CS 5300 Advanced Algorithm Design and Analysis
Final Exam
Posted on Canvas: Class Time 12/7/2023
Due: 2:50PM, 12/14/2023
(Submission to Canvas)

Name : _Shreyas Chaudhary_

Last 4 digits of your Student ID #: ___0813_________________

Read these instructions before proceeding.

- Your answers should be typed on 8.5 x 11 inches white paper, double-spaced. Use this page as your cover page. Write your name and the last 4 digits of your Student ID number at the top right. After grading, your name and the last 4 digits of your Student ID number will be matched so that your grade can be correctly recorded.
- Answer to each question is **limited to be one to two pages**.
- You need to turn in your answers for this Final-Exam on canvas.cpp.edu no later than **2:50PM, 12/14/2023.**
- No questions will be answered for this exam by emails or during my office hours.
- Answer each question the best you can. Partial credit will be awarded for reasonable efforts. If a question contains an ambiguity or a misprint, then say so in your answer, providing the answer to a reasonable interpretation of the question; give your assumptions.
- Open book. Open notes. Open to any source. If your answer is based on a particular article or a source, you need to understand it completely and write the answer on your own. Also remember to quote it properly and stick to a style, e.g. IEEE or MLA.
- There are six questions. The total is 100 points.
- Good luck!

Q#1 (16)	Q#2 (16)	Q#3 (15)	Q#4 (22)	Q#5 (16)	Q#6 (15)	Total (out of 100)

ANSWERS – Final Exam

CS-5300 – Advanced Algorithm Design and Analysis

Ans 1:

A)**True**. All NP-complete problems are NP-hard.

This is by definition that a problem is NP-hard if every problem in NP can be reduced to it in polynomial time. NP-complete problems are a subset of NP-hard problems that are also in NP, meaning they can be verified in polynomial time. Thus, by definition, if a problem is NP-complete, it is also NP-hard because it meets the criteria of being at least as hard as the hardest problems in NP and it is also verifiable in polynomial time (hence NP).

2)**False**. All NP-hard problems are not NP-complete.

All NP-complete problems are NP-hard, but not all NP-hard problems are NP-complete. NP-complete problems are those that are both in NP (nondeterministic polynomial time) and NP-hard. This means that all solutions to NP-complete problems can be verified in polynomial time, and they are at least as hard as the hardest problems in NP. However, NP-hard problems are not necessarily in NP; they could be harder and not even have verifiable solutions in polynomial time.

3)**False**.

It is assumed that A is reducible to B and that A is an NP-complete problem. Although B can be reduced in polynomial time, this does not guarantee that it will become NP complete. It will be reduced to NP-hardness, though not necessarily to every issue class. Equation A can be expressed as A ≤P B, where ≤P stands for polynomial reduction. This involves the known issue A being NP-complete, which requires it to be both NP and NP-hard. The same is true for the unknown problem B. Regarding the other class of the problem, nothing can be said.

4)**True**

If a problem C is known to be NP-hard, and we can demonstrate that C can be reduced to another problem D in polynomial time, then that D is also NP-hard. This is because the ability to transform C into D efficiently proves that any algorithm that could solve D quickly (in polynomial time) could also be used to solve C quickly, which contradicts the definition of C being NP-hard. Therefore, if C cannot be solved in polynomial time (since it's NP-hard) and C can be reduced to D, then D also cannot be solved in polynomial time, making D NP-hard. A solution to the second problem would lead to a solution to the first when we state that one problem is reducible to another. If the first problem is NP-hard, then the second problem would fall into the same complexity class as the first and be solved as a result. For this reason, a common technique to demonstrate that the second problem is also NP-hard is to demonstrate polynomial-time reducibility from one NP-hard problem to another.

Ans 2:

A) To solve part the (a) of the question, we need to provide a nondeterministic polynomial time algorithm for the SUM-OF-SUBSETS problem. This can be done as follows:

Sample Algorithm: Nondeterministic SUM-OF-SUBSETS

```
Input: A set C = {c1, c2,..., cn} and a target sum M.
Output: "Yes" if a subset with sum M exists, "No" otherwise.
1. For each element Ci in C:
   a. Non deterministically guess if Ci should be included in the subset C'. Use Guess({0,1}) for this purpose.
2. Calculate the sum of the guessed subset C'.
3. If the sum of the subset C' equals M, return "Yes".
4. If no such subset is found, return "No".
```

Since each guess is independent and can be made in a constant amount of time. It can also be computed in a linear amount of time, the algorithm executes in polynomial time. It is nondeterministic because it makes a nondeterministic estimate at every stage, which a nondeterministic Turing machine may achieve. The procedure has a nondeterministic path to a "Yes" answer if there is a subset whose total is M. The SUM-OF-SUBSETS problem is therefore in NP. Also, since SUM-OF-SUBSETS is the result of reducing THREEZONE-PARTITION, and since SUM-OF-SUBSETS is in NP, we may deduce that SUM-OF-SUBSETS is NP-Complete if and only if the reduction is polynomial, which it is because it only requires linear operations on the set.

B) Here**,** we have to first define the transformation from the THREEZONE-PARTITION problem to the SUM-OF-SUBSETS problem and then provide the if-and-only-if proof.

Transformation:

Considering a case of the THREEZONE-PARTITION problem with a set Z = {z1, z2,..,Zn}, we first transform it into an SUM-OF-SUBSETS problem as follows:

1. Let set A = Z.
2. Calculate the total sum S of all elements in Z.
3. Set the target sum for the SUM-OF-SUBSETS problem to be M = frac ¾ S. If frac ¾ *S is not an integer, multiply each element of A and M by 4 to ensure completness, resulting in M = 3S.

If-and-only-if proof:

If Part:

If there exists a subset A' ⊆ A that sums to M, this means that there is a corresponding subset Z' ⊆ Z such that the sum of Z' = 3 * sum of (Z - Z'). This is true because the transformation scales the sum by the same factor for both Z' and Z - Z', maintaining the threefold relationship.

Only-if Part:

Conversely, if there is a subset Z' ⊆ Z in the original THREEZONE-PARTITION problem that satisfies Z' = 3 * sum of (Z - Z'), then the corresponding subset Z' ⊆ Z in the transformed SUM-OF-SUBSETS problem will sum to M. The transformation keeps the overall balance of sums because it equally increases the size of all elements.

Any solution to the SUM-OF-SUBSETS problem relates directly to a solution to the THREEZONE-PARTITION problem, and vice versa, as demonstrated by this if-and-only-if condition. Therefore, since we can transform any instance of the THREEZONE-PARTITION problem into an instance of the SUM-OF-SUBSETS problem in polynomial time, and solutions are equivalent, the SUM-OF-SUBSETS problem is NP-Complete if the THREEZONE-PARTITION problem is NP-Complete.

Ans 3: The most interesting question for me would be from greedy algorithms. Greedy algorithms are an easy approach to solving problems where we always choose the best option available right now, without worrying about how these choices might affect future decisions. However, it doesn't always lead to the best overall result in every scenario. For this final exam, I have decided to choose - **Design a Greedy Algorithm to Optimize Program Storage on Tapes.**

Describing the question:

Given `n` programs with varying lengths to be stored on a single tape, design a greedy algorithm that minimizes the Mean Retrieval Time (MRT) for accessing these programs. Assume the tape is read sequentially from the beginning.

1. **Algorithm Description**:

    1. Sort the programs in ascending order of their lengths.
    2. Store them on the tape in this sorted order.

2. **Why It's Interesting and the reason for choosing:**

    1. This problem demonstrates the elegance of greedy algorithms in solving complex optimization problems through simple, intuitive steps.
    2. It highlights the importance of problem formulation and understanding the characteristics of greedy algorithms (local optimal choices leading to global optimality).
    3. The problem is a classic example of real-world applications where optimization is crucial, such as in data storage and retrieval systems.

3. **Effectiveness:**

- The algorithm is efficient, with a time complexity of O(n log n) due to the sorting step.
- By storing shorter programs first, the time to access any program is minimized, as each program's retrieval time is proportional to the sum of lengths of all preceding programs on the tape.

This question merges algorithmic theory with real-world application, challenging us students to apply concepts to practical problems in computer science.

To solve the problem of optimizing program storage on tapes, let's consider an example:

Scenario:

Tape capacity: Unlimited (for simplicity in understanding. We can make it whatever we wish to)

Number of programs (n): 5

Lengths of programs: Program 1 - 5 units, Program 2 - 3 units, Program 3 - 2 units, Program 4 - 4 units, Program 5 - 1 unit

Steps for solving it:

1. **Sort the Programs by Length**:

In ascending order: Program 5 (1 unit), Program 3 (2 units), Program 2 (3 units), Program 4 (4 units), Program 1 (5 units).

2. **Store Programs on Tape in Sorted Order**:

Order on tape: 5, 3, 2, 4, 1.

3. **Calculate Mean Retrieval Time (MRT)**:

The Retrieval time for each program is the sum of its lengths and all preceding programs.

Program 5: 1 unit, Program 3: 1 + 2 = 3 units, Program 2: 1 + 2 + 3 = 6 units, etc.

MRT = (1 + 3 + 6 + 10 + 15) / 5 = 35 / 5 = 7 units.

Ans 4:

a) Let the approximation PS algorithm return a number C, and let C* be the optimal (maximum) number of programs that can be stored on the k disks. We have to show that the above approximation PS algorithm gives a performance ratio of $C^* \leq (C+k+1)$.

In the Approximation PS algorithm, we assumed the programs were ordered in accordance with the increase in size. $S1 \leq S2 \leq S3 \leq \ldots\ldots \leq Sn$

The approximation PS algorithm gives the following ratio:

$C^* \leq (C+k+1)$

Here, the program is stored with a maximum of k-1 programs, which is less than the optimal solution. The Approximation algorithm returns a number C. Here, C* is the optimal value.

$C^* \leq (C+k+1)$. It is easy to show (S1, S2, S3,.., Sn \) and L such that C* = (C(k-1) + 1) (S1, S2,…,Sn) & L. So, here, we will consider k disks with a capacity of 2kL.

$\alpha \geq C^*$

$S1+S2+S3+\ldots\ldots\ldots+S\alpha \leq 2kL$ …………………….(1)

Let n be the index value

$(S1+S2+.........+Sn) \leq L$ } ………………………..(2)

$(S1+ S2+..........+Sn+1) \leq L$

Now, it states that $n \leq \alpha$ and n programs are stored on the 1st disk, then by using approximation algorithm, From (1) and (2), we get

$(Sn+2 + Sn+3+.........+S\alpha) \leq L$

$(Sn+1 +Sn+2 +.........+S\alpha-1) \leq L$

This approximation algorithm stores at least (n+1), (n+2),... (α-1) programs in the kth disk. Therefore, the performance ratio of C* ratio of $C^* \leq (C + k -1)$ is $C^*/C \leq 1+ (k-1)$

B) Consider k = 4 and $C^* \leq (C + 3)$

Give an example (set of programs with lengths and a value L) that achieves the performance ratio of C* = (C + 3) for the 4-disk problem of PS(4), and show that it is true.

So, we can take the Example for PS(2): 4 programs with lengths of 1,1,2,2 and L=3. Example for PS(3): 9 programs with lengths of 2,2,2,3,3,3,3,3 and L=8.

Given, K=4 and $C^* \leq C + 3$).

$C^* = C + 3$

$C^*/C = 1 + 3/C$

As k=4, $C^* =C+3$

$C^* \leq C+3$

PS (4) gives 16 programs,

L = {Program count} - 1

So L = 16 - 1 = 15

Working set of programs of PS (4): 2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6

$C^* = C+3$

C*/C = 1+3/C

L = 15 which is the total amount of storage capacity of each disk and Si represents the amount of storage required to store the ith program.

Si = 2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6 = 66

Using the Greedy method,

Disk 1 | 2 | 2 | 2 | 2 | 3 | 3 | | <-----15

Disk 2 | 3 | 4 | 4 | 4 | | | | <-----15

Disk 3 | 5 | 5 | 5 | || | <-----15

Disk 4 | 6 | | | || | <-----15

Here c = 14

Therefore, the optimal solution is, C* = C + 3 = 14 + 3 = 17

Here C = 14

The optimal solution is,

C* = C + 3 = 14 + 3 = 17

This stated solution assumes that an ideal programme arrangement can truly store 17 programmes across four discs, whereas the greedy method only stores 14 programmes. If these assumptions are correct, the offered solution fits the problem statement's requirements.

C) Given, K=5 and C* ≤ (C + 4).

C*= C+4

C*/C=1+4/C

As k=5,

C* ≤ C+4 C* ≤ C+4

PS (4) gives 25 programs,

L= Programcount-1

So, L= 25-1 =24

Working set of programs of PS (5): 2,2,2,2,2, 3,3,3,3,3, 3,3,3,3,3, 4,4,4,4,4, 5,5,5,5,5

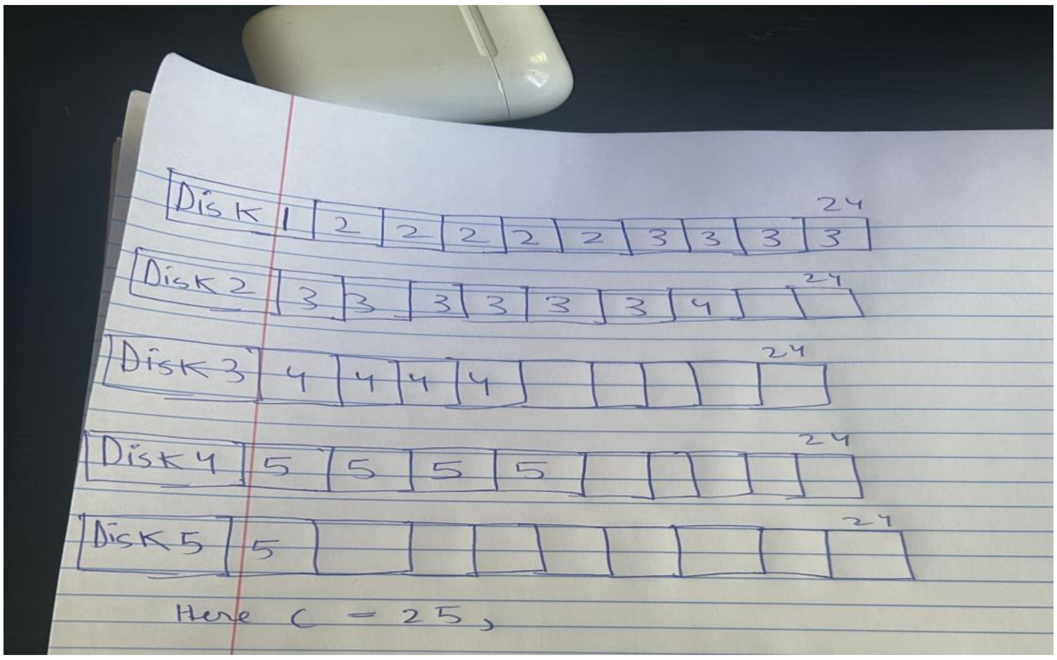
C* =C+4

C*/C= 1+4/C

L= 24 which is the total amount of storage capacity of each disk and Si represents the amount of storage required to store the ith program.

Si = 2,2,2,2,2 3,3,3,3,3 3,3,3,3,3 4,4,4,4,4 5,5,5,5,5

Using the Greedy method,



So, we have, C= 25,

Therefore, the optimal solution is, C*= C+4 = 25+4 = 29

Ans 5:

1st Problem: Pantry Organization: Suppose we are trying to organize our pantry. We have various sizes and shapes of containers and food items, and your pantry has shelves of different heights and widths. The trick is to organise the goods such that we make the most use of the available space and can readily access the things we use most regularly.

This reminds me of the NP-hard "Bin Packing Problem". There are a gazillion ways to arrange things, and the idea is to figure out the best configuration that makes the most use of available space. This is where the complexity comes in. As we increase the number of items or the constraints (such as weight balance or item fragility), finding the best solution quickly becomes computationally infeasible. The NP-hardness comes from the fact that we can't solve this problem efficiently on a large scale. There's no known algorithm that can arrange our pantry optimally in a way that scales well with the number of items.

2nd Problem: In an Amazon Hub: Finding our Locker (The Subset Sum Problem)

At an Amazon Hub, if we forget which one of the 40 lockers contains your courier and you only remember the passcode, this is similar to the Subset Sum Problem. If the passcodes are designed in such a way that they correspond to the sum of the unique identifiers of the lockers, and we are given a passcode, we need to determine which combination of locker identifiers adds up to that passcode to find our package. This problem is NP-hard because as the number of lockers increases, the number of combinations to check grows exponentially, and there's no known polynomial-time solution that works for all possible sets of identifiers and passcodes. I find it very similar to a treasure hunt game. To find our loot, we need to crack which lockers' IDs combine to match your passcode.

Ans 6:

1 - Squid Game: The Challenge (TV-Show)

https://www.netflix.com/watch/81622699?trackId=14170286

I have chosen the television programme Squid Game. It has grown a lot in popularity all around the world. Netflix has now started to broadcast it. I picked this show since it has something to do with algorithms. The players have to decide what to do. The players are divided into eight teams, each of which needs ten members, and they must organise themselves. Two teams are then chosen at random and pitted against one another in the Warship game. The players must choose a captain and lieutenant who will be in charge of launching missiles at the opposing team in an attempt to wreck their boat. This requires making decisions as well. Choosing two players is a decision problem where they have to vote YES or NO. Even the captain and his commanders have to make decisions by saying YES or NO to sink the boat of the other team. The first team to sink two ships won the round. All players in sunk ships and the Captain and Lieutenant of the losing team were eliminated, while all other players (including players on the losing team in surviving boats) moved on.

2 - **Mr Robot**

https://www.imdb.com/title/tt4652838/

"Mr. Robot" intricately portrays the world of cybersecurity, where algorithms are fundamental. The protagonist, a hacker entwined with cyber-activists, encounters challenges that mirror NP-hard problems, particularly in cryptographic security, where such complex problems ensure the robustness of encryption. The show's accurate depiction of hacking techniques implicitly acknowledges the role of algorithmic problem-solving in overcoming and exploiting system vulnerabilities. Through its narrative, "Mr. Robot" reflects the real-life significance of algorithms and the inherent difficulties presented by NP-hard problems in the domain of digital security and artificial intelligence. The protagonist, Elliot, and his group, fsociety, often face challenges that require strategic decision-making and problem-solving skills akin to those needed to tackle NP-hard problems. They must consider all possible outcomes and their associated risks, which is reminiscent of the complexity encountered in these types of problems. Cybersecurity itself is heavily reliant on algorithms. From encryption to intrusion detection systems, algorithms are used to secure data, authenticate users, and protect against cyber threats. Many of these security algorithms involve NP-hard problems, especially in cryptography, where the hardness of problems like integer factorization underpins the security of widely used encryption systems. The show also features hacking, which often involves the hacker needing to solve algorithmic puzzles to breach systems. Cracking passwords, for example, can be related to the NP-hard problem of searching through a large solution space to find the correct password or decryption key.

3 – **SiliconValley**

https://www.amazon.com/gp/video/detail/amzn1.dv.gti.045fa044-9ffb-4cda-a51c-a2b8499dbce2?autoplay=0&ref_=atv_cf_strg_wb

In "Silicon Valley," the core plot revolves around a proprietary data compression algorithm, which is a significant and realistic aspect of computer science. This algorithm, developed by the main character Richard Hendricks and his team, represents a major breakthrough in the field, offering exceptionally high compression rates without loss of data quality. The show delves into various challenges and iterations of developing and optimizing this algorithm, showcasing the importance of efficiency, speed, and scalability in software engineering.

The algorithm's development touches on real-world computational concepts such as lossless data compression, which is a process of reducing file size without losing any information. This is crucial for data storage and transmission, especially in the era of big data and cloud computing. The competitive character of the IT sector, where businesses are always racing to create better algorithms in order to obtain a competitive edge, is another topic covered in the episode. The characters struggle with problems like as managing big databases, optimising algorithms, and balancing compression efficiency and speed.

Throughout the entire series, the algorithm serves as a crucial tool that informs moral decisions and business choices alike. It is not merely a technological device. It shows how important and useful algorithms are in the computer industry, propelling company plans, inventions, and even legal disputes. "Silicon Valley" does a great job of combining comedy with these difficult ideas.

opening up the complex field of computer algorithms to a wide audience and making it enjoyable.

References used:

1. Michael R. Garey and David S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," W.H. Freeman & Co., 1979.
2. Michael Sipser, "Introduction to the Theory of Computation," PWS Publishing Company, 1997.
3. https://stackoverflow.com/questions/1857244/what-are-the-differences-between-np-np-complete-and-np-hard
4. https://softwareengineering.stackexchange.com/questions/308178/trying-to-understand-p-vs-np-vs-np-complete-vs-np-hard
5. https://www.geeksforgeeks.org/types-of-complexity-classes-p-np-conp-np-hard-and-np-complete/?ref=header_search