2 Greedy Algorithms

2.1 Changing Money

2.1.1 Algorithm

Algorithm 1 Pseudo code for Changing Money

Input:
$money$, Money to be changed.
Output:
n, Minimum number of coins needed.

$n \leftarrow 0$
if money ≥ 10 **then**
  $n \leftarrow n + money/10$
  $money \leftarrow mod(money, 10)$
end if
if money ≥ 5 **then**
  $n \leftarrow n + money/5$
  $money \leftarrow mod(money, 5)$
end if
$n \leftarrow n + money/1$
return n

Here we simplified the subtract process into division, but the idea remains the same. Every time we subtract from $money$ the biggest coin available. This gives up the minimum number of coins to change the money.

2.1.2 Correctness

The correctness of this problem stands on the assumption that the bigger coin value has to be integer times of the smaller one. Let's assume that in some step, we subtracted the non biggest coin value v_2 instead of the biggest coin value v_1, and suppose that $v_1 = kv_2$, then we have

$$\begin{aligned} money &= remain\_money_1 + v_1 \\ &= remain\_money_1 + kv_2 \\ &= remain\_money_2 + v_2 \end{aligned}$$

As can be anticipated, $remain\_money_1 < remain\_money_2$, and thus we would need to use more coins to change the remain money if in this particular step we didn't use the biggest value coin. Thus using the biggest value would be optimal.

2.1.3 Time Complexity

Since the algorithm only contains only arithmetic operation, the time complexity would be O(1).

2.2 Organizing Party

2.2.1 Algorithm

Algorithm 2 Pseudo code for Organizing Party

Input:
$relation$, A list of pairs of people who know each other. Assume that people are numbered from 1 to n.

Output:
ans, maximum number of people that can be invited to the party

```
realtionGraph, numKnows ← buildRelationGraph(relation)
while True do
    if isValid(numKnows) then
        break
    end if
    for person in relationGraph do
        x ← numKnows(person)
        if x < 2 or x > n − 3 then
            delete(relationGraph, numKnows, person)
        end if
    end for
end while
return getAnswer(relationGraph)
```

As illustrated above in the Alg. 2, we first initialize a relationGraph recording who knows who, and an array recording the number of people each person knows.

isValid() check if there is any violations of the inviting rule in the according to numKnows. If the relation between currently invited people is valid(or no people is invited), the algorithm breaks the 'while' loop and returns the number of people that is currently invited.

If the relation is not valid, the algorithm dive in to find the first unsatisfied person that either knows too few people($x < 2$) or knows too many people($x > n - 3$), Then the algorithm delete the peoson from the relationGraph, and update the numKnows at the same time.

The algorithm will loop until it finds one satisfied solution or there is no people in the relationGraph.

2.2.2 Correctness

The correctness of the algorithm is really about proving that the *delete()* method is a save move. Formally speaking, say the origin problem is formed by a group of n invited people and the $relationship$ between them. Our goal is to prove that after removing the person who knows at most one other people(or doesn't know at most 1 people), the sub-problem still contains the optimal solution. The proving is quite straight forward, since the final optimal solution won't contain anyone who knows less than 2(greater than n-3) people. Thus ruling out the person won't affect our optimal solution, i.e. as soon as we get a valid solution, it will be the optimal solution.

2.2.3 Time Complexity

The initialization would be taking O(n) time, traversing each person in the graph.

The *while* loop would run at most n time(worst case when we have to delete every single person), and thus would take O(n) time too.

The *isValid()* and *for* loop would traverse each person to check the number of people they know, and thus taking O(n) respectively.

Finally, *delete()* would have to traverse each person and delete the target from the people they know. Taking O(n) too.

Consequently, the final time complexity of the algorithm would be O(n) + O($n \times (n + n \times n)$) = O($n^3$).