 33.00( 5) --------- --------- --------- ---------
  {2,4,5,6,7} --------- 33.00( 5) --------- --------- --------- ---------
  {3,4,5,6,7} 35.00( 3) --------- --------- --------- --------- ---------
{2,3,4,5,6,7} --------- --------- --------- --------- --------- ---------
```

Exercise 23 [4 points] (target due date: Monday, October 19)

In this exercise you will be asked to develop a dynamic programming approach to a new problem, given the key observations.

Note that this exercise will require you to create a Java application that solves the sequence alignment problem. All of the following material is suggested pencil-and-paper work to do in order to prepare yourself to create the program.

Consider the problem of aligning two strings so that they match as well as possible. This problem arises in genetics, where a sequence of molecules named A, C, G, and T form a piece of DNA. As time goes on and various mutations happen, causing symbols to change, to be removed, or to be inserted, two formerly identical sequences in the population of a species become different, but not all that different.

> Apparently (according to Dr. Liu of our Biology Department) the model developed here is actually used by biologists.

As an example instance of the problem, suppose we have the sequences `AACAGTTACC` and `TAAGGTCA` and we want to align them so that as many symbols as possible match up. We could simply shift them against each other and try to find the shift so that as many symbols as possible match up, but the biology suggests that instead we should allow ourselves to insert any number of *gaps*, denoted by `-`, in each sequence, and try in this way to get lots of pairs of symbols to match up. We need a score for any given pair of sequences, with gaps inserted, so we can decide which is better, so we say that if two actual symbols don't match, we are penalized 1 point, and if an actual symbol is matched with a gap, then we are penalized 2 points.

For example, if we insert some gaps in our sequences, like so:

```
- A A C A G T T A C C
T A A - G G T - - C A
```

we would be penalized 8 points for the 4 actual symbols that are paired up with gaps, and 2 points for the 2 pairs of actual symbols that don't match, for a score of 10.

Or, if we inserted gaps like this

```
A A C A G T T A C C
T A A G G T - - C A
```

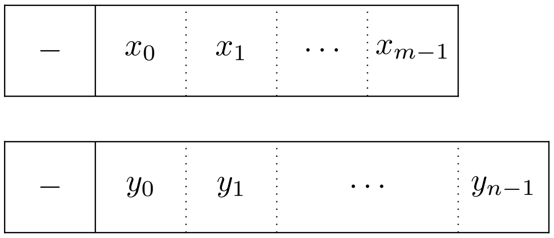
we would have a penalty of 4 for the two actual symbols that are paired with gaps, and a penalty of 4 for the four pairs of mismatched actual symbols, for a score of 8, so this alignment beats the first one.

We can now state the sequence alignment problem: given two strings of actual symbols, find a way to insert gaps that produces the smallest score.

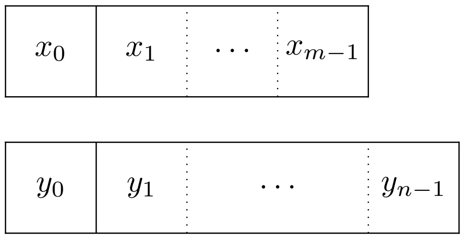
Now we will state the key recursive connection between the original problem and sub-problems with the same structure, and you will be asked to figure out the algorithm from this idea.

Consider strings $x_0x_1x_2\ldots x_{m-1}$ and $y_0y_1y_2\ldots y_{n-1}$ for positive integers m and n. The key recursive idea is simply to look at all possible decisions for the first pair of symbols, and then recursively figure out the optimal way to arrange the rest.

First, we might put gaps in both strings in the first position. This would be silly, making no progress—leaving us with the same problem we started with as the sub-problem to be solved:

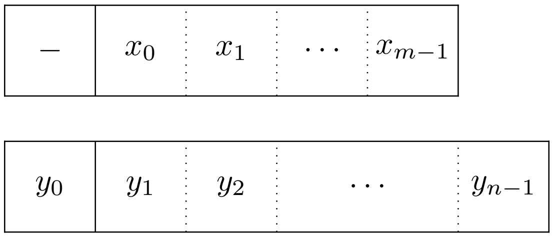


Or, we might chose to pair up x_0 and y_0:



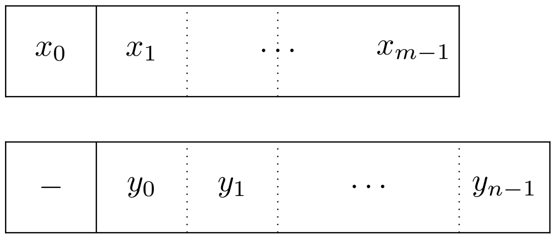
This would produce a penalty of 0 or 1, depending on whether $x_0 = y_0$. Then, we would recursively decide how to best insert gaps in $x_1x_2\ldots x_{m-1}$ and $y_1y_2\ldots y_{n-1}$, and add whatever best penalty is obtained for that subproblem to 0 or 1.

Or, we could chose to insert a gap in the first string, matching it with y_0:



for a penalty of 2, leaving us to recursively solve the sub-problem of optimally inserting gaps in $x_0x_1\ldots x_{m-1}$ and $y_1y_2\ldots y_{n-1}$.

Similarly, we could chose to insert a gap in the second string, matching it with x_0:



for a penalty of 2, and recursively solve the sub-problem of optimally inserting gaps in $x_1 x_2 \ldots x_{m-1}$ and $y_0 y_1 \ldots y_{n-1}$.

Now, we could simply implement this algorithm using recursion directly, but as usual, this would lead to sub-problems being solved repeatedly and a high degree of inefficiency, so as usual we look to build a table holding solutions to all the appropriate sub-problems. To do this, we simply note that the sub-problems we are talking about above, and the original problem, all have the same form, namely "find the optimal way to insert gaps in $x_i \ldots x_{m-1}$ and $y_j \ldots y_{n-1}$, where $0 \leq i < m$ and $0 \leq j < n$.

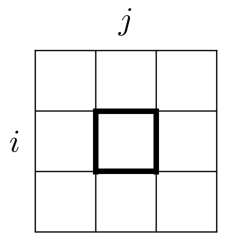
So, we create a grid of cells, one for each pair of values i and j, like so, for the given problem:

	T	A	A	G	G	T	C	A	-
A									
A									
C									
A									
G									
T									
T									
A									
C									
C									
-									

where we have added an extra gap at the end of each string to provide a convenient way to put the base cases in the chart. For consistency we say that matching two gaps has a penalty of 0.

Now you need to figure out how to compute the value in each cell in terms of already filled-in cells.

Here is a diagram showing the cell to be filled in, namely $A(i, j)$ ("A" for alignment?), and some neighboring cells. Figure out a precise description (formula, algorithm, or whatever) for how to compute $A(i, j)$ using the values in relevant neighboring cells, which as always with dynamic programming represent simpler sub-problems than $A(i, j)$.



Now, use your description to fill in all the cells in the chart below. Note that you will need to fill in base case cells "from scratch," that is, not using the same number of cells as the general case. In addition to writing the score in each cell, you must draw an arrow or otherwise indicate which of the three possible decisions is made to produce the optimal score for the cell.

	T	A	A	G	G	T	C	A	-
A	7	8	10	12	13	15	16	18	20
A	6	6	8	10	11	13	14	16	18
C	4	5	6	8	9	11	12	14	16
A	6	4	4	6	7	9	11	12	14
G	7	5	3	4	5	7	9	10	12
T	9	7	5	3	3	5	7	9	10
T	8	8	6	4	3	3	5	7	8
A	10	8	7	5	3	2	3	5	6
C	12	10	8	6	4	2	1	3	4
C	14	12	10	8	6	4	2	1	2
-	16	14	12	10	8	6	4	2	0

Once the chart is filled in, write down the optimal way of aligning the two sequences, and verify that the penalty is correct.

Demonstrate (draw your own tidy chart) your dynamic programming algorithm for the instance of the sequence alignment problem with the sequences `GACATATTAC` and

$$
\begin{array}{c|c|cccccccccc}
 & & 1 & 2 & 3 & 4 & 5 & 6\ j & 7 & 8 & 9 & \\
 & & & T & A & A & G & G & T & C & A & - \\
\hline
1 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & A & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
3 & A & 0 & 1 & 2 & 2 & \square & & & & & \\
4 & C & 0 & & & & & & & & & \\
i\ 5 & A & 0 & & & & & & & & & \\
6 & G & 0 & & & & & & & & & \\
7 & T & 0 & & & & & & & & & \\
8 & T & 0 & & & & & & & & & \\
9 & A & 0 & & & & & & & & & \\
10 & C & 0 & & & & & & & & & \\
11 & C & 0 & & & & & & & & & \\
 & - & 0 & & & & & & & & &
\end{array}
$$

$M_{i-1,j-1} + (S_i, T_j)$

$M_{3-1,5-1} + (1)$

$M_{24} + 1$

$(d_{ij} = d_{23} = 1) > (d_{ik} = d_{32} = 1) + (d_{kj} = d_{24} = 1)$

$1 > 1 + 1$

$1 > 2$ false

$= \boxed{2}$

~~$AA = d_{ij} > d_{ik} + d_{kj}$~~

~~$= d_{34} > d_{33} + d_{34}$~~

~~$= 0 > 2 + 1$~~

~~$= 0 > 3$ false $= 3$~~

~~$d_{3-1,4-1} > d_{3-1,3} + d_{3,4-1}$~~

~~$d_{23} + d_{33}$~~

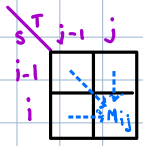
~~$d_{23} > d_{23} + d_{33}$~~

~~$1 > 1 + 2$~~

~~$1 > 3$~~

$d_{ij} > d_{ik} + d_{kj}$

$AT = d_{22} > d_{11} + d_{11} = 0 > 0 + 0 = \underline{1 ? \ldots}$



$$M_{i,j} = Max \begin{cases} M_{i-1}, M_{j-1} + Score(S_i, T_j) \\ M_{i,j-1} + \gamma \\ M_{i-1,j} + \gamma \end{cases} \rightarrow \underline{\text{gap penalty}}$$

Score (match) = 1

Since the condition is, $M_{ij} > M_{ik} + M_{kj}$ all equal 0 then we increment the value of the cell M_{ij} by 1 instead of writing a zero

$AA = 0 > 1 + 1$ false so $M_{ij} = d_{ik} + d_{kj}$
$= \boxed{2}$

`AACGTAGAC`. Fill in the chart and clearly state the optimal insertion of gaps that gives the optimal score.

All of the preceding work was preparation for the actual work of this Exercise, which is to create a Java application that will solve the sequence alignment problem.

Specifically, create a Java application that will ask the user to enter two strings, will generate and fill in the dynamic programming chart and display it, and will then display in a nice way exactly how the gaps should be optimally inserted and report the optimal score.

Submit your work by email as usual, with your preparatory work shown (at a minimum turn in pictures of the two charts you are asked to fill in), and all the Java source files for your program attached (either individually or, preferably, as a single `zip` file), where `Ex23` is the main class.