Lab assignment 3

Problem 1: Implement the randomized divide and conquer selection algorithm. In each iteration you will pick as a pivot an element of the array uniformly at random.

Problem 2: Write a O(nlogn) divide and conquer program to count the number of inversions in an input array of integers. Print each pair of elements in the inversion.

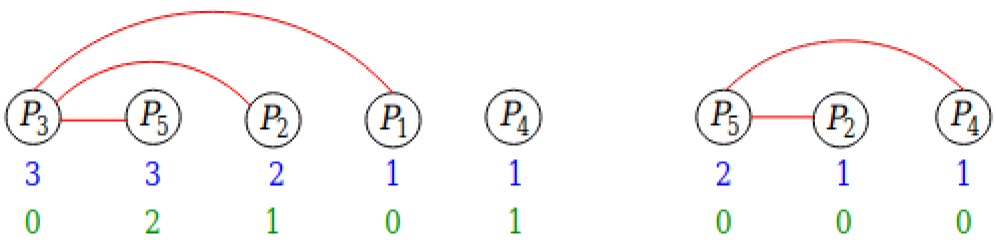
Problem 3: (from http://cse.iitkgp.ac.in/~abhij/course/lab/Algo1/Autumn16/)

Suppose that there are n persons $P_1, P_2, \ldots, P_n$. This assignment deals with answering a couple of questions about the relationships among these persons. Both the problems must be solved in $O(n^2)$ time. You may use O(n) additional space if required.

Part 1: Suppose that friendship is a mutual relation, that is, for two different i and j, P_i is a friend of P_j if and only if P_j is a friend of P_i. Also, we do not consider P_i to be a friend of P_i himself or herself. Let f_i denote the number of friends that P_i has. The sequence $f_1, f_2, \ldots, f_n$ is called a (valid) friendship sequence. We have $f_i \in \{0, 1, 2, \ldots, n-1\}$ for all $i = 1, 2, \ldots, n$. In particular, all the f_i values are O(n).

In this part, you are given a sequence $f_1, f_2, \ldots, f_n$ of n integers with $0 \le f_i \le n-1$ for all $i = 1, 2, \ldots, n$.Your task is to find out whether this can be a valid friendship sequence, and if so, to construct a friendship list that realizes this sequence. The solution is in general not unique, but it suffices to construct any solution. Since each friendship pair is counted twice, once for each person in the pair, the sum $\sum_{i=1}^{n} f_i$ must be even. In the rest of this part, assume that the given input satisfies this condition.

Repeat the following steps until n reduces to zero. First, delete all the zero entries from the sequence (because those persons do not have any friends at all), and decrement n accordingly. If n becomes zero, return *success*. If not, compute $k = \max(f_1, f_2, \ldots, f_n)$. If k >= n, this cannot be a valid friendship sequence, so return *failure*. Otherwise, sort the sequence in decreasing (actually non-increasing) order. Let this rearranged list be $k=f_{i1}, f_{i2}, \ldots, f_{in} > 0$. Choose $P_{i2}, P_{i3}, \ldots, P_{i(k+1)}$ as friends of P_{i1}. After this, the remaining counts of friends for these k + 1 persons become $0, f_{i2} - 1, f_{i3} - 1, \ldots, f_{i(k+1)} - 1$. Update the friendship count list to $0, f_{i2} - 1, f_{i3} - 1, \ldots, f_{i(k+1)} - 1, f_{i(k+2)}, f_{i(k+3)}, \ldots, f_{in}$, and go back to the top of the loop.
Here is an example:



(a) Iteration 1 (b) Iteration 2

It can be proved that this algorithm works, that is, outputs a feasible solution whenever the input friendship sequence is valid. Let us now look into some implementation details. The friendship sequence is stored in a list A of (i,f_i) pairs. The elements of A are relocated during deletion of zero entries and during sorting, so you need to maintain the index i of the person P_i along with the current friendship count f_i. The deletion of the zero entries from A can be done in place in O(n) time by relocating the non-zero entries at the beginning of the list. The maximum k can also be computed in O(n) time. If 0<k<n, we need to sort A with respect to the second component. Since the maximum value of this component is O(n) , we can use counting sort which finishes in O(n+k), that is, O(n) time. The friendship pairs added after the sorting are stored in a static two-dimensional list M which is initialized to zero before the loop. Whenever P_i is made a friend of P_j, set M[i][j] =M[j][i] =1 (so 1 means friend, 0 means not friend). Since each iteration makes at least one f_i zero, the total number of iterations is at most n, and the overall running time is $O(n^2)$.

Write a function *friendship(A, M, n)* to implement the above algorithm. The function should return *success* or *failure* (that is, a Boolean value which may be encoded as an integer).

Part 2: Now, suppose that we count the handshakes done among the n persons. Handshaking is again a mutual gesture. Let h_i denote the count of handshakes made by P_i with all of the n − 1 other persons. Let us call $h_1, h_2, \ldots, h_n$ a (valid) handshake sequence. Unlike the friendship count f_i, we cannot supply any upper bound on h_i. In particular, we cannot guarantee that $h_i = O(n)$ for all i. However, the sum $\sum_{i=1}^{n} h_i$ must be even since each handshake is counted twice in the sum. Henceforth assume that this condition holds. In this part, you are given a sequence $h_1, h_2, \ldots, h_n$ of non-negative integers. Your task is to confirm whether this is a valid handshake sequence, and if so, to determine how such a sequence can be realized. Again, the solution need not be unique, but we are done if we can come up with any correct solution.

Develop an $O(n^2)$ -time algorithm based on the following result, the correctness of which can be established. Let the given sequence in sorted order be $h_{i1}, h_{i2}, \ldots, h_{in} > 0$ for n > 4. Then, this sequence is a valid handshake sequence if and only if the sequence $h_{i1} - h_{in}, h_{i2}, \ldots, h_{i(n-1)}, 0$ is a valid handshake sequence.

The special cases n = 1 , 2 , 3 (number of non-zero entries remaining in the sequence) are not handled by the above result. For example, investigate what happens if n = 3, and we attempt to apply the result on the three remaining counts 7 , 6 , 5. This sequence has a unique solution of 4 , 3 , 2 handshakes for the three pairs. Also note that the solution for the sequence 7 , 2 , 1 is 4 , 3 , −2 handshakes for the three pairs, but a negative number of handshakes makes no sense. The iterative procedure eventually ends up in these boundary cases, so you need to work out how to handle them.

Write a function *handshake(A, M, n)* to implement the above algorithm. The function should return *success* or *failure* (that is, a Boolean value which may be encoded as an integer).

The main() function:

- Read n from the user. Then, for i = 1 , 2 ,..., n , read f_i from the user and store (i,f_i) in A. You may assume that the user ensures that the sum $\sum_{i=1}^{n} f_i$ is even. Initialize M to a 2-D n × n list of zeros.
- Call *friendship*(A,M,n). If the function returns success, print M in a format similar to that shown in the sample output.
- For i = 1 , 2 ,..., n , read h_i from the user and store (i,h_i) in A. You may assume that the user ensures that the sum $\sum_{i=1}^{n} h_i$ is even. Reuse the two-dimensional list M.
- Call handshake(A,M,n). If the function returns success, print M..

Sample output

```
n = 10

Sequence:    5   5   7   3   2   6   7   0   4   3

             1   2   3   4   5   6   7   8   9  10      Sum
         +-----------------------------------------+
       1 |       1   1       1   1   1             |    5
       2 |   1       1   1       1   1             |    5
       3 |   1   1       1   1   1   1       1     |    7
       4 |       1   1               1             |    3
       5 |   1       1                             |    2
       6 |   1   1   1               1       1   1 |    6
       7 |   1   1   1   1       1           1   1 |    7
       8 |                                         |    0
       9 |           1           1   1           1 |    4
      10 |                       1   1       1     |    3
         +-----------------------------------------+
     Sum     5   5   7   3   2   6   7   0   4   3

Sequence:    9  57  17  34  89  10  50   7  35  94

             1   2   3   4   5   6   7   8   9  10      Sum
         +-----------------------------------------+
       1 |                   9                     |    9
       2 |                  28      18          11 |   57
       3 |                  17                     |   17
       4 |                                      34 |   34
       5 |   9  28  17                      35     |   89
       6 |                                      10 |   10
       7 |      18                              32 |   50
       8 |                                       7 |    7
       9 |                  35                     |   35
      10 |      11      34      10  32   7         |   94
         +-----------------------------------------+
     Sum     9  57  17  34  89  10  50   7  35  94
```