COS128: General Computer Science —Honors	**Spring 1996**

Problem Set # 5

Handed out:March 12, 1996 — *Due: March. 27, 1996*

In this problem set, you will implement another heuristic algorithm for the Travelling Salesman Problem. This algorithm uses the fact that we are solving instances of the Euclidean Travelling Salesman Problem. The double nearest neighbor heuristic did not make use of the fact that points were in the plane and distances were Euclidean.

Addition features of your solution to this assignment that will differ from that for Problem Set # 4 are: (i) points will now be represented by structs, (ii) you must use linked lists instead of arrays to store tours (linked lists were optional for Problem set # 4), (iii) Your program will output *postscript* instructions to draw the tour.

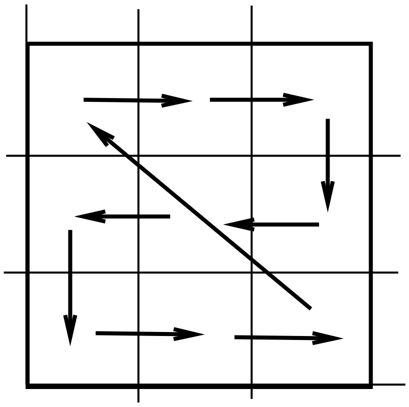
The new heuristic combines the double nearest neighbor heuristic with a partitioning of the plane. Using the `initialize` function given below (slightly different from that of Problem set # 4), the x and y coordinates of each point are numbers in the range from 0 to L; thus the points fall in a $L \times L$ square in the plane. The new heuristic technique is to partition this square into M smaller squares in grid fashion. In each grid square, use the double nearest neighbor heuristic to find a tour for the points *just within that smaller square*, but only until all points in the grid square are connected — don't include the last edge that would complete the cycle. Each tour in a grid square is called a *subtour.* We connect the subtours for neighboring grid squares to form one complete tour of all the points. In the main part of this assignment, you will use a simple zigzag pattern as shown below to connect the subtours. In the extra credit, you can try more complicated patterns.

In doing the partitioning of the square containing the points, we want neither too few squares, nor too many. Choosing M to be $\sqrt{N}$, for N points, is a good compromise (so there are M grid squares with on the average M points per square). Since we are making a $\sqrt{M} \times \sqrt{M}$ grid, M must also be a perfect square. We choose a perfect square, M, so that $M^2 \leq N$. For example, for $N = 100$ points to 256 points, we choose M to be $3^2 = 9$; for $N = 256$ to 625, we choose M to be $4^2 = 16$, and so on.

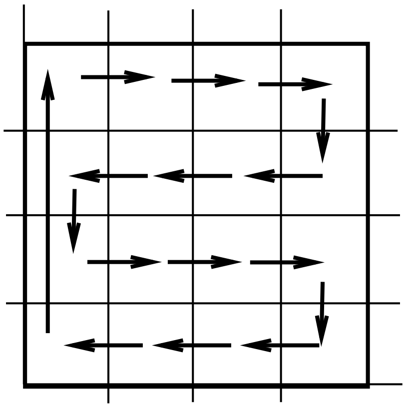
As in Problem Set # 4, we are providing you with a partially-written program (see below) that already shows you how to represent the points. The symbolic constant `N` represents the number of points. The points are stored in a global array `cities`, which contains `N` `points`. The program also defines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

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

The partial tours should be stored in an $M \times M$ array of *linked lists* called `Tours`. Each list is declared using the `node` and `list` structures. The final complete tour should also be stored in a *linked list.*



M = 9



M = 16

The Assignment:

1. Complete the given partial program into a program to find a travelling salesman tour according to the partitioning heuristic, using the double nearest neighbor heuristic within each grid square.

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 $N = 100, 400, 800$ and print out the tour produced by your algorithm in each case. Submit these printouts with your hardcopy. Compare the lengths of the tours you get with this heuristic to those you got for Problem Set # 4.

How to direct output into postcript files:

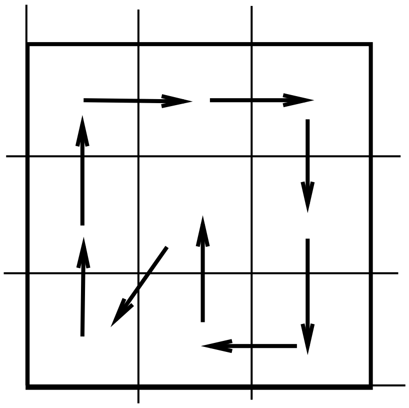
Postscript is a computer language that is used for drawing pictures. Your precepts will discuss it in greater detail (see also the postscript chapter in the reader on pages 154-160).

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("%%")`.

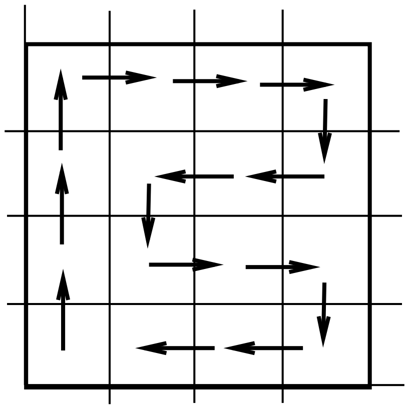
Submission:

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

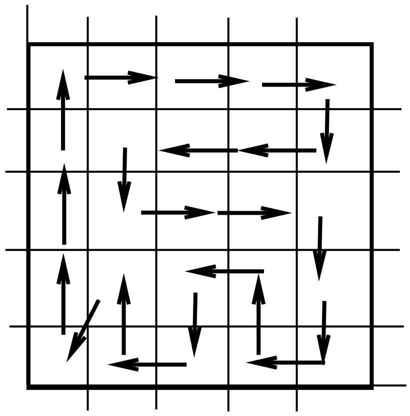
`/u/cs128/bin/submit 5 partitiontsp.c`



M = 9



M = 16



M = 25

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

Extra Credit: There are patterns for touring a grid that always go from neighbor to neighbor, eliminating the long last edge of the zigzag pattern. The general pattern differs depending on whether M is even or odd; when M is odd, one diagonal neighbor is used. The patterns for $M = 9$, 16, and 25 are shown above. Use these patterns to piece together the subtours into one tour.

The Partial Program

(can be found in /u/cs128/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 =  random()%L;
      cities[i].y =  random()%L;
      cities[i].visited =0;
    }
}
void main()
{
  struct list tour;
     /* define your variables here */

  initialize();
     /* put in your code after this point */
}
```