Indexing Method

July 21, 2020

1 Current Algorithm n^2

1.1 Pseudocode

Algorithm 1 Extracting event pairs

```
checked ← HashSet(strings)
index ← HashSet ((idA,idB),List of times)
for event in Sequence do                                  ▷ First loop through the events
    eventA,timeA ← event.id, event.time
    if !checked then
        loop ← HashSet (strings)
        for event_2 in Sequence from eventA to the end do  ▷ Second loop through the events
            eventB,timeB ← event_2.id, event_2.time
            if eventA==eventB then
                if (eventA,eventB) ! in index then
                    index append (eventA,eventB),[timeB,timeA]
                else
                    index[(eventA,eventB)]=timeB+oldList
                end if
                for r in loop do                          ▷ Loop through the unique events in loop
                    index[(eventA,r)]=timeB+oldList
                end for
                clear loop
            else if eventB ! in loop then
                if (eventA,eventB) ! in index then
                    index append (eventA,eventB),[timeB,timeA]
                else
                    index[(eventA,eventB)]=timeB+oldList
                end if
                loop append eventB
            end if
        end for
        checked append eventA
    end if
end for
for row in index do                                       ▷ Index has size O(l^2)
    if list lengh %2 != 0 then
        list drop first element —
    end if
end for
return index with list of times reversed
```

1.2 Complexity

Even though there are 2 loops iterating the events, the if statement in line 5 will be true only l times (where l is the number of distinct elements) and so the complexity for the code in lines 3-29 is $O(nl^2)$, with n being the length of the sequence. After that it will perform a validation check to all elements in index and return the reversed lists. Thus the total complexity is equal to $O(nl^2 + nl^2) \Rightarrow O(nl^2)$. Total space required is $O(n + l^2)$, for index and checked hashSets.

2 Indexing Algorithm

In this method, we first find the indexes (or the timestamps) in which each event occurs and the create the event pairs.

2.1 Pseudocode

Algorithm 2 Indexing Method

```
1: HashMap ← (event_id):[index1,index2,...]          ▷ parse sequence to get indexes for each event
2: for all events in HashMap do
3:     for all events in HashMap do                  ▷ Create pairs for every couple of events
4:         Create_Pairs(HashMap[event_a],HashMap[event_b]))
5:     end for
6: end for
```

Algorithm 3 Create pairs for every couple of distinct events

```
procedure CreatePairs(indexesA,indexesB)             ▷ indexes can be also timestamps
    i,j,prev ← 0,0,-1
    pairs ← [ ]
    while i < indexesA.size and j < indexesB.size do
        if indexesA[i] < indexesB[j] then
            if indexesA[i] > prev then pairs append (indexesA[i],indexesB[j]) prev ←indexesB[j] i←1 j←1
            elsei←1
            end if
        else
            j←1
        end if
    end while
    return pairs
end procedure
```

2.2 Complexity

In line 1 we loop once the entirely sequence, to find the indexes of each distinct event ($O(n)$), then the next loops in line 2-3, will get all the possible event pairs ($O(l^2)$) and finally the procedure in line 4, will pass through their indexes ($O(n)$). This gives a total complexity of $O(n + l^2 n) \Rightarrow O(nl^2)$. Total space required is $O(n + l^2)$, for the hashMap and the pairs.

3 State Method

In this method, we save the state of the sequence, so we can compute all the pairs without looking the previous events.

3.1 Pseudocode

Algorithm 4 State method

```
1: index ← HashSet((event_a,event_b) → [index_i, index_j, ...]) for all possible pairs    ▷ Ω(l^2) space
2: for all events in seqeuence do
3:     Add_New(index,event,distinct_events)
4: end for
```

Algorithm 5 Add new event in the structure

```
procedure ADD_NEW(index, new_event,distinct_events)
    for all combinations where new_event is first event do
        update state                          ▷ Some trivial compares and updates, O(1)
    end for
    for all combinations where new_event is second event do
        update state
    end for
end procedure
```

3.2 Complexity

In line 2, loop is passing through all the events in the sequence and for every event executes the procedure Add_new. This procedure has 2 loops passing through the distinct_events (l), which gives us a total complexity of $O(n2l) \Rightarrow O(nl)$

4 Experiments

All experiments were executed in a computer with 16GB RAM and 3.2GHz processor

5 Discussion

As we can see from both Figures 1,2, Indexing method outperforms the other two. We think that the main reason is the simplicity of the code. State method has also some advantages, in a dynamic domain, where the events have the form of a stream, it will be more efficient to process every new event in $O(l)$, which is independent from the length of the sequence. Next step is to try to optimize the disk I/Os for the State method, in order to achieve better times, even with large number of distinct events.

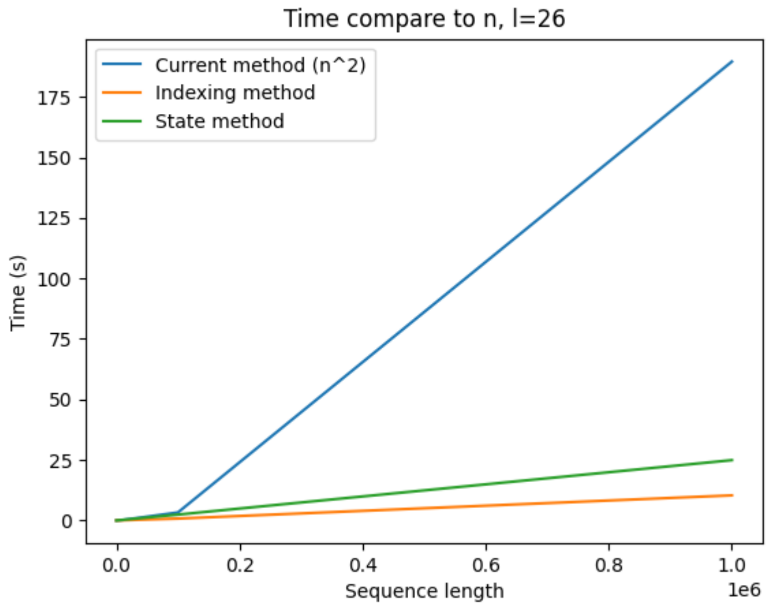

Figure 1: Compare execution times for different length of sequence (n)), l is equal to 26 (English alphabet).

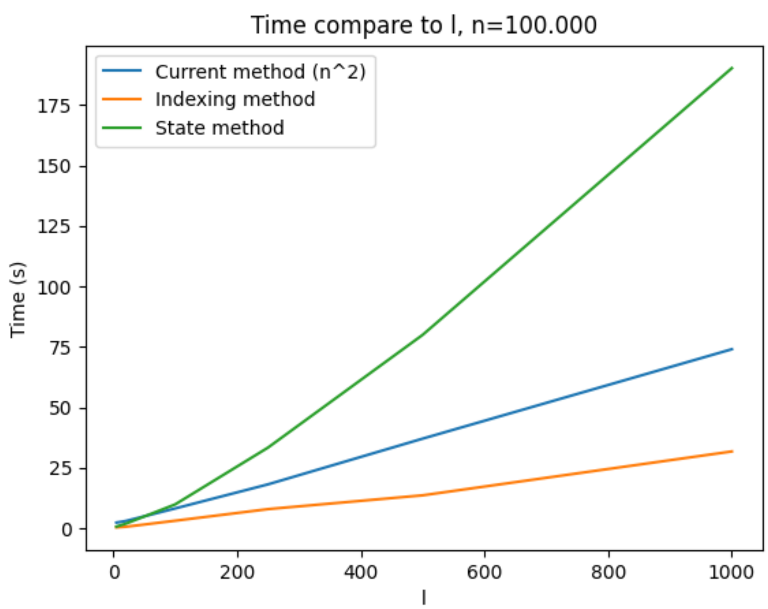

Figure 2: Execution times based on different number of distinct events (l).