```
/* CMPUT 201 Assignments                                                      */
/* Dues:                                                                      */
/* #3:    11:55pm, December 8, 2017                                           */
```

(Mandatory assignment cover-sheet or head comment; without it, your work will not be marked.)

```
/* Submitting student: ___________________________________________________________

Collaborating classmates: ______________________________________________________

Other collaborators: ___________________________________________________________

References (other than textbook & lecture slides):

________________________________________________________________________________

________________________________________________________________________________

________________________________________________________________________________*/
```

Regardless of the collaboration method allowed, you must always properly acknowledge the sources you used and people you worked with. Your professor reserves the right to give you an exam (oral, written, or both) to determine the degree that you participated in the making of the deliverable, and how well you understand what was submitted. For example, you may be asked to explain any solution that was submitted and why you choose to write it that way. This may impact the mark that you receive for the deliverable.

So, whenever you submit a deliverable, especially if you collaborate, you should be prepared for an individual inspection/walkthrough in which you explain what every line of your submission does and why you choose to write it that way.

In Assignment #1, your program can successfully read in an input file that describes $n =$ `NUM_PT` points in the two-dimensional plane, within the rectangular area $[0, \texttt{MAX\_X}] \times [0, \texttt{MAX\_Y}]$. Your program for Assignment #2 computes a *minimum-weight spanning tree* (MST) for these n points. By executing the following command,

`>myprogram -i instance10_001.txt`

your program appends the edges of the MST to the input file `instance10_001.txt`, one in a line as:

$i \;\; i^* \;\; d(i, i^*)$

where p_i and p_{i^*} are the two end points of the edge, p_i is the parent of p_{i^*}, and $d(i, i^*)$ is the weight of the edge (the distance between the two points).

The goal of Assignment #3 is to lay down the edges of the MST to achieve the maximum total overlap, and to conduct numerical experiments to collect the statistics (Lecture slide set #11 contains some illustration). The following list contains the specifications for Assignment #3 (10 marks in total):

1. Using the following command to run your program for Assignment #3,

   `>myprogram -i instance10_002.txt [-o output10_002.txt]`

   Here the input file `instance10_002.txt` is in the format resulted from Assignment #2 (provided as the sample input in eClass), that is, it contains the edges of the MST.

   The option "`-o output10_002.txt`" specifies a file to write the program output; if it is not there, your program writes output to `stdout` data stream.

2. The following data type is strongly recommended to be used in your program; the subsequent description is based on this `struct`:

```
struct point {
    int index;          /* the order in the instance file             */
    int x;              /* x coordinate                               */
    int y;              /* y coordinate                               */
    int parent;         /* parent in the tree when added              */
    int num_children;   /* has value 0 -- 8                           */
    int child[8];
    int overlap_hv;     /* total overlap when horizontal then vertical */
    int overlap_vh;     /* total overlap when the other way           */
};
```

   Essentially, this new data type is for storing information associated with a point, which has a number of entries and their meanings. In particular, it is guaranteed that the number of children `num_children` one can have is at most 8; the member '`overlap_hv`' records the total overlap of the subtree rooted at the edge (`parent, index`) when the edge (`parent, index`) is laid as an L-shape first horizontally out of `parent` then vertically to reach `index` (that is, `parent` is incident at the horizontal portion of the edge and `i` is incident at the vertical portion of the edge; in the degenerate case, the horizontal portion or the vertical portion of the edge has length 0).

   When a variable of `struct point` is declared, all its members are initialized to -1, indicating *invalid* values, except `.num_children` initialized to 0.

   In the sequel, assume you declare the following array to store the n points:

```
struct point p[n];
```

3. Assume the first given point in the instance file has index/subscript 0 (that is, `p[0]` is the root of the MST). If $n = 1$, your program terminates without doing anything; otherwise (i.e., $n > 1$), your program prints to the output the values of all the members for the second point (i.e., `p[1]`), one in a line. For the member array `.child`, you only need to print out the children that are ≥ 0. These form the first set of lines in the output; print an empty line after them.

   Using the sample input file `instance10_002.txt`, your program should print the following out:

```
p[1].index = 1;
p[1].x = 0;
p[1].y = 90;
p[1].parent = 5;
p[1].num_children = 0;
p[1].child[8] = {};
p[1].overlap_hv = 0;
p[1].overlap_vh = 0;
```

4. Starting with the root of the MST, your program prints to the output the following members in one line:
   `.index`, `.num_children`, `.child[0]`, ..., `.child[.num_children - 1]`

   Then recursively prints the same information for each child. These form the second set of lines in the output; print an empty line after them.

   (This is the *depth-first-search* order of the points, or the DFS order.)

   Using the sample input file `instance10_002.txt`, your program should print the following out:

```
p[0].index = 0, .num_children = 1, .child[0] = 9
p[9].index = 9, .num_children = 1, .child[0] = 4
p[4].index = 4, .num_children = 1, .child[0] = 5
p[5].index = 5, .num_children = 2, .child[0] = 8, .child[1] = 1
p[8].index = 8, .num_children = 1, .child[0] = 7
p[7].index = 7, .num_children = 2, .child[0] = 3, .child[1] = 6
p[3].index = 3, .num_children = 0
p[6].index = 6, .num_children = 1, .child[0] = 2
p[2].index = 2, .num_children = 0
p[1].index = 1, .num_children = 0
```

5. Let $\mathcal{O}$ denote the *reversed* DFS order.

   Using the sample input file `instance10_002.txt`, this order $\mathcal{O}$ (using the indices of the points) is

```
1, 2, 6, 3, 7, 8, 5, 4, 9, 0
```

   Suppose point p_i is at the head of the order $\mathcal{O}$. There are two possible cases (also refer to lecture slide set `lecture11.pdf`):

   (a) p_i has no children (`.num_children` $= 0$). In this case, set both members `.overlap_hv` and `.overlap_vh` to 0, and p_i is said *processed* and removed from $\mathcal{O}$.

   (b) p_i has `.num_children` > 0 children. In this case, all the children must have been processed. When the edge (`.parent`, `i`) is laid as first horizontally out of `.parent` then vertically to reach `i`, for each combination of how the edges (`i`, `.child[j]`)'s for `j` $= 0, 1, \ldots,$ `.num_children` -1 are laid, compute the overlap of these `.num_children` $+1$ edges and add the `.overlap_xx`'s of all its children. (Here `xx` corresponds to how the edge (`i, .child[j]`) is laid out.) This is the total overlap for the combination. Among all combinations, the maximum total overlap is set for the member `.overlap_hv` of point p_i.

In the same way, compute `.overlap_vh` for point p_i.

Afterwards, p_i is said *processed* and removed from $\mathcal{O}$.

Note: when p_i is the root (that is, the last point in $\mathcal{O}$), which has no parent, only the combinations of the child edges are examined to compute `.overlap_hv` for point p_i, and we certainly have `.overlap_vh` = `.overlap_hv`.

6. Your program prints to the output the following lines (the last/third set of lines):
   `The total overlap is .overlap_hv (%d)`
   `The reduction rate is ...(%.2f)`

   and appends to the instance file the following comment lines:
   `#The total overlap is .overlap_hv (%d)`
   `#The reduction rate is ...(%.2f)`

   where `.overlap_hv` is replaced by its value for the root, and the reduction rate is calculated as `.overlap_hv` divided by the length of the MST.

The above specifications on your program for Assignment #3 worth <u>8 **marks**</u>. The <u>**second task**</u> in this assignment is as follows, and worths <u>2 **marks**</u>:

1. Use your program for Assignment #1 to generate 100 random instances for each of $n = 100, 200, 300,$ $400, 600, 800, 1000$ (7 values), with the fixed circuit board area $[0, 1000] \times [0, 1000]$.

   The instance files are "`instanceXXX_YYY.txt`", where `XXX` is the number n of points, and `YYY` ranges from 001 to 100.

2. For each n, run your programs (for Assignment #2 and Assignment #3) on the 100 instances, to obtain the average reduction rate, the average execution time (in minutes and seconds) of your program for Assignment #2, and the average execution time (in minutes and seconds) of your program for Assignment #3.

3. Print these values in the following way (as a text file named `result_yourCCID.txt`):
   `n, reduction rate, running time for A2, running time for A3`

   one row for each n.

4. Submit this file together with your C program for Assignment #3.

//End of Assignment #3.