Homework2

Qinyun Song

1 coin change revisited

1. First of all, let's consider the basic condition and then extend to improve the space to $O(max_i(d_i))$. For the most simple algorithm, we define a function $f(i)$ means the minimum number of coins we need to make change for N cents. Then the final answer would be $f(N)$. The algorithm would be:

Algorithm 1 Coin Change Algorithm Version 1

```
Initialize function f for f(0) = 0.

for i = 1 to N do
    Initialize f(1) = maximum int.
    for j = 1 to k do
        if i ≥ d_j then
            f(i) = min(f(i), f(i − d_j) + 1)
        end if
    end for
end for
return f(N)
```

The simple algorithm takes $O(N)$ memory. We can see that, for each i, we only care about the values no smaller than $i - max_j(d_j)$ because we never enumerate the others. So we can only store the value no smaller than $i - max_j(d_j)$. To do this, we define a function $g(i)$, and it is different from $f(i)$. For $g(i)$, it means for a specific number who devided by $max_j(d_j)$ and the remainder is i. So the algorithm can now be changed to:

Algorithm 2 Coin Change Algorithm Version 2

Initialize function g for $g(0) = 0$.
Let max_coin equals $max_j(d_j)$.

for $i = 1$ to N **do**
    Let $pos = imodmax_coin$.
    Initialize $g(pos) =$ maximum int.
    for $j = 1$ to k **do**
        if $i \geq d_j$ **then**
            Let $pre_pos = (i - d_j)modmax_coin$.
            $g(pos) = min(g(pos), g(pre_pos) + 1)$
        end if
    end for
end for
return $g(Nmodmax_coin)$.

By using this algorithm, we only need $O(max_j(d_j))$ space.

2. First define function $f(i)$ means the number of different ways to make change for number i. The algorithm can be done as the following:

Algorithm 3 Output number of different ways

Initialize function f for $f(0) = 1$.

for $i = 1$ to N **do**
    Initialize function $f(i) = 0$.
    for $j = 1$ to k **do**
        if $i \geq d_j$ **then**
            $f(i) += f(i - d_j)$
        end if
    end for
end for
return $f(N)$.

The time complexity is $O(Nk)$. The space complexit y is $O(N)$. As shown in the previous algorithm, it can be improved so that the space complexity is $O(max_i(d_i))$.

2 Counting paths

1. *Proof.* To count the number of Hamiltonian paths, we only need to make a small change of the algorithm that finding the path. The algorithm can be done as:

Algorithm 4 Count number of Hamiltonian paths

Define a function $H(G, s, f)$ as the function to count the number of Halmiltonian paths in graph G from s to f. Define S as all the vertices in the graph.

for function $H(G, s, f)$ **do**

  for each vertex $v_i \in S$ **do**

    Call $H(G_{S/\ s}, vi, f)$

  end for

  return The sum of these functions.

end for

Initialize $answer = 0$.

for each vertex $v_i \in S$ **do**

  for each vertex $v_j \in S/\ v_i$ **do**

    $answer+ = H(G, v_i, v_j)$.

  end for

end for

return $answer$.

This algorithm is similar to the one finding the path since we only change the step after calling all possible subfunctions. We add the sum of them instead of only returning the *or* result of them. This change will not affect the complexity of time and space. Another change is that, we enumerate the start and the end point. This change will result in multipling the time complexity with $O(N^2)$ and this is also $poly(n)$. So its time complexity is also $2^n poly(n)$. For the space completixy, we won't use any more space so the space complexity is also $2^n poly(n)$. Thus we can see that this algorithm has similar time and space complexity as the algorithm finding the path. □

2. Define a function $f(i, j, l)$ means that inside the graph $G = (S, E)$, there is a path from v_i to v_j and the length of the path is exactly l. Then we only need to calculate the number of $f(u, v, k)$. The algorithm of calculating the number can be done as: The algorithm's time complexity is $O(N^3 k)$. The space complexity is $O(N^2 k)$.
   Now focus on the correctness of the algorithm. First I would like to prove that all the paths it counts are valid path. This can be proved by induction. For the path length 0, since it is just a vertex, it is definitly valid. Then suppose for path length $n-1$ all we found are valid, then we only enumerate the neighbor of the start point of the path and go from that neighbor to the start point and then follow the $n-1$-length path. If the neighbor of the start point is the same as some point in the $n-1$-length path, then there is a confliction. Because this shows that one node can arrives itself again using a path whose length is more than zero. Then this shows that the graph contains directed cycles and this is denied at the beginning. So adding a neighbor of the start point to the front of the path won't result in an invalid path. Thus for length n, the paths it counts are all valid.
   Now I need to proof that all possible paths are counted in this algorithm and they are counted once. To proof this, I also use induction. First of all, for the length of zero, the paths are definitly be counted because it is just one vertex to itself. Then suppose for all the path of $n-1$ are now be counted only once, everytime for one valid path with length $n-1$, I will pick a note that is different from the notes inside the path and add it to the beginning. So for each path with $n-1$, the resulting paths with length n are different. For different path with $n-1$, they are already different so adding a vertex in front of

Algorithm 5 Find a path with specific length in a directed acyclic graph

H

First initialize all value of $f(i, j, l)$ as -1 means that there are no ways from v_i to v_j in exactly length l.

for each $v_i \in S$ **do**
    Initialize $f(v_i, v_i, 0) = 1$.
end for

for length $l = 1$ to k **do**

    for $v_i \in S$ **do**

        for $v_j \in S$ **do**

            for All v_i's neighbor v_p **do**

                if $f(v_p, v_j, l-1)! = -1$ **then**
                    $f(i, j, l) += f(v_p, v_j, l-1)$
                end if
            end for
        end for
    end for
end for
return $f(u, v, k)$.

them cannot result in same path. Also, we have already found all possible path in length $n-1$, and add all possible vertex to the front of those path, so all the path will be counted. So all the paths will be counted and are only be counted once.

Algorithm 6 Algorithm for the coin problem

3. Define a graph G with vertices $\{v_{i,j}$ where $i \in [0, N], j \in [1, k]\}$ and no edges at the beginning.

```
for i = 0 to N do
    for j = 1 to k do
        Let l = i.

        while l ≥ 0 do
            Add edge from v_{l,j-1} to v_{i,j}.
            l− = d_j
        end while
    end for
end for
return The total number of paths of length k from v_{0,0} to v_{N,k}.
```

In the algorithm, the node $v_{i,j}$ means that to make change for money i using the first j types of coins. And it can be reached by those $v_{l,j-1}$ depending on how many type j coin we are using.

3 The Ill-prepared Burglar

1. Suppose the size of his bag is 30. There are three items with size $16, 15, 15$ and their value are all the same as one. Then the optimal way to pick up the items would be choosing the two items with size 15 so he can take total value as two. However, by using his plan, he can only take the item size size 16 and the total value would be 1.

2. Suppose the size of his bag is 3 and there are two items. The size of the first item is 2 and its value is 10. So the ratio is 5. The size of the second item is 3 and its value is 12. Its ratio is 4. Then using his plan, he would choose the first item and get total value as 10. However, if he chooses the second item, he can get total value as 12.

4 Road tripping

1. Suppose there are four songs. Their length are $10, 30, 50, 30$. So the optinal choice would be pack the first and the third songs to one CD and the remaining two into another CD. So we only need two CDs. However, if we use the greedy algorithm, we will first pack the first two songs into one CD. And then the third song to the second CD and the fourth song to the third CD. This would result in using three CDs to pack all the songs.

2. *Proof.* Claim: For each CD in the optimal algorithm, it will be separated into no more than two CDs in the greedy algorithm.

Proof. Suppose the total duration of the songs in one CD is d. Then we have $d \leq 60$. Then if in the greedy algorithm, it is separated into three CDs, then there must be two CDs can be merged into one. Otherwise, the length in each CD should be greater than 30 and will result into total 90 duration. This is contradicted with the problem. □

Thus, for each CD in the optional algorithm, the greedy algorithm will place the songs into no more than two CDs. This shows that the greedy algorithm is always within a factor 2 of the "best posssible" number of CDs. □

5 Maximizing Happiness

1. For a case there are three children and three gifts. Define the matrix A as

$$\begin{bmatrix} 99 & 1 & 0 \\ 0 & 99 & 1 \\ 100 & 0 & 99 \end{bmatrix}$$

Then the optinal way to assign the gifts would be assign the i^{th} gift to i^{th} child with total happiness as $3 \times 99 = 297$. But for the greedy algorithm, it will first assign the third gift to the first child and then assign the first to the second one and assign the second gift to the third one. This will result in total happiness equals $100 + 1 + 1 = 102$. We can see that, $102 \leq \frac{1}{2} \times 297$. Thus this is an example showing that the locally optional solution is at least a factor 2 smaller than the globally optional solution.

2. *Proof.* Define that the global optional function is $f_{i,j}$ and the local optional function is $g_{i,j}$. The total happiness of local optional algorithm is G and for global is F.

Claim: For two index $i, j \in N$, we have

$$A_{i,f(i)} + A_{j,f(j)} \leq A_{i,g(i)} + A_{j,f(i)} + A_{g^{-1}(f(j)),f(j)} \tag{1}$$

Proof. Suppose there exists a pair i, j such that the above claim does not hold. Then the local optional algorithm will not choose this g function because it is not locally optionaltoo. This is a contradiction with the definition where g is the function of optional algorithm. So the equation must hold. □

With the claim above, for all pair $(i, f(i))$ and $(j, f(j))$, if we sum them up, we have:

$$\begin{aligned} \sum_{i=1}^{N} A_{i,f(i)} + \sum_{j=1}^{N} A_{j,f(j)} &\leq \sum_{i=1}^{N} A_{i,g(i)} + \sum_{j=1}^{N} A_{j,f(i)} + \sum_{j=1}^{N} A_{g^{-1}(f(j)),f(j)} \\ F + F &\leq G + G + G \\ 2F &\leq 3G \end{aligned} \tag{2}$$

This shows that the locally optional solution produced this way has a value that is at least (2/3) the optimum value. □