Algorithms (6470) HW03a

Alex Darwiche

July 1, 2024

Answers

Q1

(a) Find the optimal way to parenthisize matrices with dimenions: <9,3,8,2,5,6>

	1	2	3	4	5
1	0	216	102	192	270
2		0	48	78	144
3			0	80	156
4				0	60
5					0

	1	2	3	4	5
5	270	144	156	60	0
4	192	78	48	0	
3	102	48	0		
2	216	0			
1	0				

	1	2	3	4	5
1		1	1	3	3
2			2	3	3
3				3	3
4					4
5					

	1	2	3	4	5
5	3	3	3	4	
4	3	3	3		
3	1	2			
2	1				
1					

Q2

(a) To find the maximum total product of a subarray within A we can use the following logic. We first know that the array A has positive and negative integers. This means, that we need to keep track of both maximum and minimum values throughout our dynamic programming solution. The

intuition for this will be that if we're trying to find the Maximum Product subarry of A, we can rewrite that as the maximum of 4 different elements:

(1) A[1]

(2) MaxSubArrayOf(A[2:n])

(3) A[1]*MaxSubArrayOf(A[2:n])

(4) or A[1]*MinSubArrayOf(A[2:n])

(b) We can see here, how the larger problem can be broken down into smaller parts, by understanding that the Maximum Product is simply the maximum AND MINIMUM product of smaller arrays.

(c) Recurrence Relation

```
MaxSubArray(A) = Max(
                        A[1],
                        A[1]*MaxSubArray(A[2:n]),
                        A[1]*MinSubArray(A[2:n]),
                        MaxSubArray(A[2:n])
                        )
```

(d) It is important to note that we keep track of the minimum of the sub arrays as well. This is because a negative times a negative is a positive number.

(e) Sample Psuedocode of an iterative solution

```
def MaxSubArray(A):
    max = min = A[1]
    n = A.size()
    upper_bound_max = lower_bound_max = 1
    upper_bound_min = lower_bound_min = 1

    for i in 2 to n:
        // Update Max Values
        if (A[i] > max):
            lower_bound_max = i
            upper_bound_max = i
            max = A[i]
        else if (A[i]*max > max):
            upper_bound_max = i
            max = A[i]*max
        else if (A[i]*min > max):
            lower_bound_max = lower_bound_min
            upper_bound_max = i
            max = A[i]*min
```

```
            // Update Min Values
            else if (A[i] < min):
                lower_bound_min = i
                upper_bound_min = i
                min = A[i]
            else if (A[i]*max < min):
                lower_bound_min = lower_bound_max
                upper_bound_min = i
                min = A[i]*max
            else if (A[i]*min < min):
                upper_bound_min = i
                min = A[i]*min

        return max, lower_bound, upper_bound
```

Q3

(a) This problem requires that we "convert" DISTANCE to DESTINY. This can be done using the bottom up approach where we determine the actions needed and fill out the following table.

(b) Recurrence Relation

```
Assume A = "DISTANCE" and B = "DESTINY"

if A[i] = B[j] then this is a match
    edit(i+1,j+1)
if A[i] does not equal B[j]:
    1 + minimum(
            edit(i, j+1) //Insertion
            edit(i+1, j) //Deletion
            edit(i+1,j+1) //Replacement
        )
```

(c) Table illustrating this process:

	D	I	S	T	A	N	C	E	
D	4	5	5	6	7	7	6	6	7
E	5	4	4	5	6	6	6	5	6
S	5	4	3	4	5	5	5	5	5
T	6	5	4	3	4	4	4	4	4
I	6	5	5	4	3	3	3	3	3
N	7	6	5	4	3	2	3	2	2
Y	8	7	6	5	4	3	2	1	1
	8	7	6	5	4	3	2	1	0

(d) 4 total actions with 3 Replacements and 1 Deletion, along with 4 matches.

$$\text{Cost} = 3 * (-4) + (-2) + 4 * (4) = +2$$

Q1 (Graduate students only)

(a) This problem asks us to find the min total perimeter of triangles in a triagulated convex polygon.

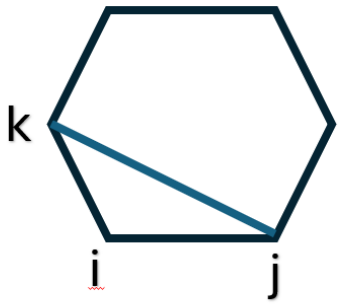
(b) The inuition for this will be to continually break down the polygon into smaller and smaller polygons until it is all triangles. Then you return the minimum perimeter after looking through all possible triangulations.

(c) To do this, we find the minimum perimeter triangle between vertex i, j, and k. You Start by setting i to the first vertex and j to the last vertex. Then, k iterates from the vertex adjacent to i, until it is adjacent to j. We can splice a polygon into subpolygon and calculate the cost of all triangulations.

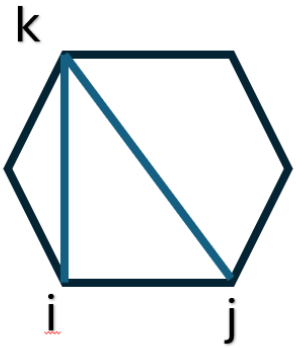
(d) 3 Vertices will always create a triangle. We want to find the perimeter of that triangle, and then find the triangles that describe the remaining polygons.

(e) As you can see in the following picture, there are 2 cases when splitting a polygon. You either create an edge triangle and a polygon, or you create a triangle and two polygons. Either way, the following psuedocode will work.

(f) In both cases, you calculate MinPerimeter(i,j) = MinPerimeter(i,k) + MinPerimeter(j,k) + Perimeter(i,j,k)



Case 1:
You can see here, that there are two polygons, the ijk triangle, and the polygon from k to j. With the 2 calls of minPerim, we will return the cost of triangulation of both sections. The perimeter of ijk triangle is calculated in the call.



Case 2:
There are three polygons in this case, the ijk triangle, the polygon from k to j and the polygon from i to k. The perimeter of ijk triangle is calculated, and then we solve for the other 2 polygons in the recurrence call of minPerim.

(g) Psuedocode

```
# Use distance formula to calculate total cost
def perimeter(i,j,k):
    return d(i,j) + d(i,k) + d(j,k)

def minPerim(vertices, i, j):
    vertices = (x1,y1), (x2,y2)...(xN,yN)
    i = 1
    j = n

    # Base case vertices next to one another
    if i + 2 > j:
        return 0

    # Next, find the min triangulation
    for k in i+1 to j:
        min = min(min,
            minPerim(vertices, i, k) +
            minPerim(vertices, k, j) +
            perimeter(i,j,k)
        )
    return min
```