Tutorial 1

YASH PARAG BUTALA - 17CS30038

23-07-2019

1 Problem Statement

$A[1..m]$ and $B[1..n]$ are two 1D arrays containing m and n integers respectively, where $m \leq n$. We need to construct a sub-array $C[1..m]$ of B such that $\sum_{i=1}^{m} \left|A[i] - C[i]\right|$ is minimized.

2 Recurrences

In the brute-force approach we will try all sub-sequences of B of length m and try to find minimum sum of corresponding elements' absolute differences. But it will take exponential time (nCm). To make it polynomial we will use the optimal sub-structure property of the problem.
Suppose we have solution for A=a1,a2,...,ai and B=b1,b2,..,bj such that $j \geq$i and all solutions to their smaller prefixes stored in a matrix. Then: Solution of A=a1,a2,...,ai+1 and B=b1,b2,...bj+1 will depend on whether we consider the absolute difference of ai+1 and bj+1 in the sum or not.
Hence solution to A[1 to i+1] and B[1 to j+1] is minimum of:
1. **Solution of A[1 to i],B[1 to j]+absolute difference(ai+1,bj+1)** : we add absolute difference between (ai+1)th and (bj+1)th term as bj+1th term is considered in sub-sequence corresponding to first j+1 elements.
2. **Solution of A[1 to i+1]),B[1 to j]** : when (bj+1) element is not considered in i+1 length sub-sequence of B corresponding to first j+1 elements.

Thus recursive solution :
Let dp[i][j] contain solution of prefixes of length i and j of A and B.The subsequence of B has length i which we wish to be m.

$$dp(i,j) = \begin{cases} abs(a1-b1), & \text{if } i = 1 and j = 1 \\ min(dp[i][j-1], abs(a[i]-b[j])), & \text{if } (i = 1 and j > 1) \\ min(dp[i][j-1], dp[i-1][j-1] + abs(b[j]-a[i])), & \text{if } (j > i and i \neq 1) \\ dp[i][j] = dp[i-1][j-1] + abs(a[i]-b[j]), & \text{if}(i = j and i \neq 1) \\ No solution, & \text{if}(i > j) \end{cases}$$

3 Algorithm

Result: dp(m,n)
for *i in range 1 to m* **do**
  for *j in range i to n* **do**
    if *i=1 and j=1* **then**
      dp(i,j)=abs(ai-bj)
    else if *i=1 and j>1* **then**
      dp(i,j)=min(dp(i,j-1),abs(ai-bj))
    else if *i=j* **then**
      dp(i,j)=dp(i-1,j-1)+abs(ai-bj)
    else
      dp(i,j)=min(dp(i-1,j-1)+abs(ai-bj),dp(i,j-1))
    end
  end
end

Algorithm 1: Find minimum absolute sum

Result: sub-sequence
stack C
i=m
j=n
dp,A,B
`Get-sub-sequence`(i, j)
  if $i = 1$ **then**
    C.push(bj)
    return
  end
  else
    if *dp(i,j)=dp(i-1.j-1)+abs(ai-bj)* **then**
      C.push(bj)
      `Get-sub-sequence`$(i - 1, j - 1)$
    end
    else
      `Get-sub-sequence`$(i, j - 1)$
    end
  end

Algorithm 2: Find the corresponding sub-sequence

4 Demonstration

1. A =(2,7,2) and B=(5,3,6,8)
m = 3 and n = 4
Let dp be a mXn matrix.
Once we run the the algo :

```
dp(1,1)= abs(2-5) =3
```

```
    dp(1,2)= min(abs(2,3),dp(1,1))=1
    dp(1,3)= min(abs(2,6),dp(1,2))=1
    dp(1,4)= min(abs(2,8),dp(1,3))=1
    dp(2,2)= dp(1,1)+abs(7-3) =7
    dp(2,3)= min(dp(1,2)+abs(7-6),dp(2,2)) =2
    dp(2,4)= min(dp(1,3)+abs(7-8),dp(2,3)) =2
    dp(3,3)= dp(2,2)+abs(6-2) =11
    dp(3,4)= min(dp(2,3)+abs(2-8),dp(3,3)) =8
    Thus dp is :
        3   1   1   1
        -   7   2   2
        -   -   11  8
    C will push for j=4,3,2
    Thus C={3,6,8}

```

2. A =(3,1,9,2) and B=(5,3,6,1,12)
m = 4 and n = 5
Let dp be a mXn matrix.
Once we run the the algo :

```
    dp(1,1)= abs(3-5) =2
    dp(1,2)= min(abs(3,3),dp(1,1))=0
    dp(1,3)= min(abs(3,6),dp(1,2))=0
    dp(1,4)= min(abs(3,1),dp(1,3))=0
    dp(1,5)= min(abs(3,12),dp(1,4))=0
    dp(2,2)= dp(1,1)+abs(1-3) = 4
    dp(2,3)= min(dp(1,2)+abs(1-6),dp(2,2)) =4
    dp(2,4)= min(dp(1,3)+abs(1-1),dp(2,3)) =0
    dp(2,5)= min(dp(1,4)+abs(1-12),dp(2,4)) =0
    dp(3,3)= dp(2,2)+abs(9-6) = 7
    dp(3,4)= min(dp(2,3)+abs(9-1),dp(3,3)) =7
    dp(3,5)= min(dp(2,4)+abs(9-12),dp(3,4)) =3
    dp(4,4)= dp(3,3)+abs(2-1) =8
    dp(4,5)= min(dp(3,4)+abs(2-12),dp(4,4)) =8
    Thus dp is :
        2   0   0   0   0
        -   4   4   0   0
        -   -   7   7   3
        -   -   -   8   8

    Function GetC runs as:
    i=4,j=5
    ->  dp[4][5]!=dp(3,4)+abs(ai-bj)
        j=4
    ->  dp[4][4]=dp(3,3)+abs(ai-bj)
        (push 1 in C)
        i=3 j=3
    ->  dp[3][3]=dp(2,2)+abs(ai-bj)
        (push 6 in C)
        i=2 j=2
    ->  dp[2][2]=dp(1,1)+abs(ai-bj)
        (push 3 in C)
        i=1 j=1
    ->  i=1
        (push 5 in C)
   Thus C={5,3,6,1}
```

5 Time and space complexities

5.1 Time Complexity

We need a loop running inside other. The operation in the loop take constant time O(1).
Hence, Time complexity = O(m*n)

5.2 Space Complexity

The algorithm mentioned above will require order (m*n) space inorder to store the value in a matrix dp like above.
Hence Space required = O(m*n)