CS2010 PS7 - A Trip to Supermarket v2 (with R-option)

Released: Wednesday, 24 October 2012 (Happy 1st Birthday for Jane)
Due: Saturday, 03 November 2012, 8am

Collaboration Policy. You are encouraged to work with other students or teaching staffs (inside or outside this module) on solving this problem set. However, you **must** write the Java code **by yourself**. In addition, when you write your Java code, you **must** list the names of every collaborator, that is, every other person that you talked to about the problem (even if you only discussed it briefly). This list may include certain posts from fellow students in CS2010 IVLE discussion forum. Any deviation from this policy will be considered cheating, and will be punished severely, including referral to the NUS Board of Discipline. It is not worth it to cheat just to get 15% when you will lose out in the other 85%.

R-option. This PS has R-option at the back. This additional task usually requires understanding of data structure or algorithm *beyond* CS2010. A self research on those relevant additional topics will be needed but some pointers will be given. CS2010R students *have to* attempt this R-option. CS2010 students can choose to attempt this R-option too for higher points, or simply leave it.

Last Year's Story – that still continues until now. Before Jane's birth, Grace could not travel far and could not carry heavy stuffs (see PS4). After Jane's birth, Grace needs a lot of time to rest and to breastfeed Jane. Therefore, since last year, Steven has to help Grace with a certain household chore that he thought he does not need to do anymore after he got married: Grocery Shopping (To the guys out there, get ready :D).

The Actual Problem. This time, Steven has to visit a supermarket at Clementi area (name omitted to avoid indirect advertising). Steven has visited this place numerous times and therefore he knows the location of various N items in that supermarket. Steven has even estimated the *direct* walking time from one point to every other points in that supermarket (in seconds) and store that information in a 2D table `T` of size $(N+1) \times (N+1)$. Given a list of K items to be bought today, he wants to know what is the minimum amount of time to complete the shopping duty of that day. As mentioned several time in lectures, we have to make some simplifying assumptions in order for this problem to be solvable... :

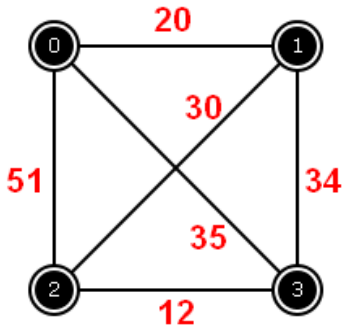
1. Steven always starts at vertex 0, the entrance (+ cashier section) of that supermarket.
2. There are $V = N + 1$ vertices due to the presence of this special vertex 0.
   The other N vertices corresponds to the N items. The vertices are numbered from `[0..`N`]`.
3. The direct walking time graph that is stored in a 2D table `T` is a **complete graph**.
   `T` is a symmetric square adjacency matrix with `T[i][i] = 0` $\forall$`i` $\in$ `[0..`N`]`.
4. Steven has to grab all K items (numbered from `[1..`K`]`) that he has to buy that day, or he will have hard time explaining (to Grace) why he does not buy certain items... $1 \leq K \leq N$.
5. Steven is very efficient, once he arrives at the point that stores item i, he can grab item i into his shopping bag in 0 seconds (e.g. for item 'banana', he does not have to compare the price

of 'banana A' versus 'banana B' and he does not have to select which banana looks nicer, etc...). So, Steven's shopping time is only determined by the total walking time inside that supermarket to grab all the K items.

6. In this problem, Steven ends his shopping duty when he arrives at cashier (also at vertex 0).

7. Steven can grab the K items in **any order**, but the total walking time (i.e. time to walk from vertex 0 $\rightarrow$ to various points in the that supermarket in order to grab all the K items $\rightarrow$ back to vertex 0) **must be minimized**. Steven can choose to bypass a certain point that is *not in his shopping list* or even *revisit* a point that contains item that he already grab (he does not need to re-grab it again) if this leads to faster overall shopping time.

See below for an example supermarket:



	0	1	2	3
0	0	20	51	35
1	20	0	30	34
2	51	30	0	12
3	35	34	12	0

2D Table T

Figure 1: A Sample Supermarket Layout, $N = 3$, $V = 3 + 1 = 4$

If today, Steven has to buy all item 1, item 2, and item 3, then one of the best possible shopping route is like this: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0$ with a total walking time of: 20+30+12+35 = 97 seconds.

If tomorrow, Steven has to buy only item 1 and item 2, then one of the best possible shopping route is still: $0 \rightarrow 1 \rightarrow 2(\rightarrow 3) \rightarrow 0$ with a total walking time of: 20+30+(12+35) = 97 seconds. Notice that although Steven does not have to buy item 3, taking sub path $2 \rightarrow 3 \rightarrow 0$ (where Steven will just bypass item 3) is faster than taking sub path $2 \rightarrow 0$.

If two days later, Steven has to buy only item 2, then one of the best possible shopping route is like this: $0(\rightarrow 3) \rightarrow 2(\rightarrow 3) \rightarrow 0$ with a total walking time of: (35+12)+(12+35) = 94 seconds. Steven bypass through item 3 twice!

If three days later, Steven has to buy only item 2 and item 3, then one of the best possible shopping route is like this: $0 \rightarrow 3 \rightarrow 2(\rightarrow 3) \rightarrow 0$ with a total walking time of: 35+12+(12+35) = 94 seconds. See that Steven can revisit point 3 although he does not have to grab item 3 twice.

These four examples should be clear enough to describe this problem.

The skeleton program `Supermarket.java` is already written for you, you just need to implement one (or more) method(s)/function(s):

- `int Query()`
  You are given a 2D matrix `T` of size $(N + 1) \times (N + 1)$ and an array `shoppingList` of size K. Query these two data structures and answer the query as defined above.

- If needed, you can write additional helper methods/functions to simplify your code.

Subtask 1 (25 points). The supermarket is a small supermarket and everything there have to be grabbed/bought. $1 \leq K = N \leq 9$.

Subtask 2 (Additional 25 points). The supermarket is slightly bigger than in Subtask 1 and everything there have to be grabbed/bought. $1 \leq K = N \leq 15$.

Subtask 3 (Additional 25 points). The supermarket has the same size as in Subtask 2, but only a subset of K out of N items have to be grabbed/bought. $1 \leq K \leq N \leq 15$.

Subtask 4 (Additional 10 points). Typical supermarket size that sells $\approx$ *hundreds* of grocery items. However, Grace only asks Steven to buy some items. $1 \leq N \leq 200$, $1 \leq K \leq 15$, $K \leq N$.

Subtask 5 (Additional 15 points). Typical supermarket size that sells $\approx$ *thousands* of grocery items. However, Grace only asks Steven to buy some items. $1 \leq N \leq 1000$, $1 \leq K \leq 15$, $K \leq N$.

Note: The test data to reach 75 points: `Subtask1.txt`, `Subtask2.txt`, and `Subtask3.txt` as well as `Sample.txt` are given to you. You are allowed to check your program's output with your friend's. You are encouraged to generate and post additional test data in IVLE discussion forum. The simple checker program `SupermarketVerify.java` is attached. This program will test if:

1. The shopping items are within [1..N].
2. The given matrix contains non-negative integers and symmetric.

R-option (110 points). This is a similar *but actually different* problem compared to the normal PS7. Students who want to attempt the R-option have to write a different `SupermarketR.java`. The skeleton program `SupermarketR.java` is given. They key differences are highlighted below:

- Instead of 2D matrix `T`, you are given two arrays `X` and `Y` each containing $(N + 1)$ integers. The pair `(X[i], Y[i])` denotes the location of item i in a 2D plane.
  We guarantee that `X[i - 1]` $<$ `X[i]` $\forall$`i` $\in$ `[1..N]` and no two items have the same `X` value.
- The time to walk from item i to another item j is the *Euclidean distance* between them, i.e. $\sqrt{(X[i] - X[j])^2 + (Y[i] - Y[j])^2}$. Use double data type to store these Euclidean distances and do not do any rounding in computing these values.
- In this R-option, you have to visit (and bought) all items, i.e. $K = N$. You have to start from the leftmost item 0 (as this item 0 has the lowest `X[0]`), **go strictly "left to right"** to the rightmost item N (as this item N has the highest `X[N]`), and then **go strictly "right to left"** back to item 0 to conclude your supermarket tour.
- K is not given as $K = N$, the array `shoppingList` is also not given as everything must be grabbed/bought.
- Constraints: $1 \leq N \leq 1000$.

For example, look at Figure 2, left. Here we have 7 vertices. Vertex/item 0 is located at the leftmost side and vertex/item $N = 6$ is located at the rightmost side. All vertices/items have different `X` values. The standard Traveling Salesman Problem (TSP) solution produces Figure 2, middle. But this is not what we want, because path $0 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 4 \rightarrow \underline{1 \rightarrow 2} \rightarrow 0$ is not a "left to right" then "right to left" path, because at vertex/item 1, we go right again to vertex/item 2: $1 \rightarrow 2$. Figure 2, right is what we want. Notice that path $(0 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow (6) \rightarrow 4 \rightarrow 1 \rightarrow 0)$ can be decomposed into "left to right" sub-path and "right to left" sub-path.

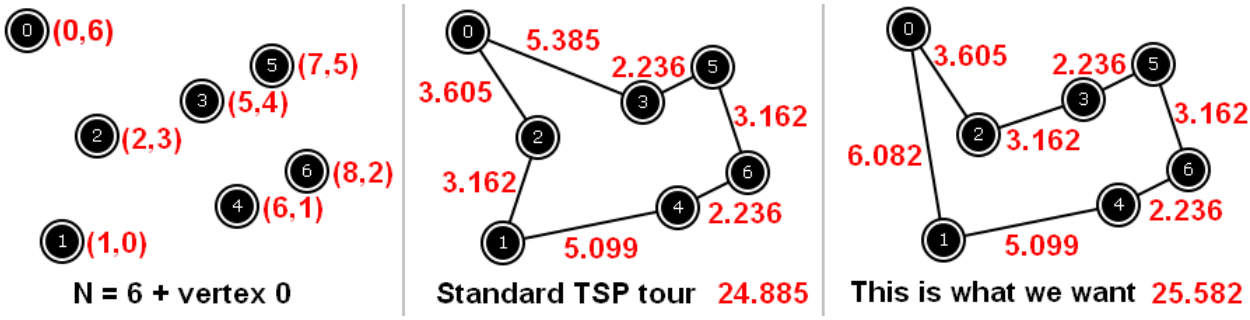


Figure 2: Supermarket (R-option), $N = 6$, $V = 6 + 1 = 7$

As the solution of this problem is totally different from the normal PS7 and the time needed to complete the R-option can be significant if you have not seen this problem before, non CS2010R students can choose to attempt the R-option only (and get 110 points if it is correct) or choose to go safe with the normal PS7 (and get at most 100 points).

PS: Minor floating point error (if any) will be ignored by lab TA during manual grading. For example, a more precise number for Figure 2, right is 25.5840246. If your answer is very close to this number (within 0.1 precision), lab TA will accept your solution, i.e. you can answer any value between [25.4840246 .. 25.6840246].

A test data verifier program `SupermarketRVerify.java` is attached. This program will test if:

1. X values are given in increasing order.
2. No two items have the same X value.