Week 11: Theory of computation (Solutions)

1 Enumerability

1. Each client already present moves to the room numbered one unit higher than the one they currently occupy. This frees up room 1 for the new client.

2. Each current client moves to the room with the number that is double that of his current room. This way we can use infinity odd numbers for the bus passengers.

3. More difficult, we have to draw inspiration from enumerating pairs of integers to in order to generalize the solution to part 2. If we imagine that buses are lined up in a car park (one per line), we can imagine that the passengers of each bus are displayed in columns (one per column). Even with an infinite number of rows and columns we can construct a route of this grid by housing pairs (bus, passenger) of the same sum because there is always a finite number of pairs for a given sum. We start with a sum equal to 2 (bus 1, passenger 1), then equal to 3 (bus 1, passenger 2) and (bus 2, passenger 1), etc. (which we place in rooms with odd numbers, as in the previous exercise).
   Another possible solution is leverage the fact that the set of all prime numbers is countably infinite. Hence, we can move all the current hotel guests to room number $2^{their\ current\ room\ number}$, the passengers from the first bus to room number $3^{their\ passenger\ number}$, passengers from bus 2 to room number $5^{their\ passenger\ number}$ and so on, since there is an infinite set of prime numbers!

2 Various questions

Only statement 1 is true.

At the moment we only know that the class P is smaller than or equal to the class NP, i.e., we know that if a problem is in P, it is also in NP (statement 1 is true and statement 3 is false). At the moment, we do not know whether or not there is a problem that is in NP but not in P (statements 2 and 4 are false). There exist problems (e.g., the "Halting problem") which are not in NP (statement 5 is false).

3 From NP to P

1. If S is given, we just have to cut each pie into as many pieces as possible (of size S). The solution is valid if we can get at least F pieces in total.

Algorithm 1 Check that S is valid

```
input: N, F, list A, S
output: True or False
sum ← 0
for i from 1 to N do
    sum ← sum + ⌊A[i]/S⌋
if sum ≥ F then
    return True
else
    return False
```

   The complexity is $O(N)$.

2. The solution above is polynomial, so the problem is NP.

3. One can try all possible S, the solution is the largest valid S.

Algorithm 2 Solve the problem for each S

input: N, F, list A, M
output: L, as much as possible
$L \leftarrow 0$
for S from 1 to M **do**
    Check if S is a valid solution using Algorithm 1
    $f \leftarrow$ the response of the algorithm 1
    if $f = True$ **then**
        $L \leftarrow S$
return L

4. The previous solution is polynomial (complexity in $O(M \cdot N)$), so the problem is in P.

5. We can see that if Algorithm1(x) = False, then we have Algorithm1(y) = False for all $y > x$. We can therefore use binary search to solve the problem:

Algorithm 3 Binary search

input: N, F, list A, M
output: as much as possible
$l \leftarrow 0$
$r \leftarrow M$
while $l < r$ **do**
    $m = \lceil (l + r)/2 \rceil$
    Check if S is a valid solution using Algorithm 1
    $f \leftarrow$ the response of algorithm 1
    if $f = True$ **then**
        $l \leftarrow m$
    else
        $r \leftarrow m - 1$
return l

4 Colouring of graphs

1. For the left graph, the answer is yes; for the right graph, the answer is no.

2. The algorithm is as follows: we start with a vertex (any vertex), which we colour with one colour (for example, green). After that, we colour all its neighbours in the other colour (say red), then we try to colour all the neighbours' neighbours in green again, and so on. If, acting in this way, we find ourselves with two neighbouring vertices of the same colour, then we know that colouring the whole graph with only two colours is not possible. If, on the other hand, we reach the end without any problem, then we have found a solution to the problem.

3. The number of operations to perform to execute the above algorithm is as follows: in the worst case we must check that none of the neighbours has been coloured in the same colour as the one we are using to colour the vertex. As each vertex may have up to $n - 1$ neighbours (this will be the case for some vertices in some graphs), we risk at worst performing $O(n^2)$ operations in total (but of course, we may be able to conclude much faster than a in some cases).

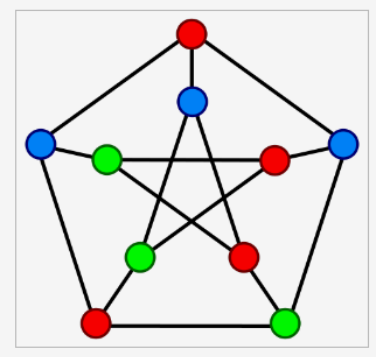
   In any case, the fact that the problem needs at worst $O(n^2)$ operations means that the problem of colouring a graph with two colours belongs to class P (this problem is solved in polynomial time in the number of variables).

4. To check if a given colouring with k colours works, we simply traverse all the vertices of the graph, and for each of the vertices, we check that none of its neighbours is coloured with the same colour as

the vertex itself. Since there are n vertices and each vertex can have at most $n-1$ neighbours, the total number of operations required for this check is again $O(n^2)$. The problem of colouring a graph with k colours is therefore in the NP class (a problem that, if we are given a solution, it is possible to check whether that solution is correct or not in time proportional to polynomials in n (where n is the number of elements we're handling)).

5. Without a starting point, finding a colouring with three colours that works for a given graph is a priori (much) more complicated. Of course, if it turns out that two-colour colouring works, as in the case of the graph on the left, then we automatically know in this case that three-colour colouring works too (just replace the colour of one of the vertices with the third colour, for example); in fact, many three-colour colourings work in this case.

   On the other hand, if no colouring of the graph with two colours works (as in the case of the graph on the right), then everything becomes more complicated for the case of three colours. It turns out that for the graph on the right, such a colouring exists:



   How can we find this solution? (How did you find it yourself?) You can, of course, try by trial and error and hope for luck... You could also try every possibility, as suggested in the statement of the exercise, but the number of operations needed would then be 3^n, so exponential in the number of elements... Can it be done better? At the moment, the answer to this question is: we don't know if there is an algorithm able to solve this problem in polynomial time in n (this would mean proving that P=NP).