CS 4820, Spring 2014	**Homework 2, Problem 1**
Name: Piyush Maheshwari	
NetID: pm489	
Collaborators: None	

(2) *(10 points)* Solve Exercise 4.16 in Chapter 4 of the Kleinberg-Tardos textbook.

Hint: The simplest proof of correctness that we know of uses an exchange argument.

Karma exercise: Show how to implement your algorithm in time $O(n \log n)$.

Solution

Problem Statement

Given a N transactions where each transaction is represented by $t_i \pm e_i$ and a list of N events represented by x_i, we have to match every event to a distinct transaction such that $|t_i - x_i| < e_i$. If this matching is possible output Yes, else output No.

Algorithm

The algorithm starts by first maintaining an count array A[] initialized to 0. At any point in the algorithm a[i] is equal to one if that event has been matched to an transaction and a[i] is equal to zero if that event hasn't been matched to any transaction. We maintain the transactions in a list where each list element points to another list of candidate events which could match with this transaction. Then go through the list of events sorted by time one by one, and for each transaction we check if it can be matched with this event. This can be done by checking $|t_i - x_i| < e_i$. If this is true then we add that event into the list of candidate events for this transaction. Then we sort the transaction list based on the number of events in its candidate list. Then we go through the sorted transaction list. For each transaction we choose an event e_i which hasn't been picked yet. This check can be done by whether the value of a[i] is zero or not. Pick the first such element in the list and mark a[i] equal to 1. If no such element can be picked which is possible if the list if empty or all the elements in the list have been already been matched, output No. If we find a matching event for all the transactions in the sorted transaction list then output Yes.

Algorithm 1

```
1: procedure EVENT_MATCHER(T[], E[], X[])
2:     A[] ← 0                                          ▷ Count array for events
3:     i ← 0
4:     result ← Yes                                     ▷ Default value of result is True
5:     while i < N do        ▷ Initialize list to contain candidate events for each transaction S
6:         L[i] ← (0, [])                               ▷ Each element is the size and an empty list
7:     x[] ← sorted(x[])                                ▷ Sort x[]
8:     for all x_i ∈ X[] do
9:         for all t_j, e_j ∈ T[], E[] do
10:            if |t_j − x_j| < e_i then
11:                Append i to L[j][1]                  ▷ list[i][1] represents the event list of list[i]
12:                L[i][0] ← L[i][0] + 1                ▷ Increase the size of the candidate event list
13:    L[] ← sorted(L[])                                ▷ Sort l[i] based on size of list contained in list[i][0]
14:    for all l_i ∈ L[] do
15:        found ← false
16:        for all e_j ∈ l[i][1] do
17:            if a[j] == 0 then
18:                found = true
19:                a[j] = 1
20:                break
21:        if found == false then                       ▷ This transaction had no matching event
22:            result ← No
23:            break
24:    return result
```

Proof of Correctness

Lemma 1. *If there exists no valid match solution to the problem then the method EVENT_MATCHER would return No as output.*

Proof. There are two case when there would be no valid match to the problem

1. If there is no event which lies within a transaction - In this case the algorithm will return false since the check in line 21 of the algorithm would fail for that particular transaction and return No

2. If **for any set** S of transaction the total number of distinct events that lie within the transactions in S is less that the size of S - Let S be any such set of size N. Assume the total number of distinct transactions that lie within S is $X < N$ and let they be $e_1, e_2 ... e_x$. Since this case is different from the case above, this means that no transaction has empty event list, which means that the transactions have overlapping events. Every time we execute the for loop in line 14, we find a matching event to that transaction and mark it 1 in the count array a. Since $X < N$, this means that there must be some transaction j which would not be able to find a matching transaction. And hence the algorithm will return No on such a transaction.

Combining both case 1 and 2, we prove that if there is no valid match to the problem then the algorithm would return No

□

Lemma 2. *If there is some transaction which contains a list of candidate events, then the algorithm matches the transaction with an event which has the least time stamp among the unmatched events in the candidate list for that transaction*

Proof. First we observe that we sort the event list in step 7 of the algorithm. Hence when the events are matched with the transaction in steps $8 - 12$, they are added in the transaction list in a sorted order based on time stamp. Now when the for loop is running in lines $16 - 20$ to find an event to match with a transaction , it would pick the an unmatched event with the least time stamp since that list is sorted. By unmatched event, we mean any event which hasn't been matched to any transaction yet. □

Lemma 3. *If a solution exists where each transaction is matched with an unmatched event in its list with the least time stamp, our algorithm will find it.*

Proof. This follows from lemma 2. □

Lemma 4. *If there exists a valid match solution to the problem then the method EVENT_MATCHER would return Yes as output.*

Proof. We can divide the solution set into two cases and prove this lemma separately for both

1. The candidate event list for each transaction contains only one event. In this case there is only one output and our algorithm would find it and output Yes.

2. In this case there is some transaction which has more than one event in its candidate list. Let the solution where each transaction is matched with an unmatched event in its list with the least time stamp be S_0. We will prove that if there is a solution S which is different from the solution output by our algorithm S_0, we can transform it into S_0 (which is our greedy solution). So we will try to show that any solution can be transformed into S_0 and we know from lemma 3 that if a solution S_0 exists our algorithm will find it. Hence we can complete the proof.
   Also when we call a solution S_a closer to S_0 than S_b we mean that S_a contains more transactions which are matched to their first(in timestamp order) unmatched event than S_b does.

   From lemma 2, we know that our algorithm matches each transaction to an unmatched event in its list which has the least time stamp. Now consider a solution S in which an transaction t_i has been matched to an event $e_b > e_a$ but e_a was in its candidate list at that point and was unmatched. This means that there exists some solution S' where t_i had been matched to e_a otherwise e_a is not a valid unmatched candidate. Let in the solution S, e_a is matched to some t_j.

   Now there can two cases. In the first case t_j can span e_b. Then we can simply swap the events of t_i and t_j and we get a solution which is strictly closer to S_0.

In the second case if t_j does not span e_b, this means that $t_j + e_j < e_b$. Assume t_j is matched to some e_m in S' such that $e_m < e_b$ since $t_j + e_j < e_b$. Let this e_m is matched to some t_k in S. Now either $t_k + e_k < e_b$ or there exists some e_n to which it was matched in S' such that $e_n < e_b$. We can keep on using the same argument and since we have a solution with finite number of transactions, ultimately we will get a t_g which spans across e_b that is $t_g + e_g < e_b$. Now apply to following transformation to S - match t_j to e_m, t_k to e_n ... t_g to e_b and then we can finally match t_i to e_a. We have basically found a sort of a circular matchings and just shifted the matched pairs such that they still have valid matches. We will still have a valid solution S'' which is closer to S_0 than S, since we have matched t_i to an unmatched candidate event with the least time stamp. Now we can repeatedly use the same transformation, until the solution S is completely transformed into S_0. And hence we have proved that if there exists a valid solution S different from S_0, there will exists a valid solution S_0 which can be found by applying the above transformation.

Combining lemma 3 and the argument above, we have proved that if there exists a valid solution, then out algorithm will find it.

Complexity Analysis

1. Steps 5-6 take $O(n)$
2. Step 7 takes $O(nlogn)$
3. Step 8-12 takes $O(n * n)$
4. Step 13 takes $O(nlogn)$
5. Step 14-23 takes $O(n * n)$

Total time is $O(n * n)$

□