COS126: General Computer Science	Spring 1996
Problem Set # 5	
Handed out:March 12, 1996	*Due: March. 27, 1996*

Conceptually, this problem set continues where Problem Set # 4 left off. You will again write a program that tries to find a Travelling Salesman tour. But there are some new twists. (i) Your program will use the *Double Nearest Neighbor Heuristic* instead of the nearest neighbor heuristic. (ii) You will use linked lists instead of arrays to store the current partial tour. (iii) Your program will output *postscript* instructions to draw the tour.

Like the nearest neighbor heuristic, the double nearest neighbor heuristic also maintains a partial tour and at each step extends the partial tour to a point that has not been visited yet. The difference is that whereas the nearest neighbor heuristic can extend the current partial tour only at the endpoint that was just added, the double nearest neighbor heuristic can extend the current partial tour at either of the two endpoints of the partial tour. That is to say, the heuristic searches among all points that are not yet in the tour, and adds the one whose distance to either of those two end-points is the lowest.

As in Problem Set # 4, we are providing you with a partially-written program (see below) that already shows you how to represent the points, and how to represent the current partial tour. The symbolic constant `N` represents the number of points. The points are stored in a global array `cities`, which contains `N points`. The program also defines the structure `points` for you.

A function `initialize()` initializes the `cities` array using a random number generator.

The partial tour should be stored in a *linked list* called `tour`. The list is declared using the `node` and `list` structures.

The Assignment:

1. Complete the given partial program into a program to find a travelling salesman tour according to the double nearest neighbor heuristic.

2. Write a function `printtour` with the definition

   `void printtour(struct list *tour)`

   which takes as its input a pointer to a linked list containing a tour, and prints out postscript instructions for drawing this tour.

3. Run your program for $N = 100, 200, 400$ and print out the tour produced by your algorithm in each case. Submit these printouts with your hardcopy. For the case $N = 100$, also submit printouts of the partial tour when it contains $N/4$ and $N/2$ points.

How to direct output into postcript files:

Postscript is a computer language that is used for drawing pictures. Your precepts will discuss it in greater detail (see also the postscript chapter in the reader on pages 154-160).

To produce postscript output, your program uses the usual `printf` statements. But while running `a.out`, you direct your program's output into a file, say `tour.ps`, by typing `a.out > tour.ps`. You can examine the file `tour.ps` by typing `ghostview tour.ps &`. (Warning: if your file doesn't conform to postscript conventions, `ghostview` will return an error.) You can also print the file from the ghostview window. Finally, note that the character "`%`", which is used in postscript files, has a special meaning inside a `printf` statement in C. The C command to print a "`%`" is `printf("%%")`.

Submission:

Put your program in a file called `dnearest.c`. To electronically submit it, type

`/u/cs128/bin/submit 5 dnearest.c`

The deadline for electronic submission is **10pm on Wednesday, March 27**, and for hardcopy submission is the lecture on **Thursday, March 28**.

The Partial Program

(can be found in `/u/cs126/progs/tspprog2.c`)

```
#include <stdio.h>
#include <stdlib.h>        /*contains srandom() and random()*/
#include <math.h>          /*contains sqrt()*/
#define N 100
#define L 432
/*L= length of the box that contains all the cities */

struct point {
        double x;
        double y;
        int visited;
       };
struct point cities[N];

struct node {
```

```
        int index;
        struct node *next;
       };
struct list {
        struct node *head;
        struct node tail;
};
/* Modify the above definitions if you need to */

void initialize()
{
  int i;

  srandom(1);    /*set seed for random numbers */
  for(i=0; i<N; i++)
    {
      cities[i].x =  72 + random()%L;
      cities[i].y =  72 + random()%L;
      cities[i].visited =0;
    }
}
void main()
{
  struct list tour;
     /* define your variables here */

  initialize();

     /* put in your code after this point */
}
```