

# Sample Solution for Problem Set 12

Data Structures and Algorithms, Fall 2020

January 4, 2021

## Contents

<b>1</b>	<b>Problem 1</b>	<b>2</b>
<b>2</b>	<b>Problem 2</b>	<b>3</b>
<b>3</b>	<b>Problem 3</b>	<b>4</b>
<b>4</b>	<b>Problem 4</b>	<b>5</b>
<b>5</b>	<b>Problem 5</b>	<b>6</b>
5.1	$L_1 \cup L_2$ . . . . .	6
5.2	$L_1 \cap L_2$ . . . . .	6
5.3	$L_1 L_2$ . . . . .	6
5.4	$\overline{L_1}$ . . . . .	6
5.5	$L_1^*$ . . . . .	7
<b>6</b>	<b>Problem 6</b>	<b>7</b>
<b>7</b>	<b>Problem 7</b>	<b>9</b>

## 1 Problem 1

**Algorithm Overview:** Use dynamic programming. Let  $dp[i][j] = True$  means we can parenthesize  $S[i...j]$  to make the value be  $True$ . Then we have the transition function:

$$dp[i][j] = \bigvee_{i \leq k \leq j, S[k] \in \{\wedge, \vee, \oplus\}} dp[i][k-1] S[k] dp[k+1][j] \quad (1)$$

**Implementation:** We use a simple down-top dynamic programming. Initialize  $dp[i][i] = S[i]$  for all even  $i$ . Then do the following loop

- For  $k = 2, 4, 6, 8, \dots$  until  $k = 2n$ , do
  - For  $i = 0, 2, 4, 6, \dots$  until  $i + k = 2n$ , let  $j = i + k$ , do
    - \*  $dp[i][j] = \bigvee_{i \leq k \leq j, S[k] \in \{\wedge, \vee, \oplus\}} dp[i][k-1] S[k] dp[k+1][j]$

**Complexity:**  $O(n^3)$ . Recall that the complexity of dynamic programming is the number of states ( $O(n^2)$ ) times the complexity for transition function ( $O(n)$ ).

## 2 Problem 2

Let  $f_{max}(i)$  represent the largest product in the subarrays ending with  $i$ , and  $f_{min}(i)$  represent the smallest product in the subarrays ending with  $i$ .

---

**Algorithm 1:** LARGESTPRODUCT( $A, i, j$ )

---

```
1  $f_{max} = f_{min} = ans = 1$ ;  
2 for  $i = 1$  to  $n$  do  
3    $temp = f_{max}$ ;  
4    $f_{max} = \max(A[i], f_{max} * A[i], f_{min} * A[i])$ ;  
5    $f_{min} = \min(A[i], temp * A[i], f_{min} * A[i])$ ;  
6    $ans = \max(ans, f_{max})$  ;  
7 end  
8 return  $ans$ ;
```

---

### 3 Problem 3

First sort all the points based on their  $x$  coordinate. To index our subproblem, we will give the rightmost point for both the path going to the left and the path going to the right. Then, we have that the desired result will be the subproblem indexed by  $v$ , where  $v$  is the rightmost point. Suppose by symmetry that we are further along on the left-going path, that the leftmost path is going to the  $i$ th one and the right going path is going until the  $j$ th one. Then, if we have that  $i > j + 1$ , then we have that the cost must be the distance from the  $i - 1$ st point to the  $i$ th plus the solution to the subproblem obtained where we replace  $i$  with  $i - 1$ . There can be at most  $O(n^2)$  of these subproblem, but solving them only requires considering a constant number of cases. The other possibility for a subproblem is that  $j \leq i \leq j + 1$ . In this case, we consider for every  $k$  from 1 to  $j$  the subproblem where we replace  $i$  with  $k$  plus the cost from  $k$ th point to the  $i$ th point and take the minimum over all of them. This case requires considering  $O(n)$  things, but there are only  $O(n)$  such cases. So, the final runtime is  $O(n^2)$ .

## 4 Problem 4

(a) To index the subproblem, we will give the first  $k$  items from the item list and the maximum weight of current knapsack. There can be at most  $O(nW)$  of these subproblem. For each item, we can decide to put it in or not. If we put the  $i$ st item in, we need to prepare  $w_i$  weight for it and earn  $v_i$  value. The value is  $v_i$  plus the solution to the subproblem with the first  $i - 1$  items, and  $W - w_i$  capacity. Otherwise, the value is as same as the solution to the subproblem with the first  $i - 1$  items, and  $W$  capacity. We take the maximum value from the two choices for each item.

(b) No since  $W$  can increase exponential as input length.

## 5 Problem 5

From the definition, there is an algorithm  $A_1$  solves  $L_1$  in  $O(n^{k_1})$ , and  $A_2$  solves  $L_2$  in  $O(n^{k_2})$ .

### 5.1 $L_1 \cup L_2$

---

**Algorithm 2:**  $A_3(x)$ 

---

```
1 if  $A_1(x)$  then
2   | return true;
3 end
4 if  $A_2(x)$  then
5   | return true;
6 end
7 return false;
```

---

$A_3$  solves  $L_1 \cup L_2$  in  $O(n^{\max(k_1, k_2)})$ , so  $L_1 \cup L_2 \in P$ .

### 5.2 $L_1 \cap L_2$

---

**Algorithm 3:**  $A_4(x)$ 

---

```
1 if  $A_1(x)$  then
2   | if  $A_2(x)$  then
3     | return true;
4   | end
5 end
6 return false;
```

---

$A_4$  solves  $L_1 \cap L_2$  in  $O(n^{\max(k_1, k_2)})$ , so  $L_1 \cap L_2 \in P$ .

### 5.3 $L_1 L_2$

---

**Algorithm 4:**  $A_5(x)$ 

---

```
1 for  $i = 1$  to  $(n - 1)$  do
2   | if  $A_1(x_1 \dots x_i)$  then
3     | if  $A_2(x_{i+1} \dots x_n)$  then
4       | return true;
5     | end
6   | end
7 end
8 return false;
```

---

$A_5$  solves  $L_1 L_2$  in  $O(n^{\max(k_1, k_2)+1})$ , so  $L_1 L_2 \in P$ .

### 5.4 $\overline{L_1}$

$A_6$  solves  $\overline{L_1}$  in  $O(n^{k_1})$ , so  $\overline{L_1} \in P$ .

---

**Algorithm 5:**  $A_6(x)$ 

---

```
1 if  $A_1(x)$  then  
2   | return false;  
3 end  
4 return true;
```

---

## 5.5 $L_1^*$

Let  $f[i][j]$  indicate whether  $x_i \dots x_j$  is belong to  $L_1^*$ .

$\forall x \in L_1^*, x \in L_1$  or  $\exists k, x_1 \dots x_k \in L_1^* \wedge x_{k+1} \dots x_n \in L_1^*$ .

Therefore, we can construct a dynamic programming algorithm  $A_7$ .  $A_7$  solves  $L_1^*$  in  $O(n^{k_1+2})$ , so  $L_1^* \in P$ .

---

**Algorithm 6:**  $A_7(x)$ 

---

```
1 for  $i = 1$  to  $n$  do  
2   | for  $j = i + 1$  to  $n$  do  
3     |  $f[i][j] = \text{false};$   
4   | end  
5 end  
6 for  $i = 1$  to  $n$  do  
7   | for  $j = i$  to  $n$  do  
8     | if  $A_1(x_i \dots x_j)$  then  
9       |  $f[i][j] = \text{true};$   
10    | end  
11   | end  
12 end  
13 for  $i = 1$  to  $n$  do  
14   | for  $j = i + 1$  to  $n$  do  
15     | for  $k = i$  to  $(j - 1)$  do  
16       | if  $f[i][k]$  and  $f[k + 1][j]$  then  
17         |  $f[i][j] = \text{true};$   
18       | end  
19     | end  
20   | end  
21 end  
22 return  $f[1][n];$ 
```

---

## 6 Problem 6

(a)

Consider (deterministic polynomial time) Turing Machine  $M$  defined as follows: Let  $G_1 = (V_1, E_1)$ ,  $G_2 = (V_2, E_2)$ , Turing Machine  $M$  accept input  $(G_1, G_2, \pi)$  if and only if  $\pi$  is a bijection between  $G_1$  and  $G_2$ , and  $\pi(u)\pi(v) \in E_2$  if and only if  $uv \in E_1$ .

**(b)**

Note that  $\overline{TAUTOLOGY}$  is the language of boolean formulas which can be false for some input.



## 7 Problem 7

(a)

for any language  $L \in P$ , there exists a (deterministic polynomial time) Turing Machine  $M$  defined as follows:  $M(x) = 1$  if and only if  $x \in L$ . Therefore, we may define a (deterministic polynomial time) Turing Machine  $M'$  in the following way.

- for any  $x$  and proof  $w$ ,  $M'(x, w) = 1 - M(x)$ .

It can be seen that  $\bar{L} \in NP$ . Hence,  $L \in coNP$ .

(b)

If not,  $L \in NP \Rightarrow L \in P \Rightarrow \bar{L} \in P \Rightarrow \bar{L} \in NP \Rightarrow L \in coNP$ . Similarly, we have  $L \in coNP \Rightarrow L \in NP$ , leading to contradiction.