

COS 126

Traveling Salesperson Problem

Step 0: preparation (0 points)

- Read the assignment in the course packet to get an overview of the problem.
- Read the assignment checklist (on the course website) for some good hints and additional information & details about what to submit.
- To learn about circular linked lists, re-read Sedgewick: sections 3.3 - 3.4; in particular, table 3.1 contains a useful code fragment for traversing a circular linked list.
- Copy the files you will need for the assignment from the checklist page:
tsp4.txt, tsp10.txt, tsp10.ans, tsp100.txt, tsp1000.txt, POINT.h, point.c, TOUR.h, tour.c, and tsp.c

Alternatively, you can use the following commands to get the files:
(first create a directory for the tsp assignment and use cd to change into that directory)

```
cp /u/cs126/code/POINT.h .
cp /u/cs126/code/point.c .
cp /u/cs126/code/TOUR.h .
cp /u/cs126/code/tour.c .
cp /u/cs126/code/tsp.c .
cp /u/cs126/data/tsp* .
```

Step 1: tour.c implementation (0 points)

- If you try to compile your program with the following command, you will get an error:
lcc tsp.c tour.c point.c
- The compiler reports that TOURdist and TOURshow are undefined.
- So, you should add both of these functions to tour.c:
void TOURshow()
{
 printf("in TOURshow()\n");
}

float TOURdist()
{
 float distance = 0.0f;
 printf("in TOURdist()\n");
 return distance;
}
- These functions don't really do anything, but by adding the code to tour.c, you will be able to compile your program.
- Once you compile the file, run it with one of the text files: a.out < tsp10.txt
result:

```
in TOURshow()
in TOURdist()
total distance = 0.000000
```

Step 2: writing the TOURshow() function (2 points)

- For this function, you will need to write code that traverses the tour list and prints the point stored at each node of this list
- Note that the global variable 'tour' (of type link) points to the first node on the list.

- You should use the function POINTprint (point p) to print each point. (read the code provided in point.c)
- A good way to start is to try to write a single line of code that prints just the first point on the tour. (Test your code with any of the tsp.txt files.)
- You may want to use a do- while loop to print the whole list. (see D&D, page 118)
- When your function is working properly, you should get the following output for tsp4. txt


```
0. 1791  0. 7507
0. 3945  0. 6792
0. 1384  0. 5627
0. 4106  0. 8469
in TOURdist()
total distance = 0. 000000
```
- Notice that the points seem to be out of order. That's because the TOURinsert function is inserting each new point *between the first and second nodes*. So, if the order of the points in the input file is A B C D, the order of the points you will traverse is A D C B.

Step 3: writing the TOURdist() function (2 points)

- The code for this function will be similar to that of TOURshow(). You need to write code that traverses the list, calculates the distance between the points stored in each pair of adjacent nodes, and returns the sum of all those distances.
- Note that you do **not** need to *print* the distance calculated; just return the value;
- You should use the function POINTdist (point p, point q) to calculate the distance between a pair of points. (read the code provided in point.c)
- Again, if you're having trouble getting started, just write some code to return the distance between the first and second points on the list. (correct result for tsp4.txt: 0.226888)
- When your function is working properly, you should get the following output for tsp4. txt


```
0. 1791  0. 7507
0. 3945  0. 6792
0. 1384  0. 5627
0. 4106  0. 8469
total distance = 1. 152534
```
- If your total is higher, check to see that you're not adding any pairs more than once. If it's lower, you may be forgetting to add the distance between the last point on the list and the first point (don't forget that the list is circular - the traveling salesman has to return home!)
- Debugging ideas: each time through the loop, print the distance between the current 2 points and print the sub-total so far. For 4 points (A,B,C,D), there should be 4 distances added: (A-B, B-C, C-D, D-A). Another thing you can do is create a very simple tsp.txt file. For example, just put 2 points in it: (0.0, 1.0) & (1.0, 1.0). The distance for that tour should be 2.

Step 4: writing tour-nearest.c (3 points)

- Make a copy of your tour.c file named tour-nearest.c
- The only function you will need to change is TOURinsert(point p)
- Like TOURshow() and TOURdist(), the code for this function will also involve a traversal of the circular tour list.
- For each node of the list, you will calculate the distance between the point p and the point stored at the node. As you traverse, you need to keep track of both the lowest distance calculated so far *and* the location of the node for which that distance was calculated.

- When you've finished traversing the whole list, you need to create a new node for the point p and insert that node at the place where you found that least distance. (right *after* the node for which the distance was least)
- IMPORTANT: think about what should happen if the point p is the *first* point you're inserting on the list. And, don't forget to make sure that the list you create is circular - the last node should always point to the first.
- When your function is working properly, you should get the following output for tsp4.txt


```
0.1791 0.7507
0.1384 0.5627
0.4106 0.8469
0.3945 0.6792
total distance = 0.981395
```
- Debugging ideas:
 1. Make sure you're compiling `tour-nearest.c` *not* `tour.c`
 2. Again, try using a simpler file of points: try an input file with just 1 point, then one with just 2 points. That should help you find the major problems. DRAW PICTURES on paper and work through the code by hand.
 3. In the beginning of the function, use `POINTprint(p)` to display the point you're trying to insert. Then, each time you calculate the distance between a point in a node on the list and p, use `POINTprint()` to display the point in the node, and print the distance calculated. For example, this is some sample debugging output for `tsp4.txt` (starting with the second point)


```
inserting point: 0.4106 0.8469
0.1791 0.7507 d: 0.250717
inserting point: 0.1384 0.5627
0.1791 0.7507 d: 0.192401

0.4106 0.8469 d: 0.393602
inserting point: 0.3945 0.6792
0.1791 0.7507 d: 0.226888

0.1384 0.5627 d: 0.281327

0.4106 0.8469 d: 0.168504
0.1791 0.7507
0.1384 0.5627
0.4106 0.8469
0.3945 0.6792
total distance = 0.981395
```
- Notice that since the point (0.1384, 0.5627) is closest to (0.1791, 0.7507), that's where we insert it. Then, (0.3945, 0.6792) is inserted after (0.4106, 0.8469).

Step 5: writing `tour-smallest.c` (2 points)

- Make a copy of your `tour-nearest.c` file named `tour-smallest.c`
- You will need to add/change a few lines of code in `TOURinsert(point p)`
- Instead of inserting the point at the place in the tour where its *closest* to another point, you will insert the point between two nodes so that the resulting increase in the tour distance is the least.
- Now, the *temptation* for this part of the assignment is to use the following algorithm (**which is *not* correct**):
 1. create a new node for point p
 2. insert the new node between the first two nodes on the list
 3. calculate the total new distance using `TOURdist()`

4. delete the new node from the list & insert it between the next pair of nodes
 5. repeat steps 2 through 4 for every pair of adjacent nodes on the tour
- Yes, that would work, but... it's extremely inefficient. Recall that your algorithm for calculating the distance of the whole tour involved traversing the list. So, in order to insert the Nth node into the tour, you will call TOURdist() N-1 times (once for each possible position) and as a result, you will traverse the list N-1 times. A good, efficient solution to this problem will involve only one traversal per point.
 - The correct algorithm for this part of the assignment is as follows:

For each point you want to insert, calculate the *change* in the distance of the tour that would occur if the point were inserted between the first two nodes on the list. Now, traverse through the rest of the list, and for each pair of adjacent nodes, calculate the distance that would be added if the new point were inserted between them. Just like in tour-nearest.c, you should keep track of the best position you've seen so far.

At the end, create a new node for the new point p and insert it at that optimal position.
 - Again, you should think about the special case of inserting the first node. Also, think about what happens when you want to insert the second node. DRAW PICTURES on paper.
 - When your function is working properly, you should get the following output for tsp4.txt


```
0. 1791  0. 7507
0. 1384  0. 5627
0. 3945  0. 6792
0. 4106  0. 8469
total distance = 0. 892949
```
 - Debugging : The same ideas from step 4 apply here. The file tsp10.ans contains the correct results for tsp10.txt. The total distance for tsp100.txt is 7.663055. For tsp1000.txt the total distance is 23.482428

Step 6: PostScript output (1 point, *not* 20%)

- Make a copy of your tour-smallest.c file named tourps-smallest.c
- When working on this part of the assignment, comment-out the line of code in tsp.c that prints the total distance.
- The only function you need to change for this part is TOURshow().
- Remember to start your function by printing, "%%! \n 50 50 translate..." and end the function by printing "closepath stroke showpage\n".
- Use the moveto command to move to the starting position (the first point on the tour). For every other point, use lineto to draw a line to that point. (Don't forget to connect the last point to the first!) See the course packet for example postscript output.
- Since all the points have values between 0.0 and 1.0, you will need to scale the points to fit the 512x512 postscript box.
- The only 'tricky' thing you will need think about is when to use '.' and when to use '->' to access elements of various data structures. Review your notes on this concept!
- The resulting picture will be circular within the 512x512 box, and very few of the lines will cross. If you have completely all the other parts of the assignment correctly, you shouldn't have any problems with this code.

Please contact Lisa Worthington <lworthin@cs.princeton.edu> if you find any typos/bugs in these instructions.