CS205: Design and analysis of algorithms

Mid-semester examination • Maximum marks: 30

Spring 2023

Notes

- Instructor/TA will not be available for clearing doubts during the exam.
- If any assumptions are made, state them and justify.
- There are 5 questions in this question paper. Each carries 6 marks.
- If it is helpful for your answers, you can use the results we obtained in the lectures, without repeating the proofs discussed in the lectures.
- You are allowed to carry hand-written materials, but no printed materials and electronic gadgets are allowed.

Q1: Let EvenNum be the problem defined as follows: Given an integer k find whether k is even or not. Let OddNum be the problem defined as follows: Given an integer k find whether k is odd or not. Find a polynomial-time reduction from EvenNum to OddNum.

Answer: Let k be an instance of EvenNum. Then return 1 (which is a yes-instance for OddNum). If k is odd, then return 2 (which is a no-instance of OddNum). There are many other reductions. Another one: Let k be an instance of EvenNum. Return $k + 1$ as an instance of OddNum.

Evaluation: 3 marks for a correct reduction. 3 marks for correctness proof.

Q2: We have discussed many variants of boolean satisfiability problems. One of the most important variant among them is 3-SAT, which is defined below.

3-SAT
Input: A formula Φ in Conjunctive Normal Form (CNF) such that each clause contains exactly three literals of distinct variables.
Question: Does there exist a truth assignment such that at least one literal per clause in Φ is TRUE?

We have learned that 3-SAT is NP-complete. Now, let us come up with a new problem involving boolean variables. For two positive integers p, q, we define the problem (p, q)-Set-SAT as follows:

(p, q)-Set-SAT
Input: A set Φ such that each element in Φ is a set (known as a clause) of p literals of distinct boolean variables.
Question: Does there exist a truth assignment of the boolean variables such that only at

most q literals per clause in Φ are FALSE?

For example, $\{\{x_1, \overline{x_2}, \overline{x_3}\}, \{\overline{x_1}, \overline{x_3}, x_4\}, \{x_1, \overline{x_3}, \overline{x_5}\}, \{x_3, x_4, x_5\}\}$ is a yes-instance of $(3,1)$-Set-SAT. This is because, an assignment of $\{x_1 \leftarrow T, x_2 \leftarrow F, x_3 \leftarrow F, x_4 \leftarrow T, x_5 \leftarrow T\}$ makes sure that each clause has only at most 1 FALSE literal. On the other hand, $\{\{x_1, \overline{x_2}, \overline{x_3}\}, \{\overline{x_1}, x_2, x_3\}\}$ is a no-instance of $(3,1)$-Set-SAT. This is because, no truth assignment can make sure that only at most 1 literal per clause is FALSE.

Obtain a polynomial-time algorithm for (4,2)-Set-SAT or prove that (4,2)-Set-SAT is NP-complete. If you are proving the NP-completeness, then the polynomial-time reduction that you come up with must be from 3-SAT.

Answer: The problem is clearly in NP. We give a reduction from 3-SAT, which proves that the problem is NP-complete. Let Φ be an instance of 3-SAT. Create a set Φ' from Φ, where each element (which is also a set) corresponds to a clause in Φ and contains the literals of the clause. Introduce a new variable y and add it to each set in Φ'. For example, if Φ is $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_3} \vee x_4) \wedge (x_1 \vee \overline{x_3} \vee \overline{x_5}) \wedge (x_3 \vee x_4 \vee x_5)$, then Φ' is $\{\{x_1, \overline{x_2}, \overline{x_3}, y\}, \{\overline{x_1}, \overline{x_3}, x_4, y\}, \{x_1, \overline{x_3}, \overline{x_5}, y\}, \{x_3, x_4, x_5, y\}\}$. If Φ is satisfiable, then by setting y to be TRUE, we can make sure that at least two literals in each set of Φ' is TRUE (therefore, only at most 2 literals per set is FALSE). If there is a truth assignment for variables in Φ' which sets only at most 2 literals per clause to be FALSE, then at least one literal per clause in Φ must be TRUE under the same truth assignment.

Evaluation: 3 marks for a correct reduction, 3 marks for the proof.

Q3: In the Clique problem, we are given with a graph G and an integer k, and the objective is to find whether G has a clique (mutually adjacent set of vertices) of size at least k or not. We have learned that the problem is NP-complete. Now, let's define a variant of it called Clique-Special: Given an integer k and a graph G which has a universal vertex (a universal vertex of a graph is a vertex adjacent to all other vertices in the graph), find whether G has a clique of size at least k or not. Note that there is only one difference between Clique and Clique-Special - The input domain of Clique is the set of all graph whereas the same for Clique-Special is constrained that every input graph for the problem has at least one universal vertex. Either prove that Clique-Special is NP-complete or come up with a polynomial-time algorithm to solve it.

Answer: Clearly, Clique-Special is in NP. We will give a polynomial-time reduction from Clique to Clique-Special, which proves that Clique-Special is also NP-complete. Let (G, k) be an instance of Clique. We add a universal vertex w to G to get a new graph G'. It can be proved that (G, k) is a yes-instance of Clique if and only if $(G', k+1)$ is a yes-instance of Clique-Special.

Evaluation: 3 marks for the reduction, 3 marks for the correctness proof.

Q4: Let A_1, A_2, A_3, A_4, A_5 be matrices with dimensions $3*4, 4*7, 7*2, 2*8, 8*4$ respectively.

1. Write down the recursive formulation for $m[i,j]$, where $m[i,j]$ is the minimum number of scalar multiplications required to obtain the product $A_i A_{i+1} \ldots A_j$.
2. Compute the table (for the matrices given above) where the entries are $m[i,j]$s.
3. What is the minimum number of scalar multiplications required to compute $A_1 A_2 A_3 A_4 A_5$?
4. What is the optimal parenthesization of $A_1 A_2 A_3 A_4 A_5$? Mention how you obtained the answer.

i/j	**1**	**2**	**3**	**4**	**5**
1	0	84	80	128	168
2	-	0	56	120	152
3	-	-	0	112	120
4	-	-	-	0	64
5	-	-	-	-	0

Table 1

Answer: 1. Assuming the dimensions of the matrices are given in an array p (such that A_i has $p[i-1]$ rows and $p[i]$ columns):

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p[i-1]p[k]p[j]\} & \text{otherwise} \end{cases}$$

2. See Table 1
3. 168
4. $((A_1(A_2A_3))(A_4A_5))$. This can be obtained while computing the table by seeing for which k $m[i,j]$ is optimized.

Evaluation : 0.5 marks for (1), 3 marks for the table (2) (there are 15 entries, so 0.2 marks per entry), 0.5 mark for (3), and 2 marks for (4) (1 mark for the correct parenthesization and 1 mark for explaining how you obtained it).

Q5: You are given with a number n. The objective is to find out the number of n-bit strings without any consecutive 1s. For example, if $n = 3$, then the value is 4 (3-bit strings without consecutive 1s are 000, 001, 100, 101). Do the following.

1. Come up with a recursive formulation for $c[n]$: the number of n-bit strings without consecutive 1s.
2. Design a dynamic programming algorithm to find $c[n]$ based on the recursive formulation.
3. Derive the worst-case running time of your algorithm.

Answer: The example given in the question misses 010. But the remaining of the question is unambiguous. The recursive formulation is

$$c[n] = \begin{cases} 2 & \text{if } n = 1 \\ 3 & \text{if } n = 2 \\ c[n-1] + c[n-2] & \text{otherwise} \end{cases}$$

The algorithm is similar to that of finding n^{th} Fibonacci number. The algorithm runs in time $O(n)$ time.

Evaluation: 3 marks for the formulation, 2 marks for the algorithm, and 1 mark for deriving the worst-case time-complexity.