CS260: ASSIGNMENT 1

Hao Liu(185896) Mohamed Bouaziz(184732) Tongzhou Gu(186085) Juyi Lin(187176)
Jinjie Mai(179794)
{hao.liu, mohamed.bouaziz, tongzhou.gu, juyi.lin, jinjie.mai}@kaust.edu.sa

1 Problem 1

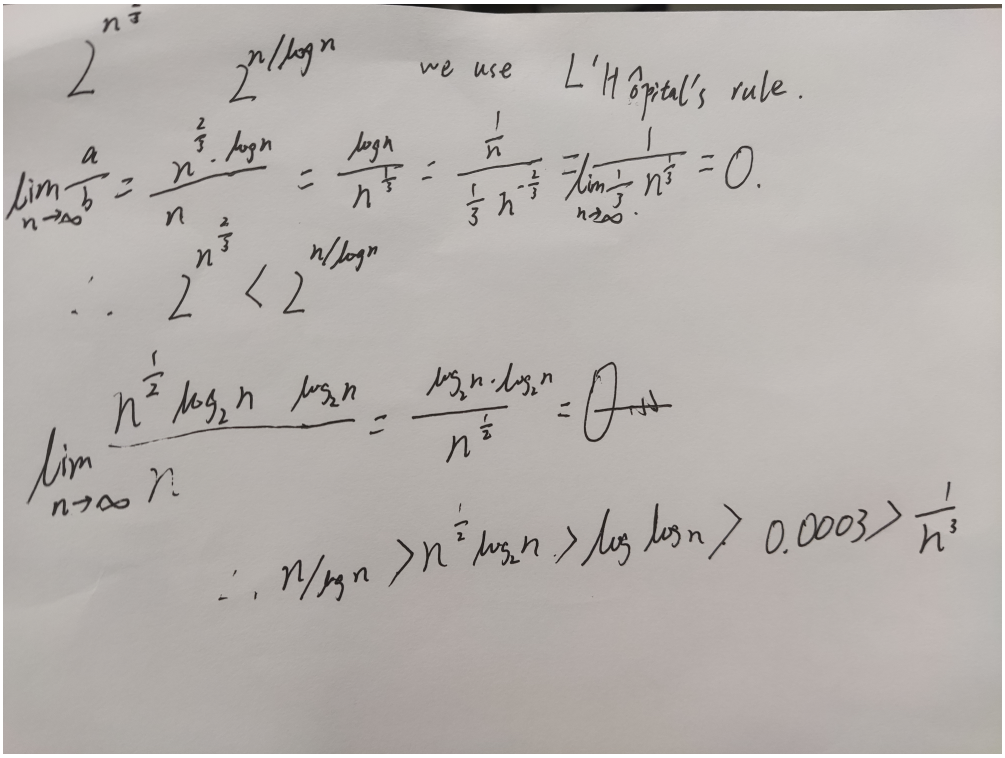

Figure 1: problem 1

$$n^n > (\sqrt{n}^n) > 2^{(n/logn)} > 2^{(n^{2/3})} > 10^{-6}n^7 > n/logn > \sqrt{n}logn > loglogn > 0.0003 > 1/n^3$$

2 Problem 2

See figure 2.

3 Problem 3

7
7
0 7
8
0 7 8
28
0 7 8 28
28
7 28
0 7 8 28
28
7 28
0 7 8 16 28
28
7 28
0 5 7 8 16 28
56
7 56
0 5 7 8 16 28 56
56
7 16 56
0 5 7 8 16 28 56

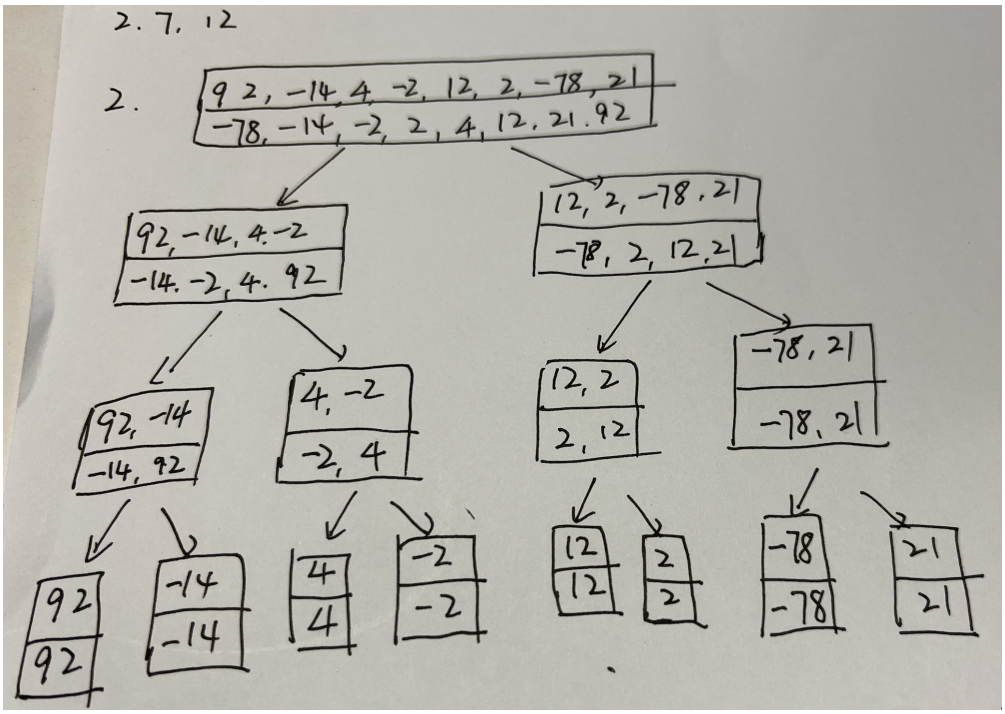

Figure 2: problem 2

4 Problem 4

- **First case** $f(n) = O(g(n))$**, is** $\log_a f(n) = O(\log_a g(n))$ **for** $a > 1$ **?**
  $\Rightarrow$ False
  Proof
  let $(n) = 2, \quad g(n) = 0,5$
  We have $f(n) = O(g(n))$ because $2 \leq 6 \times 0.5$ and therefore $\exists C = 6 > 0, f(n) \leq Cg(n)$
  For simplicity, let $a = e$ so that $log_a = log$
  We have $log(2) > 0$ since $2 > 1$ therefore $log(f(n)) > 0 \; \forall n$
  and $log(0.5) < 0$ since $0.5 < 1$ therefore $log(g(n)) < 0 \; \forall n$
  Therefore $log(g(n)) < 0 < log(f(n)) \; \forall n$
  Therefore $(\forall C > 0, Clog(g(n)) < 0 < log(f(n))) \; \forall n$
  Therefore $((\forall C > 0, Clog(g(n)) < log(f(n))) \; \forall n)$
  Which is the negation of the statement $((\exists C > 0, log(f(n)) < Clog(f(n))) \; \forall n)$
- **Second case** $f = O(g(n))$**, is** $2^{f(n)} = O\left(2^{g(n)}\right)$ **?**
  $\Rightarrow$ False
  Proof
  let $(n) = \log_2(n), \quad g(n) = \log_2(\sqrt{n})$
  We have $g(n) = \log_2(\sqrt{n}) = \frac{1}{2}\log_2(n)$
  For $C = 4$, $\log_2(n) \leq C\frac{1}{2}\log_2(n)$
  Therefore $\exists C > 0, f(n) \leq Cg(n)$ and accordingly $f(n) = O(g(n))$
  However, $\frac{2^{f(n)}}{2^{g(n)}} = \frac{2^{\log_2(n)}}{2^{\log_2(\sqrt{n})}} = \frac{n}{\sqrt{n}} = \sqrt{n}_{n \to +\infty} \to +\infty$
  Therefore, in this case, $\nexists C'$ such that $2^{f(n)} \leq C' 2^{g(n)}$
- **Third case** $f = O(g(n))$**, is** $\sqrt{f(n)} = O\left(\sqrt{g(n)}\right)$ **?**
  $\Rightarrow$ True

Proof

$f(n) = O(g(n)) \Rightarrow \exists C, f(n) \leq Cg(n)$

$\Rightarrow \exists C, \sqrt{f(n)} \leq \sqrt{Cg(n)}$ because $\sqrt{\cdot}$ is non-descending

$\Rightarrow \exists C, \sqrt{f(n)} \leq \sqrt{C}\sqrt{g(n)}$

let $C' = \sqrt{C}$

$\Rightarrow \exists C', \sqrt{f(n)} \leq C'\sqrt{g(n)}$

$\Rightarrow \sqrt{f(n)} = O\left(\sqrt{g(n)}\right)$

5 Problem 5

Considering that for any $a_i \in \{a_1, a_2, ..., a_n\}$, we need to find whether there exists a $a_j \in \{a_1, a_2, ..., a_n\}$ that equals to $a_i + x$ or $a_i - x$. However, we cannot directly apply the binary search algorithm because it is not guaranteed that the array $a_1, a_2, ..., a_n$ is ordered. Thus, sorting before searching could be one solution. Here is the pseudo-code of the whole algorithm 1.

Algorithm 1 The algorithm of Problem 5

```
procedure ALGORITHM(A, x)
    B ← Sorted A with Merge Sort in ascending order
    n ← Length of A
    for each element a ∈ A do
        if Search(B, a + x, 0, n) = Exist then
            return Exist
        else if Search(B, a − x, 0, n) = Exist then
            return Exist
        end if
    end for
    return NotExist
end procedure

procedure SEARCH(A, a, lower, upper)
    while lower ≤ upper do
        mid ← (lower + upper)/2
        if A[mid] = a then
            return Exist
        else if A[mid] > a then
            upper ← mid − 1
        else if A[mid] < a then
            lower ← mid + 1
        end if
    end while
    return NotExist
end procedure
```

Since the time complexity of the binary search and the merge sort algorithm is $O(\log n)$ and $O(n \log n)$ respectively, and we need to perform sorting once and searching n times, thus the overall time complexity of our algorithm is $O(n \log n) + n \cdot O(\log n) = O(n \log n)$.

6 Problem 6

The section is following:

```
n = 10
C = [0] * 5
B = [1] * 5
A =  C+B
```

```
def solution6a( arr):
    i = 0
    j = n-1
    while i <= j :
        mid = (i +j)//2
        if arr[mid] == 1 :
            j = mid - 1
        elif arr[mid] == 0 and arr[mid +1] == 1:
            return mid
        else:
            i = mid +1
    return i
print(solution6a(A))
```

The result is 4. Justify the time complexity of the algorithm :

$T(n) = T(n/2) + O(1)$

Using Master's theorem case 2 to calculate

$f(n) = \theta(n^{log(1)}(log(n))^0)$

so we have $T(n) = \theta(n^{log(1)}(log(n))^1) = \theta(logn)$

The b section is following:

```
B2 = [1] * 5000
A = C+B2
def solution6b( arr):
    i = 0
    step = 1
    dir = 1
    while arr[i] == 0:
        i = i + dir * step
        if arr[i+1] == 1 and arr[i] == 0:
            return i
        elif arr[i+1] == 0 and arr[i] == 0:
            step *=2
    dir = -1
    step//=2
    while not ( arr[i+1] == 1 and arr[i] == 0):
        i = i + dir * step
        if arr[i+1] == 1 and arr[i] == 0:
            return i
        elif arr[i+1] == 0 and arr[i] == 0:
            step //=2
            dir = 1
        elif arr[i] == 1:
            dir = -1
            step //=2
    return i
print(solution6b(A))
```

The result is 4.

7 Problem 7

$176 \times 3018 =$

$(10^2 \times 1 + 76) \times (10^2 \times 30 + 18) = 10^4 \times 1 \times 30 + 10^2 (1 \times 18 + 76 \times 30) + 76 \times 18$

$X_L Y_R + X_R Y_L = 1 \times 18 + 76 \times 30 = (1+76)(30+18) - 1 \times 30 - 76 \times 18$

We need to calculate 1×30, 76×18, 77×48

① $1 \times 30 =$

$(10 \times 0 + 1)(10 \times 3 + 0) = 10^2 \times 3 \times 0 + 10(0 \times 0 + 3) + 1 \times 0$

$X_L Y_R + X_R Y_L = 0 \times 0 + 3 \times 1 = (1+0)(3+0) - 1 \times 0 - 3 \times 0 = 3$

Calculate $1 \times 0 = 0$, $3 \times 0 = 0$, $1 \times 3 = 3$

So $1 \times 30 = 30$

② $76 \times 18 =$

$(10 \times 7 + 6)(10 \times 1 + 8) = 10^2 \times 1 \times 7 + 10(7 \times 8 + 1 \times 6) + 6 \times 8$

$(7 \times 8 + 1 \times 6) = (7+6)(1+8) - 7 \times 1 - 6 \times 8$

Calculate $7 \times 1 = 7$, $6 \times 8 = 48$, 13×9

$(7+6)(1+8) = 13 \times 9 = (10 \times 1 + 3)(10 \times 0 + 9) = 10 \times 1 \times 0 + 10(9 \times 1 + 3 \times 0) + 9 \times 3$

$9 \times 1 + 3 \times 0 = (1+3)(0+9) - 1 \times 0 - 9 \times 3$

$1 \times 0 = 0$ $\quad 9 \times 3 = 27$ $\quad$ So $13 \times 9 = 117$

So $76 \times 18 = 1368$

③ $77\times48=(10\times7+7)(10\times4+8)=10^2\times7\times4+10(7\times8+4\times7)+7\times8$

$7\times8+4\times7=(7+7)(4+8)-7\times4-7\times8$

We need to calculate $7\times4=28$ $7\times8=56$ 14×12

$14\times12=(10\times1+4)(10\times1+2)=10^2\times1\times1+10(1\times2+1\times4)+4\times2$

$1\times2+1\times4=(4+1)(2+1)-4\times2-1\times1$.

calculate 4×2 1×1 5×3

so $14\times12=168$

So $77\times48=3696$

So $176\times3018=10^4\times30+10^2\times(3696-30-1368)+1368$

$=531168$

8 Problem 8

$A = \begin{pmatrix} 1 & 3 \\ 2 & 1 \end{pmatrix}$, $B = \begin{pmatrix} 4 & 1 \\ 2 & 0 \end{pmatrix}$

Let $C = AB = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$

Then we have:

$D_1 = (1+1)(4+0) = 8$

$D_2 = (-1+2)(4+1) = 5$

$D_3 = (3-1)(2+0) = 4$

$D_4 = (*(1-0) = 1$

$D_5 = (1+3)*0 = 0$

$D_6 = (*(-4+2) = -2$

$D_7 = (2+1)*4 = 12$

Finally we can solve:

$C_{11} = D_1 + D_3 - D_5 + D_6 = 8+4-0-2 = 10$

$C_{12} = D_4 + D_5 = 1$

$C_{21} = D_6 + D_7 = 10$

$C_{22} = D_1 + D_2 + D_4 - D_7 = 8+5+1-12 = 2$

In the end, we have $C = \begin{pmatrix} 10 & 1 \\ 10 & 2 \end{pmatrix}$, 7 multiplications.

9 Problem 9

Algorithm 2 Algorithm of problem 9

```
1: input : G = (V, E)
2: G_sc ← strongly_connected_components(G)
3: G_D ← meta_graph(G, G_sc)
4: T ← topological_order(G_D)
5: for i ∈ {1, ..., |T| − 1} do
6:     if (T[i], T[i + 1]) ∉ E_D then
7:         return False
8:     end if
9: end for
10: return True
```

We start by the very simple case where we assume that the input graph is a DAG (directed acyclic graph). We know that for a DAG, we can make a topological ordering, which is a process of complexity $O(|V| + |E|)$ (from the Graph lecture presentation slides 21-24). We also know that if for two successive vertices in the topological order there is no edge that connects them, then they could not be connected in different way. The complexity of iterating through all the vertices in the topological order and check if there is an edge that connects them is $O(|V| + |E|)$ as all vertices will be visited once and each edges will be visited once (we have a DAG and therefore there is no cycle and so either there is an edge from a vertex to the other or it is impossible to find another path). This case could be seen in Algorithm 2 from line 4 to 9. Line 4 makes the topological order of a DAG and the loop in Lines 5-9 performs the iterations.

In reality we want this to be applicable for all directed graphs that are not necessarily DAGs. Therefore, in order to transform a directed graph into a DAG on which we can apply the previous process, we start by extracting its strongly connected components (as shown in Line 1) as given in the Graph lecture presentation slides 16-17. This extraction is of complexity $O(|V| + |E|)$ as given in the same presentation slide 15. After that, the strongly connected components will be used in Line 3 to construct a meta-graph of the input graph where the strongly connected components are represented in one node. The resulting meta-graph is a DAG (slide 19) and can be fed as an input to the simple case treated in the last paragraph. The meta-graph construction is of complexity $O(|V| + |E|)$ as it updates all the edges at most once and visits the vertices once for each. To finally link the two parts, if two nodes from the meta graph are not connected via a path then their elements are not connected via path and therefore we return by a false.

Overall, the complexity is $O(|V| + |E|)$ for this algorithm as all its separate steps are $O(|V| + |E|)$.

10 Problem 10

See figure 3 and 4.

Note that for Graph 2 we have the other two topological orderings:

$$d, a, c, b, g, h, i, j, e, f$$
$$d, c, a, b, g, h, i, j, e, f$$

We didn't the show the full procedure because of space limit.

11 Problem 11

Firstly, we get the reverse graph.Secondly, we choose ‘a‘ as the first node.

In origin graph: L0 is ‘a‘, L1 is ‘b‘ and ‘d‘ , L2 is ‘c‘ and ‘g‘, L3 is ‘e‘ and ‘f‘

In reverse graph: L0 is ‘a‘, and then end.

The overlap between the nodes found by two trees is only ‘a‘, so a is a strongly connected component.

Then we choose another node except ‘a‘ to do a tree search. We choose ‘b‘.

In the origin graph: L0 is ‘b‘ L1 is ‘g‘ , L2 is ‘e‘ , L3 is ‘c‘ and ‘f‘

In reverse graph: L0 is ‘b‘ , L1 is ‘c‘ and ‘d‘ , L2 is ‘e‘, L3 is ‘g‘, L4 is ‘b‘ and it is end .

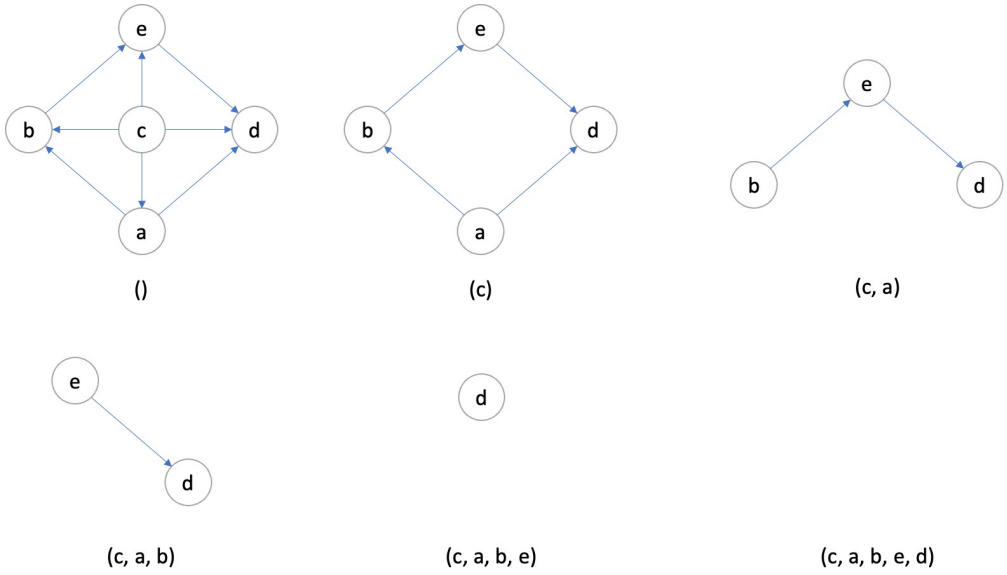

Figure 3: Problem 10.1

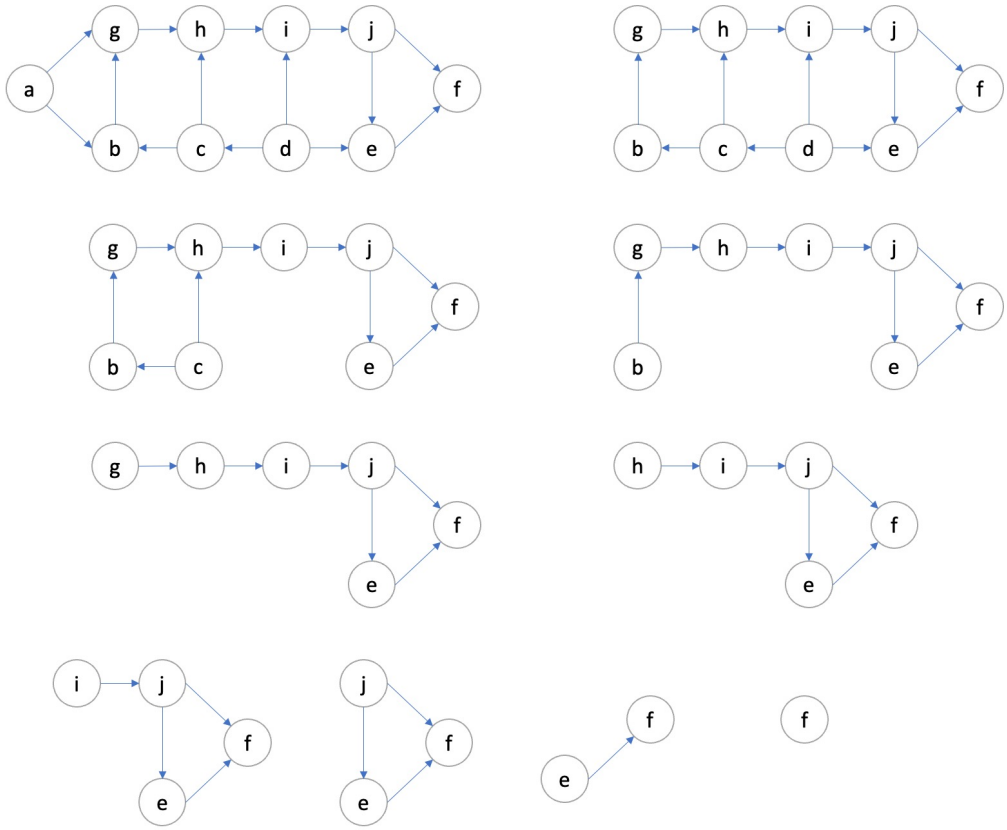

Figure 4: Problem 10.2

The overlap between the nodes found by two trees is e g b c, so e g b c is a strongly connected component.
Then we move to ‘d‘ , which is not in any strongly connected components.
In the origin graph: L0 is ‘d‘ L1 is ‘c‘, L2 is ‘g‘
In reverse graph: L0 is ‘d‘ L1 is ‘a‘
The overlap between the nodes found by two trees is d, so d is a strongly connected component.
Then we move to ‘f‘ , which is not in any strongly connected components.
In the origin graph: L0 is ‘f‘
In reverse graph: L0 is ‘f‘ L1 is ‘c‘ and ‘e‘ , L2 is ‘g‘ and ‘d‘
The overlap between the nodes found by two trees is f, so f is a strongly connected component.
Overall, the strongly connected components are a, e g b c, d and f
reference: https://stackoverflow.com/questions/53437095/strongly-connected-components-with-bfs
one topo bfs, vistied , and then reverse graph https://oi-wiki.org/graph/scc/
https://leetcode.cn/submissions/detail/189908577/

12 Problem 12

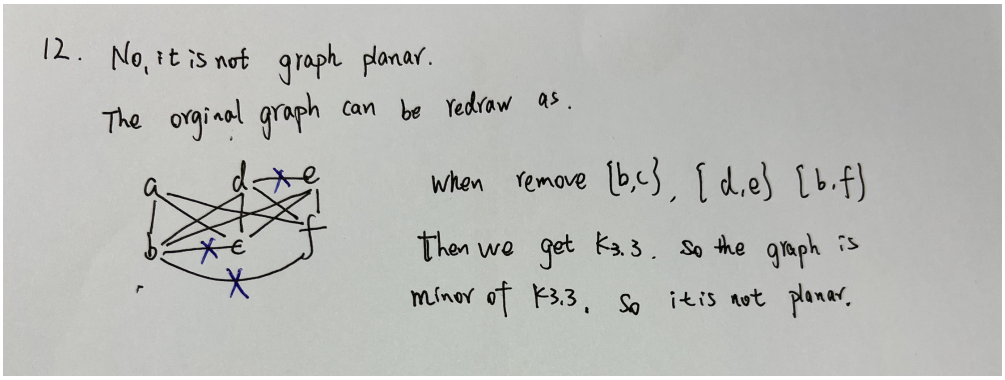

Figure 5: problem 12

13 Problem 13

Chain Matrix Multiplication :

M_1	M_2	M_3	M_4	$M_1 \times M_2 \times M_3 \times M_4$
1×3	3×2	2×4	4×6	

$m_0 \times m_1$ $m_1 \times m_2$ $m_2 \times m_3$ $m_3 \times m_4$

From the slides we have :

$$C(i,j) = \min_{i \le k < j} \{ C(i,k) + C(k+1,j) + m_{i-1} m_k m_j \}$$

And then :

$$C(1,1) = C(2,2) = C(3,3) = C(4,4) = 0$$

$$C(1,2) = C(1,1) + C(2,2) + m_0 m_1 m_2 = 0 + 0 + 1 \times 3 \times 2 = 6$$

$$C(2,3) = C(2,2) + C(3,3) + m_1 m_2 m_3 = 0 + 0 + 3 \times 2 \times 4 = 24$$

$$C(3,4) = C(3,3) + C(4,4) + m_2 m_3 m_4 = 0 + 0 + 2 \times 4 \times 6 = 48$$

$$C(1,3)_{k=1,2} = \min \begin{cases} C(1,1) + C(2,3) + m_0 m_1 m_3 = 0 + 24 + 1 \times 3 \times 4 = 36 \\ C(2,3) + C(4,4) + m_1 m_3 m_4 = 6 + 0 + 1 \times 2 \times 4 = 14 \end{cases}$$

$$= 14$$

$$C(2,4)_{k=2,3} = \min \begin{cases} C(2,2) + C(3,4) + m_1 m_2 m_4 = 0 + 48 + 3 \times 2 \times 6 = 84 \\ C(2,3) + C(4,4) + m_1 m_3 m_4 = 24 + 0 + 3 \times 4 \times 6 = 96 \end{cases}$$

$$= 84$$

$$C_{(1,4)}{}_{k=1,2,3} = \min \begin{cases} C_{(1,3)} + C_{(4,4)} + m_0 m_3 m_4 = 14 + 0 + 1 \times 4 \times 6 = 38 \\ C_{(1,2)} + C_{(3,4)} + m_0 m_2 m_4 = 6 + 48 + 1 \times 2 \times 6 = 66 \\ C_{(1,1)} + C_{(2,4)} + m_0 m_1 m_4 = 0 + 84 + 1 \times 3 \times 6 = 102 \end{cases}$$

$$= 38$$

So we select $C_{(1,4)} = C(1,3) + C(4,4) \cdots$

$$= C(1,2) + C(3,3) + C(4,4) \cdots$$

The minimum cost is 38 times of multiplication.

And the optimal parenthesization is:

$$(C(M_1 \times M_2) \times M_3) \times M_4$$

14 Problem 14

We construct the graph G that describs teh set S that encloses all the subarrays of {1,3,-3,1,2}.

$$S_1 = \{1\}$$
$$S_2 = \{1, 3; 3\}$$
$$S_3 = \{1, 3, -3; 3, -3; -3\}$$
$$S_4 = \{1, 3, -3, 1; 3, -3, 1; -3, 1; 1\}$$
$$S_5 = \{1, 3, -3, 1, 2; 3, -3, 1, 2; -3, 1, 2; 1, 2; 2\}$$

We have $S = S_1 \cup \ldots \cup S_5$

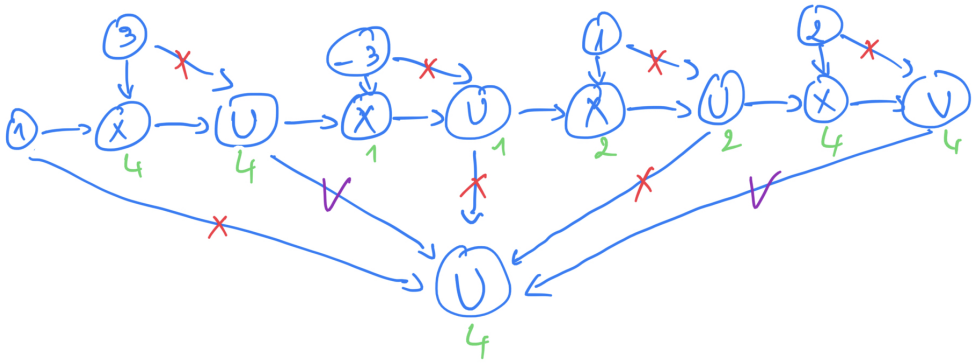

Figure 6: Optimization for sum

In Figure 6, we see that the optimization relative to sum for the array {1,3,3,1,2} gives that the subarrays {1,3} and {1,3, 3,1,2} are those with the maximum sum.

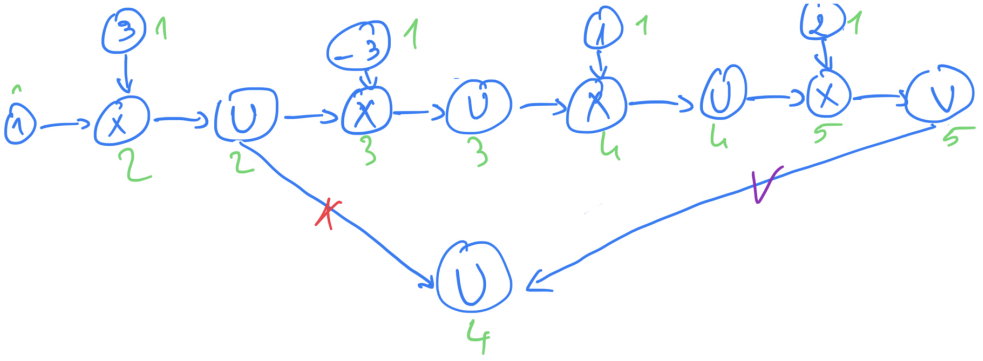

Figure 7: Optimization for length

In Figure 7, we see that the optimization relative to length for the array {1,3,3,1,2}, the subarray {1,3, 3,1,2} is the contiguous one with maximum length.

15 Problem 15

Assume $a_1, a_2, ..., a_n$ is the input sequence, and $L(i), 1 \leq i \leq n$ represents the length of longest monotonically increasing subsequence of $a_1, a_2, ..., a_i$. Then we will have,

$$L(i) = max \begin{cases} L(j) + 1 & a_j < a_i, 1 \leq j < i \leq n \\ L(j) & a_j \geq a_i, 1 \leq j < i \leq n \\ 1 & i = 1 \end{cases} \tag{1}$$

For $i = 1$, $L(1) = max(1)$. 1 comparison.

For $i = 2$, we need to access a_j and $L(j), j = 1$ and make 1 comparison.

For $i = n$, we need to access a_j and $L(j), 1 \leq j < n$ and make $n - 1$ comparisons.

Thus, the number of comparisons in total will be

$$1 + (1 + 2 + 3 + ... + n - 1) = \frac{n^2 - n + 2}{2} \tag{2}$$

So the time complexity is $O(n^2)$.

16 Problem 16

Firstly, we initialize the edit distance matrix for dynamic programming. Then we have transform equation:

$dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1] - 1)$if same character

$dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])$ if different character

i is the index in word1, j is the index in word2

```
str1 = "STATIONARY"
str2 = "TERTIARY"

def minDistance(word1: str, word2: str) -> int:
    n1, n2 = len(word1), len(word2)
    dp = [[0 for i in range(n2 + 1)]for j in range(n1+1)]
    for i in range(n2+1):
        dp[0][i] = i
    for i in range(n1+1):
        dp[i][0] = i

    for i in range(1, n1+1):
        for j in range(1, n2+1):
            if word1[i-1] == word2[j-1]:
                dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1] - 1)
            else:
                dp[i][j] = 1 + min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])
    return dp[n1][n2]

print(minDistance(str1,str2))
```

the answer is 5