Tutorial 6 Exercise Solutions

COMP526: Efficient Algorithms

11–12 November, 2024

Exercise 1. Suppose you are given an array $A = A[0..n)$ containing the price history of shares of a stock of the Acme Corporation. That is, $A[i]$ stores the price of a share of Acme stock on day i. Given this price history, you would like to find the maximum profit achievable by buying and selling a single share of Acme stock during the time interval $0..n-1$. That is, you wish to find the maximum possible value of $P[s] - P[b]$ where $b \le s$ is the day on which you buy the stock and s is the day on which you sell the stock.

(a) Explain how this problem can be solved in $\Theta(n^2)$ time using a brute force approach.

(b) Devise a divide an conquer algorithm for this problem. Be sure to:

- explain how the array A is divided;
- describe how sub-solutions can be combined to an overall solution;
- analyze the running time of your procedure.

(c) (Challenge.) Can you solve the profit maximization problem in $O(n)$ time?

Solution. For the brute force approach, consider the following procedure:

```
procedure BruteForceMax(P[0..n))
    max ← 0
    for b = 0, 1, ..., n − 1 do
        for s = b, b + 1, ..., n − 1 do
            if P[s] − P[b] > max then
                max ← P[s] − P[b]
            end if
        end for
    end for
    return max
end procedure
```

For the divide and conquer approach, we make the following observation: in order to find the maximum profit achievable during the time interval $[i..k]$ with $j = (i+k)/2$, one of the three cases must occur:

1. the maximum occurs with $i \le b, s \le j$ (i.e., buying and selling in the left half of the sub-interval),

2. the maximum occurs with $j \le b, s \le k$,

3. the maximum occurs with $i \le b \le j \le s \le k$.

To devise a divide and conquer procedure, we can solve cases 1 and 2 by recursion (with a base case of profit 0 when $i = k$). For case 3, we observe that maximum profit is found by taking b to be the index of the minimum value in $P[i..j]$ taking s to be the index of the maximum value in $P[j..k]$. This approach suggests the following algorithm:

```
1: procedure DCMax(P[i..k])
2:     if i = k return 0
3:     j ← (i + k)/2
4:     m ← max{DCMax(P[i..j]), DCMax(P[j..k])}
5:     b ← i
6:     for b′ = i + 1, i + 2, ..., j do
7:         if P[b′] < P[b] then
8:             b ← b′
9:         end if
10:    end for
11:    s ← j
12:    for s′ = j + 1, j + 2, ..., k do
13:        if P[s′] > P[s] then
14:            s ← s′
15:        end if
16:    end for
17:    return max{m, P[s] − P[b]}
18: end procedure
```

For the analysis of this procedure, we claim that if $T(n)$ denotes the running time DC-MAX on an input of size $n = k - i$, then T satisfies $T(n) = 2T(n/2) + \Theta(n)$. To see this, note that the two recursive calls in Line 4 have a running time of at most $T(n/2)$ each. The remaining code in lines 5–17 takes time $\Theta(n)$ as the procedure iterates over the values of $P[i..j]$ once. Since the running time satisfies $T(n) \le 2T(n/2) + \Theta(n)$, the overall running time is $\Theta(n \log n)$ as in our analysis of MERGESORT.

For the final part of the problem, consider the following approach: each day, $s = 0, 1, \ldots, n-1$, you wish to determine whether or not selling on day s would maximize your profit in the interval $P[0..s]$. In order to do so, you should compare the maximum value achievable in $P[0..s-1]$ to the maximum achievable by selling on day s. In order to compute the latter value, note that we only need to store the index b of the minimum value in $P[0..s]$. Then $P[s] - P[b]$ is the maximum value achievable by selling on day s. We can formalize this approach with the following algorithm:

```
procedure FastMax(P[0..n])
    max, b ← 0
    for s = 0, 1, ..., n − 1 do
        if P[s] < P[b] then
            b ← s
        else if P[s] − P[b] > max then
            max ← P[s] − P[b]
```

```
        end if
    end for
    return max
end procedure
```

□

Exercise 2. Consider the the pattern $P = \texttt{ABACADABA}$ on the alphabet $\Sigma = \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}\}$.

(a) Compute the deterministic finite automaton (DFA) for searching for the pattern P in a text T

(b) Compute the look-up table $\delta[][]$ corresponding to the DFA you found in part (1).

(c) Use your DFA or lookup table to search for P in the text $T = [0, 30)$ below.

```
T = ABABACABABACADBABABACADABAABAB
```

For each index $i = 0, 1, \ldots, 29$ write the state that the DFA is in after reading the character at index i in T.

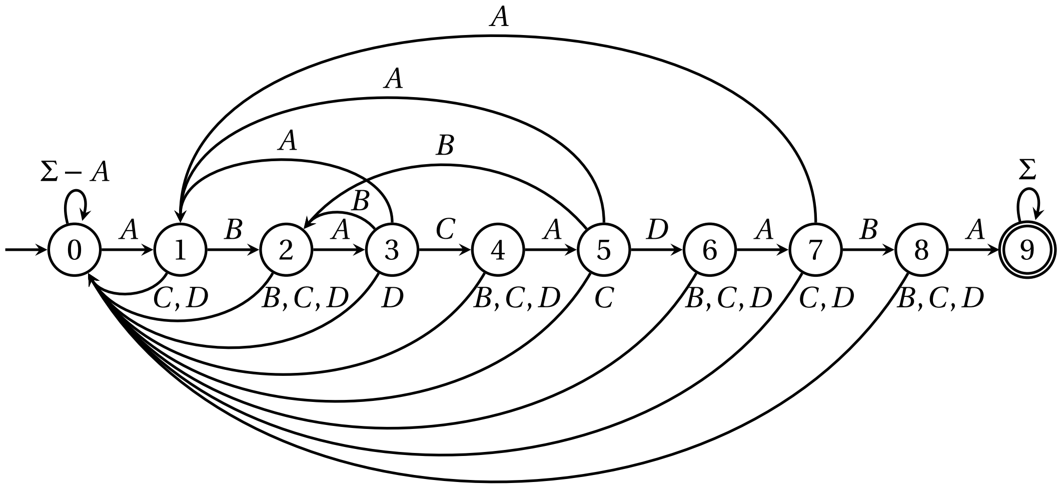
Solution. First, we compute the DFA look-up table using the following algorithm described in class:

```
procedure CONSTRUCTDFA(P[0..m))
    for c ∈ Σ do
        δ[0][c] ← 0
    end for
    δ[0][P[0]] ← 1
    x ← 0
    for q = 1, 2, ..., m − 1 do
        for c ∈ Σ do
            δ[q][c] ← δ[x][c]
        end for
        δ[q][P[q]] ← q + 1
        x ← δ[x][P[q]]
    end for
end procedure
```

Applying this procedure to P gives the following table:

	0	1	2	3	4	5	6	7	8	9
A	1	1	3	1	5	1	7	1	9	9
B	0	2	0	2	0	2	0	8	0	9
C	0	0	0	4	0	0	0	0	0	9
D	0	0	0	0	0	6	0	0	0	9

From the look-up table it is easier to draw the associated DFA diagram.



If we apply this DFA to the text T we obtain the following sequence of states:

```
[1, 2, 3, 2, 3, 4, 5, 2, 3, 2, 3, 4, 5, 6, 0,
 1, 2, 3, 2, 3, 4, 5, 6, 7, 8, 9, 9, 9, 9, 9]
```

□