Homework-7

CSCE 5150 Section 003. Venkata Sai Gireesh. P (11715440).

1A) The decision problem of whether a given player can win a game of chess is decidable. This means that exists an algorithm that can correctly answer this question for any legal chess position and any player.

The algorithm works by recursively exploring the game tree, starting from the given position. At each position, the algorithm considers all possible moves that the player can make. if any of these moves lead to a winning position for the player, then the algorithm returns "yes". Otherwise, the algorithm recursively explores the game tree for each of the player's moves. if all of the player's possible moves lead to losing positions, then the algorithm returns "no".

Explanation:

This algorithm is guaranteed to terminate because the game tree of chess is finite. The number of different chess positions is finite, and the number of possible moves from each position is also finite. Therefore, the algorithm will eventually reach a position where there are no more possible moves, and it will be able to return a definitive answer.

It is important to note that this algorithm is not very practical. The game tree of chess is extremely large, and the algorithm would

require a vast amount of time and memory to explore it completely. However, the fact that the algorithm exists proves that the decision problem of whether a given player can win a game of chess is decidable.

In addition to the general algorithm described above there are also a number of more specific algorithms that can be used to determine wheather a given player can win a game of chess in a particular number of moves. For example, there is an algorithm that can determine wheather a player can checkmate their opponent in two moves. These algorithm are typically more efficient than the general algorithm, but they are only applicable to specific cases.

overall, the decision problem of whether a given player can win a game of chess is decidable. There exists an algorithm that can correctly answer this question for any legal chess position and any player. However, the algorithem is not very practical because the game tree of chess is extremely large.

The final answer is that the decision problem of wheather a given player can win a game of chess is decidable. This means that there exists an algorithm that can correctly answer this question for any legal chess position and any player.

2 A) P is the class of decision problems D such that there exists an algorithm A such that

* A with input I returns "yes" if and only if I is in D (i.e, A correctly divides D)

* The Running time of $A = O(n^k)$
Where,
n is size of instance to be decided,
k is constant.

Now, the given algorithm in the question doesn't put the problem in class P.

Here, The problem is divisors of n and the instance is successive integers from 2 to n/2 and the question asks about the divisions of n (if one of them divides n evenly, returns yes, if none of them do return no).

Considering the instance, the time complexity of the brute force algorithm is $O(n)$. Now, if we compute the proper measure, it is $b = \lceil \log_2 n+1 \rceil$, then $O(n) = O(2^b)$. So, this algorithm is not in class P as it can't be solved efficiently.

in summary, the reason the described algorithm is not in P class is because it's time complexity is not polynomial with respect to the size of the input, and it becomes increasingly inefficient as the size of the input (number) grows.

Brute-force Algorithm:

Algorithm to check if a Number is composite.

1. Start with an integer n, where n is checked.
2. Set isComposite = false
3. For each integer i from 2 to n/2: if n is divisible by i (i.e, n % i == 0). then Set isComposite to true and break from the loop.
4. If isComposite is true, return "composite" (or "yes").
5. If isComposite is still false after checking all integers, returns "prime" (or "No").

3A) First, lets look Hamilton circuit problem, if we are given a graph G(V, E). We start our search from any arbitrary vertex.

Let's say 'a' this becomes the root. The first element of our partial solution is the first intermediate vertex of Hamilton cycle. The next adjacent vertex is selected by alphabetic order. If at any stage any arbitrary vertex other than vertex 'a'. then we say that we reached a dead end. In this case, we backstrap one end and again the search begins by selecting another vertex and backtrap then the element from the parallel solutions must be removed.

a) Knapsack problem:

Decision version:

To determine wether there is a subset of a given set of 'n' items that fits into the knapsack and how has a total value not less than a given positive integer m.

polynomial - Time algorithm (To verify):

a) Sum the weights and values of the ~~proper~~ proposed set.

b) i) Weight sum should not be more than the knapsack capacity.
ii) Value sum should not be less than m.

c) Time Efficiency is O(n).

b) Bin packing problem:

Decision version:

Determine whether one can place given set of items into no more than m bins.

polynomial time algorithm (TO verify):

a) Determine the no. of bins and sum of size of items assigned to same bin.

b) i) Number of bins should not exceed m.

ii) Sum doesn't exceed the bin capacity for each of the bins

c) Time Efficiency is $O(n)$.

n → no. of items in given instance.

The decision versions of the knapsack and bin packing problems involve determining feasible solutions. polynomial-time verification algorithms check if proposed solutions meet constraints, demonstrating NP complexity class membership.

+ 20m

use of a sequence
and 26 lowercase).

4A) A problem is said to be NP complete if it is a NP-hard problem and belongs to set NP.

First look at the following properties of the problem.

1) Each Instance of the problem has Solution in YES/NO. So it is a decision problem.

2) The number of Solutions too the problem is dependent on the length of the sequence So, it is finite.

3) If we are given the sequences and a possible answer, then we can say in polynomial time whether it is correct or not look at the example given in question, we can tell that is Correct. So, problem is polynomially verifiable.

Now, lets look at whether it is NP hard or not. NP hard problems are those for which it is not possible to prove which polynomial Solution exists or not. In the given problem, we have to check all combinations to find the Solution. one method is following.

* Remove all the duplicate character from each sequence.
* Now the maximum width of a sequence has be 52 (26 uppercase and 26 lowercase).

* From sequence1 select the first letter, delete its opposite case from each sequence. This step takes $n * 52 \log 52$ = linear no. of steps.
* Now from sequence 2, select the first letter and continue as in previous step.
* Continue this process of all the sequences total complexity till now is $O(n^2)$.
* This may not give the solution. So, we have to non-deterministically select all possible combinations. We have 52 ways of selecting a letter from each sequence. We have total of n sequences. Total complexity is in order of 52^n which is exponential.

Since, there is no polynomial algorithm that we have found out yet. So it is in NP hard category.

It is both NP and NP-hard category. therefore, it is said to be NP-complete problem.

NP complete is the subset of NP and NP hard problems.

In summary, the problem of selecting a letter from each sequence of uppercase and lowercase letters without selecting both the upper and lowercase versions of any letter is NP-complete. This means that it's computationally challenging and likely does not have an efficient algorithm for solving it exactly in all cases.

5A). a) Here we can consider Hamilton undirected cycle. This is the NP Complete problem that can be used to solve this knights problem. There are 150 knights but you are not sure about the no. of conflicting knights so there is a general solution.

Consider graph G=(V,E) where V is the number of knights and E is the Edge connecting u and v where u,v is the two non-conflicting knights.

Start connecting and form edge using the non-conflicting knights continue till these forms a complete cyclic graph.

This is a valid model of king Arthur's problem because it captures the two key constraints

1. No two quarreling knights can sit next to each other. This is equivalent to saying that no two adjacent vertices can have the same color.

2. All of the knights must be seated. This is equivalent to saying that all of the vertices must be colored.

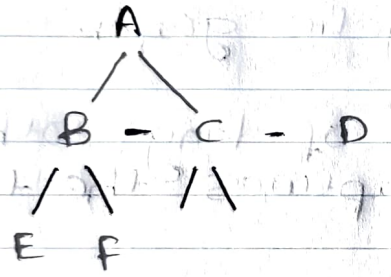
It is well known that the graph coloring problem is NP-complete.

Let me explain in more detail.

Let's say that the knights are named A, B, C, D, E, and F, and that the following pairs of knights quarrel with each other:

* A and B
  B and C
  C and D
  D and E
  E and F

We can represent this information as a graph as follows



In this graph each vertex represents a knight, and each edge represents a pair of knights who quarrel with each other. So, for example, the edge between A and B means that knights A and B quarrel with eachother.

To color the graph's vertices so that no two adjacent vertices have the same color is the aim of the graph coloring problem. We want to color the knights in King Arthur's problem so that no two fighting knights have the same color.

In this case, we can color the knights with 3 colors: red, green and blue. We could color the knights as follows:

A - red
B - green
C - blue
D - red
E - green
F - blue

This coloring is a valid solution to the graph coloring problem because no two adjacent vertices have the same color. In other words, no two knights who quarrel with each other are seated next to each other.

It is important to note that there may be other valid solutions to the graph coloring problem. For example, we could also color the knight's as follows.

A - green
B - red
C - blue
D - green
E - red
F - blue

This is also a valid solution to the graph coloring problem!

b) If only 75 of them quarrels it's easy to form a cyclic graph so the king can arrange them in a round table. If more than 75 quarrels it's is not possible to form a cyclic graph. Hence, it is impossible to arrange we can make use of Dirac's theorem.

This problem can be solved using the pigeonhole principle, which states that if n pigeons are placed in m holes, with $m<n$, then at least one hole must contain more than one pigeon. In this case, the pigeons are the knights, and the holes are the seats at the table. Since there are 150 knights and the table only has 150 seats, by the pigeonhole principle at least one seat must contain more than one knight. Now let's assume that each knight does not quarrel with atleast 75 other knights. This means that each knight can be matched with at least 75 other knights. If we were to create a graph with the knights and their matches, the result would be a graph in which every knight is connected to at least 75 other knights. Since the graph is connected. It must contain in a Hamiltonian cycle (a path that visit each node exactly once). This means that if we assign the knights to the table's seats in the order in which the Hamiltonian cycle visits them, no two quarreling knights will sit next to each other. Therefore, if each knight does not quarrel with at least 75 other knights, Arthur's problem has a solution.