Data Structure and Algorithm

Homework #5

Due: 2:20pm, Thursday, May 30, 2013

TA email: dsa1@csie.ntu.edu.tw

=== Homework submission instructions ===

- For Problem 1, submit your source codes, a Makefile to compile the source, and a brief documentation to the SVN server (katrina.csie.ntu.edu.tw). You should create a new folder "`hw5`" and put these three files in it.
- The filenames of the Makefile, and the documentation file should be "`Makefile`" and "`report.txt`", respectively; you can use any filenames for the source codes, but the filename of the executable file which is generated after executing "`make`" command should be "`main.exe`". You will get some penalties in your grade if your submission does not follow the naming rule. The documentation file should be in plain text format (`.txt` file). In the documentation file you should explain how your code works, **the reference**, and anything you would like to convey to the TAs.
- For Problem 2 through 5, submit the answers via the SVN server (electronic copy) or to the TA at the beginning of class on the due date (hard copy). Please hand in your homework with A4 paper.
- Except the programming assignment, each student may only choose to submit the homework in only one way; either all in hard copies or all via SVN. If you submit your homework partially in one way and partially in the other way, you might only get the score of the part submitted as hard copies or the part submitted via SVN (the part that the grading TA chooses).
- If you choose to submit the answers of the writing problems through SVN, please combine the answers of all writing problems into only ONE file in the pdf format, with the file name in the format of "`hw5_[student ID].pdf`" (e.g. "`hw5_b01902888.pdf`"); otherwise, you might only get the score of one of the files (the one that the grading TA chooses).
- Discussions with others are encouraged. However, you should write down your solutions by your own words. In addition, for each problem you have to specify the references (the Internet URL you consulted with or the people you discussed with) on the first page of your solution to that problem, but on `report.txt` for the programming assignment.
- **NO LATE SUBMISSION IS ALLOWED** for the homework submission in hard copies - no score will be given for the part that is submitted after the deadline. For submissions via SVN (including the programming assignment and electronic copies of the writing problems), up to one day of delay is allowed; however, the score of the part that is submitted after the deadline will get some penalties according to the following rule (the time will be in seconds):

$$\text{LATE SCORE} = \text{ORIGINAL SCORE} \times (1 - DelayTime/86400)$$

Problem 1. The DSA Kingdom (25%)

Once upon a time, there was a Devil Scary Awful Kingdom. There were T islands in the kingdom and several cities on each island. To simplify the problem, the cities are assumed to scatter on a 2-D plane.

One day, a new communication technology was developed. It can send and receive the messages between two cities with laser. Then, the king wanted to deploy the communication network on each island such that any two cities on the same island can communicate with each other.

A naïve way to build this communication network is to build laser connections between all pairs of cities. However, these laser connections would cost a lot and consume too much energy. A more efficient method is to allow some cities to act as relay stations and transfer the messages for other cities.

On the other hand, due to their inaccurate positioning technology, the laser could only be launched toward 4 directions: north, south, east, and west. In other words, only the cities with the same x coordinates or the cities with the same y coordinates can construct laser connections between them.

The cost of a laser connection depends on the distance between the two cities that it connects. In this problem, the distance is defined as the Euclidean distance, which means that the distance between (x_1, y_1) and (x_2, y_2) is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

You were a chancellor in the DSA kingdom. The king summoned you and gave you an order, "build a minimum cost laser communication network for every island. "

Input Format

The test data will be given in the following format.

The 1st line contains an integer T, followed by the description of T island.

For each island description, the 1st line contains an integer n, indicating the number of cities on this island.

In the next n lines, the i-th line contains 2 integers x_i, y_i which are separated by a space character, indicating the coordinate of the i-th city.

All cities have **distinct** coordinates - no two cities are at the same location.

Output Format

For each island, your program should output an integer indicating the cost (in terms of distance) of the minimum cost network.

If there is no way to achieve the goal on this island, just output "`-1`" (without quote).

Grading

- 3 points for the report, including the reference.

  Remember to write the reference of *Problem 1* in `report.txt`.

- 2 points for Makefile, including whether the compilation of your source codes is successful.

- 2 points for each set of test data; 20 points in total.

  The execution time limit for each set of test data is 3 seconds.

You MUST use **Kruskal's Algorithm** if you need to find a minimum spanning tree (MST); other algorithms are not allowed.

The following are conditions that you can assume for the test data.

- For 3 sets of test data, $n \leq 2000$, $0 \leq x_i, y_i \leq$ n.
- For all test data, $T \leq 10$, $n \leq 100\ 000$, $-10^9 \leq x_i, y_i \leq 10^9$.
- For 5 sets of test data, the solutions of all islands always exist.

Sample Input

```
3
5
1 1
2 1
2 2
0 2
0 0
2
1 1
0 0
1
0 0
```

Sample Output

```
6
-1
0
```

The minimum cost network of the first island is $(0,0) \leftrightarrow (0,2) \leftrightarrow (2,2) \leftrightarrow (2,1) \leftrightarrow (1,1)$. The cost is $2+2+1+1=6$.

Hints and Technical Reminders

- Some laser connections can be proven useless.

  For example, if there are three cities at coordinates $(0, a)$, $(0, b)$, and $(0, c)$, respectively, and we have $a < b < c$. The direct connection $(0, a) \leftrightarrow (0, c)$ could be replaced by two connections $(0, a) \leftrightarrow (0, b) \leftrightarrow (0, c)$ at the same cost. Though the costs are the same, the latter is better since it can connect 3 cities instead 2.

  Then, you will find out, a network has minimum cost only if all connections are between pairs of adjacent cities.

  By this result, the maximum number of possible connections in an island is $4n = O(n)$ (in 4 directions).

- Please make sure that your algorithm and data structure are efficient enough.

  Your time complexity should be $o(n^2)$ (note the small-o notation here). For example, the official solution is $O(n \lg n)$.

- You do not need to use `double` or `float` data types for this problem. (think about why.)

- Some answers can NOT be stored in a 32-bit integer. An overflow may occur with some test data. Therefore, use `long long int` when calculating the cost. Use `scanf("%lld")` to read and `printf("%lld")` to write. For WinXP user, use `"%I64d"` when testing your program locally. **Don't forget to change it back to `"%lld"` before committing.**

- The length of readable official solution contains about 150 lines of C code.

- Each test data can be solved in 2 seconds on the CSIE 217 work stations by the official solution without any compiler optimization.

Problem 2. Quicksort Execution (20%)

2.1. (10%) Partitioning is the key to the quicksort algorithm. It re-arrange the input array `A[p...r]` into two subarrays `A[p...q-1]` and `A[q+1...r]`. After partitioning, every element in `A[p...q-1]` is less than or equals to `A[q]` and every element in `A[q+1...r]` is greater than or equals to `A[q]`. In the lectures, we have learned an implementation of the partition algorithm. Here, we'll introduce a **different** version that is described on page 171 of the textbook (Cormen). The pseudo code of this partition algorithm is as follows.

```
1  PARTITION(A, p, r)
2      x = A[r]
3      i = p - 1
4      for j = p to r - 1
5          if A[j] <= x
6              i = i + 1
7              exchange A[i] with A[j]
8      exchange A[i+1] with A[r]
9      return i+1
```

Let `A[0...12]` = {16, 19, 7, 28, 22, 20, 4, 26, 5, 14, 1, 3, 17}. Please fill up the following table with the content of array `A[0...12]` after each iteration of `for` and before `return` to show the progress of `PARTITION(A, 0, 12)`.

initial	16	19	7	28	22	20	4	26	5	14	1	3	17	i= -1
j=0														i=_____
j=1														i=_____
j=2														i=_____
j=3														i=_____
j=4														i=_____
j=5														i=_____
j=6														i=_____
j=6														i=_____
j=7														i=_____
j=8														i=_____
j=9														i=_____
j=10														i=_____
j=11														i=_____
final														i=_____

2.2. (10%) The following pseudo code implements quicksort. The `PARTITION` procedure is given in 2.1.

```
1 QUICKSORT(A, p, r)
2     if p < r
3         q = PARTITION(A, p, r)
4         QUICKSORT(A, p, q-1)
5         QUICKSORT(A, q+1, r)
```

Let `A[0...9]`={29, 25, 5, 27, 8, 9, 15, 26, 13, 22}. Please use the method shown on slide 22 in the "sorting" lecture slides to show the progress of `QUICKSORT(A, 0, 9)`. Note that you need to carefully examine the order of the recursive function calls to fill up the table with correct answers.

initial	29	25	5	27	8	9	15	26	13	22

Problem 3. Quicksort Evaluation (25%)

3.1. (5%) Suppose that we'd like to use quicksort to sort 15 elements: $\{1, 2, \ldots, 15\}$. Initially, these 15 elements are in random order. In each stage of quicksort, we always choose the first element as the pivot. What is the minimum number of comparisons used in quicksort? How about the maximum? For both questions, please give examples (i.e., the initial order of these 15 elements) and show that these cases will result in the minimum/maximum. (You don't have to prove that your answers can generate the minimum/maximum)

3.2. (5%) Assume we choose the pivot randomly instead of always choosing the first element in quicksort. Out of the two cases you have given in the previous question, which do you expect to run faster? Explain (or prove if you can) you answer.

3.3. (5%) We have seen that the method of choosing the pivot in quicksort is very important: if the pivot is too small or too large, the speed of quicksort decreases. So let's try to put more efforts in choosing the pivot. Think about this: when we are going to choose a pivot, we randomly pick 3 elements (from the numbers to be sorted), then choose the median of the 3 to be the pivot. Prove that the maximum number of comparison is still $O(N^2)$ in the worst case when sorting N elements.

3.4. (5%) Let's modify the previous strategy a little bit: when we are going to choose a pivot, we randomly pick a elements and sort them using insertion sort. Then we choose the median of these a elements as the pivot. Give an example in which the number of comparison is $O(\frac{N^2}{a} + Na)$. (If you can, prove that this is the worst case - the maximum number of comparisons that can happen if we use this method to select a pivot)

3.5. (5%) For a given N, due to the result of the previous question, please choose the best a that minimizes the time complexity.

Problem 4. How do you count these inversions? (15%)

Given a sequence of numbers $\langle a_n \rangle$ with N distinct numbers, we call a pair of two numbers (a_i, a_j) an inversion if and only if $i < j$ and $a_i > a_j$. Let $I(\langle a_n \rangle)$ be the set of inversions in $\langle a_n \rangle$. For example, if $\langle a_n \rangle = \langle 13, 78, 90, 47 \rangle$, $I(\langle a_n \rangle) = \{(78, 47), (90, 47)\}$.

In this problem, we will learn to count the numbers of inversions $|I(\langle a_n \rangle)|$ in an efficient way.

4.1. (2%) Design a naïve brute-force algorithm to calculate the number of inversions $|I(\langle a_n \rangle)|$ in $O(N^2)$. The algorithm should be written in pseudo code or C code. Briefly argue the correctness of your code and show that it indeed runs in $O(N^2)$.

4.2. (3%) Consider the sequence $\langle a_n \rangle = \langle b_n \rangle \langle c_n \rangle$. It means $\langle a_n \rangle$ is a concatenation of two sequences $\langle b_n \rangle$ and $\langle c_n \rangle$. For example, if $\langle b_n \rangle = \langle 13 \rangle$ and $\langle c_n \rangle = \langle 78, 90, 47 \rangle$, $\langle a_n \rangle$ will be $\langle 13, 78, 90, 47 \rangle$. Let $inv(\langle b_n \rangle, \langle c_n \rangle)$ be the number of pairs (b_i, c_j) where $b_i \in \langle b_n \rangle$ and $c_j \in \langle c_n \rangle$ such that $b_i > c_j$. Prove that $|I(\langle a_n \rangle)| = |I(\langle b_n \rangle)| + |I(\langle c_n \rangle)| + inv(\langle b_n \rangle, \langle c_n \rangle)$.

4.3. (5%) You are given two **sorted** sequences $\langle b_n \rangle$ and $\langle c_n \rangle$. Design an efficient algorithm to calculate $inv(\langle b_n \rangle, \langle c_n \rangle)$, as defined in problem 4.2. If the two sequences contain M and K numbers, respectively, your algorithm should run in $O(M + K)$ time. The algorithm should be written in pseudo code or C code. Please also show the correctness of your code and that your code indeed runs in $O(M + K)$.

4.4. (5%) You are given an **arbitrary unsorted** sequence $\langle a_n \rangle$. Design an efficient algorithm to calculate $|I(\langle a_n \rangle)|$. If the sequence contains N numbers, your algorithm should run in $O(N \log N)$ time. The algorithm should be written in pseudo code or C code. Please also show the correctness of your code and that your code indeed runs in $O(N \log N)$.

Hint 1. Try to split the original sequence $\langle a_n \rangle$.

Hint 2. The merge sort algorithm may help you a lot.

Problem 5. Maximum Gap (15%)

You are given an array A which contains N distinct numbers. The `findGap` function calculates the maximum value of $y - x$ for all x, y in A where $x < y$ and $\nexists z \in A$ such that $x < z < y$. The function definitions are shown below. For each subproblem, write down your answer in C code and show that it meets the running time requirement.

5.1. (5%) Design a `findGap` function that runs in $O(N + M)$ time. Assume that all numbers in A are natural numbers and less than M.

```
int findGap(int* A, int N, int M){
    int i, gap = 0;
    if(N <= 1) return gap;
    if(N == 2) return A[0]<A[1] ? A[1]-A[0] : A[0]-A[1];
    int *array = (int*)malloc(M * sizeof(int));
    for(i=0;i<M;i++) array[i] = 0;
    // TODO: add some statements here

    return gap;
}
```

5.2. (10%) Design a `findGap` function that runs in $O(N)$ time. Assume that all numbers in A are real numbers.

```
double findGap(double* A, int N){
    double gap = 0.0;
    if(N <= 1) return gap;
    if(N == 2) return A[0]<A[1] ? A[1]-A[0] : A[0]-A[1];
    // TODO: add some statements here

    return gap;
}
```

Hint 1. As mentioned in the lecture, the lower bound of the time complexity for comparison-based sorting algorithms is $\Omega(NlogN)$. To bring the running time of your algorithm to be lower than this bound, you need to use a non-comparison sorting algorithm such as counting sort or bucket sort.

Hint 2. Let M and m be the maximum and minimum numbers in A. We can put all numbers in A in the range $[m, M]$ into $N-1$ buckets. The interval of each bucket is $(M-m) \div (N-1)$.

Hint 3. Since there are $N-2$ numbers other than M and m and there are $N-1$ buckets, at least one of the buckets is empty if M and m are not considered. Thus, any two numbers that are in the same bucket cannot have the largest gap - we only need to store the largest number and the smallest number in each bucket.

Hint 4. Finally, we can get a sorted list by going through the buckets sequentially.