HOMEWORK 6

CS677 - Algorithm Analysis

Due Date : November 14, 2019

Dzung Bui

Exercise 1 - *100 points*

a. Recursive formula, definitions and explanation

maxS(j) = maximum number of events can be watched if event j must be seen.
Because event i must be seen, so the previous seen event i must meet the requirement:

$$j - m \geq |d_j - d_m| \tag{1}$$

Introducing set E - consists of all possible events i which can be seen prior event j. If E is empty, the maximum number of event can be watched if we must watch event j is 1. If not, just searching for the event which can bring most seen event prior to it. Thus, the problem for event j is converted to the sub-problem i, where $i < j$. Thus, the recursive formula of this problem is:

$$maxS(j) = \begin{cases} 1, & \text{if } E \text{ is empty} \\ 1 + max(maxS(E_1), maxS(E_2), ..., maxS(E_k)), & \text{else} \end{cases} \tag{2}$$

b. Write and algorithm that computes the optimal value for this subproblem

maxS(S, L, C, k)

// Pre-processing data ——-

Define *pre_event* variable

for i = 0 to n

    if i = the first event

        S[i] = 1

        pre_event = - 1 *(start at this event)*

    else

        for i = 0 to i *(run from event 0 to current event i)*

            group all possible previous events of current event i into set **O**

        select even j which provide most seen events

        save maximum event and event index to arrays S and L

//print out the event list - providing max seen event

for j = 0 to *L.length*

    start at the last event, tracing back the event list from L

return true

```
The input array (event and coordinate ist):
   0     1     2     3     4     5     6     7     8     9
   0     1    -4    -1     4     5    -4     6     7    -2
```

Figure 1: Input Data

```
Maximum number of event can be seen if event (0 to 9) MUST be seen:
 0  1  2  3  4  5  6  7  8  9
 1  2  1  3  3  4  4  5  6  5

Previous event List to reach maximum number of event can be seen for event (0 to 9):
(-1 value means that event is the starting event)
 0  1  2  3  4  5  6  7  8  9
-1  0 -1  1  1  4  3  5  7  6
```

Figure 2: Solution to sub-problem

```
If event  9 must be seen
There are  5 events in total can be seen
go to Event  6
go to Event  3
go to Event  1
go to Event  0
```

Figure 3: Optimal Solution

c.

maxS_adjusted(S, L, C, k)

// Pre-processing data ——-

Define *pre_event* variable

for i = 0 to n

if i = the first event

S[i] = 1

pre_event = - 1 *(start at this event)*

else

for i = 0 to i *(run from event 0 to current event i)*

group all possible previous events of current event i into set **O**

select even *j* which provide most seen events

save maximum event and event index to arrays S and L

//print out the event list - providing max seen event
for g = *S.length* to 0
//introduce event_index g: running from the last event to the first event
apply each *g* to array *L* to find the optimal solution for the new select event
return true

d. If the MUST seen event is changed, all the previous event also changed to reach maximum number of event can be seen. The figures following shows the algorithm output.

```
If event  9 must be seen
There are  5 events in total can be seen
go to Event  6
go to Event  3
go to Event  1
go to Event  0
```

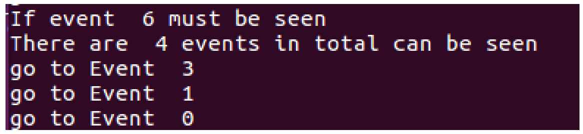
Figure 4: Event 9 must be seen

```
If event  8 must be seen
There are  6 events in total can be seen
go to Event  7
go to Event  5
go to Event  4
go to Event  1
go to Event  0
```

Figure 5: Event 8 must be seen

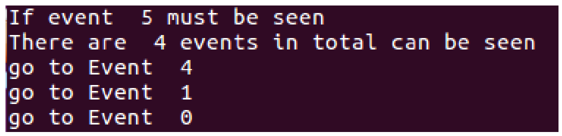
```
If event  7 must be seen
There are  5 events in total can be seen
go to Event  5
go to Event  4
go to Event  1
go to Event  0
```

Figure 6: Event 7 must be seen



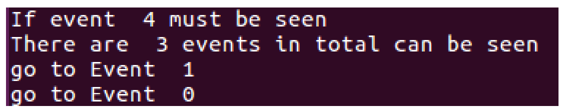
```
If event  6 must be seen
There are  4 events in total can be seen
go to Event  3
go to Event  1
go to Event  0
```

Figure 7: Event 6 must be seen



```
If event  5 must be seen
There are  4 events in total can be seen
go to Event  4
go to Event  1
go to Event  0
```

Figure 8: Event 5 must be seen



```
If event  4 must be seen
There are  3 events in total can be seen
go to Event  1
go to Event  0
```

Figure 9: Event 4 must be seen

Exercise 2 -Show that in order to fully parenthesize an expression having n matrices we need n-1 pairs of parentheses. *20 points*

Answer Assuming we have n matrix multiplication:

$$A = A_1 A_2 ... A_n \tag{3}$$

We prove this by induction.

Base Case : Let n = 2. multiply two matrices, A_1 and A_2 : The (unparenthesized) product is A_1 A_2 . Since there is only one multiplication operation, there is only one parenthesization $(A_1 A_2)$.

Induction Hypothesis : Suppose that for n < k, a full parenthesization of an n-element expression has exactly n − 1 pairs of parentheses.

Induction Case: considering a product of $n = k+1$ matrices, A_1 , A_2 , . . . , A_k , A_{k+1}. Now, suppose that the optimal parenthesization splits the product at the j^{th} element. That is, the optimal solution has the recursive structure: $((A_1 A_2 \ldots A_j)(A_{j+1} A_{j+2} \ldots A_{k+1}))$ where the two sub-products $(A_1 A_2 \ldots A_j)$ and $(A_{j+1} A_{j+2} \ldots A_{k+1})$ are recursively solved optimally.

By the induction hypothesis, the two sub-products require $j{-}1$ and $((k{+}1){-}j{+}1){-}1$ parenthesis respectively. Taken together, the induction hypothesis allows us to compute number of parentheses required to fully parenthesize the two sub-products is: $(j{-}1)+(((k+1){-}j+1){-}1)=k$

Exercise 3

Assuming the dimensions of matrices A, B, C, D, and E are compatible, and they can be multiplied in any order. For the first matrix selection, there is 5 choices. For the second, third, and fourth matrix, there are 4, 3, 2 options. Thus, the number of order for multiplication are:

$$5x4x3x2 = 120 \tag{4}$$

ABCDE	ABCED	ABDCE	ABDEC	ABEDC	ABECD
ACBDE	ACBED	ACDBE	ACDEB	ACEBD	ACEDB
ADBCE	ADBEC	ADCBE	ADCEB	ADEBC	ADECB
AEBCD	AEBDC	AECBD	AECDB	AEDBC	AEDCB
BACDE	BACED	BADCE	BADEC	BAEDC	BAECD
BCADE	BCAED	BCDAE	BCDEA	BCEAD	BCEDA
BDACE	BDAEC	BDCAE	BDCEA	BDEAC	BDECA
BEACD	BEADC	BECAD	BECDA	BEDAC	BEDCA
CBADE	CBAED	CBDAE	CBDEA	CBEDA	CBEAD
CABDE	CABED	CADBE	CADEB	CAEBD	CAEDB
CDBAE	CDBEA	CDABE	CDAEB	CDEBA	CDEAB
CEBAD	CEBDA	CEABD	CEADB	CEDBA	CEDAB
DBCAE	DBCEA	DBACE	DBAEC	DBEAC	DBECA
DCBAE	DCBEA	DCABE	DCAEB	DCEBA	DCEAB
DABCE	DABEC	DACBE	DACEB	DAEBC	DAECB
DEBCA	DEBAC	DECBA	DECAB	DEABC	DEACB
EBCDA	EBCAD	EBDCA	EBDAC	EBADC	EBACD
ECBDA	ECBAD	ECDBA	ECDAB	ECABD	ECADB
EDBCA	EDBAC	EDCBA	EDCAB	EDABC	EDACB
EABCD	EABDC	EACBD	EACDB	EADBC	EADCB