Discrete Mathematics Problem Set 4

Department of Computer Science and Engineering
2021-16988 Jaewan Park

Problem 1

1. When $a_1, a_2, \cdots, a_k$ are given as intial states, $p_0, p_1, \cdots, p_t$ can be determined to satisfy the following, since it is a linear system of independent states.

$$\begin{aligned}
a_1 &= (p_t + p_{t-1} + \cdots + p_0) \cdot s \\
a_2 &= \left(p_t \cdot 2^t + p_{t-1} \cdot 2^{t-1} + \cdots + p_0\right) \cdot s^2 \\
&\cdots \\
a_k &= \left(p_t \cdot k^t + p_{t-1} \cdot k^{t-1} + \cdots + p_0\right) \cdot s^k
\end{aligned}$$

Suppose the given equation is a particular solution of the relation at $n = i-1, i-2, \cdots, i-k$. Now we should prove that it is also a particular solution at $n = i$.

We know that

$$\begin{aligned}
a_{i-1} &= \Big(p_t \cdot (i-1)^t + p_{t-1} \cdot (i-1)^{t-1} + \cdots + p_0\Big) \cdot s^{(i-1)} \\
a_{i-2} &= \Big(p_t \cdot (i-2)^t + p_{t-1} \cdot (i-2)^{t-1} + \cdots + p_0\Big) \cdot s^{(i-2)} \\
&\cdots \\
a_{i-k} &= \Big(p_t \cdot (i-k)^t + p_{t-1} \cdot (i-k)^{t-1} + \cdots + p_0\Big) \cdot s^{(i-k)}
\end{aligned}$$

Therefore the recurrence relation gives :

$$\begin{aligned}
a_i &= c_1 \cdot a_{(i-1)} + c_2 \cdot a_{(i-2)} + \cdots + c_k \cdot a_{n-k} + F(n) \\
&= c_1\Big(p_t \cdot (i-1)^t + p_{t-1} \cdot (i-1)^{t-1} + \cdots + p_0\Big) \cdot s^{(i-1)} \\
&\quad + c_2\Big(p_t \cdot (i-2)^t + p_{t-1} \cdot (i-2)^{t-1} + \cdots + p_0\Big) \cdot s^{(i-2)} \\
&\quad + \cdots + c_k\Big(p_t \cdot (i-k)^t + p_{t-1} \cdot (i-k)^{t-1} + \cdots + p_0\Big) \cdot s^{(i-k)} \\
&\quad + \left(b_t \cdot i^t + b_{t-1} \cdot i^{t-1} + \cdots + b_0\right) \cdot s^i \\
&= \left(p_t \cdot i^t + p_{t-1} \cdot i^{t-1} + \cdots + p_0\right) \cdot s^i
\end{aligned}$$

2.

Problem 2

1. When n people vote twice each there are total $2n$ votes, so the maximum number of candidates with more than $n/2$ votes is three.

Also, if we devide the people into two groups, for a candidate to get more than votes from half of the people, that candidate should get more than votes from half of the people in at least one group.

Therefore dividing groups will give a maximum of three candidates with more than votes from half of the people each, and the positions will go to two of these candidates. (There can be repeated candidates in both groups.) So the following algorithm will give all candidates with more than $n/2$ votes and their count of votes, and then we can determine whom the winners are.

Algorithm 1 Candidate Qualify Algorithm

```
function ELECTION($a_{11}, a_{12}, a_{21}, a_{22}, \cdots, a_{n1}, a_{n2}$ : list of integers, all votes of $n$ people)
    $(r_{11}, c_{11}), (r_{12}, c_{12}), (r_{13}, c_{13}) = $ ELECTION$(a_{11}, a_{12}, \cdots, a_{\lfloor \frac{n}{2} \rfloor 1}, a_{\lfloor \frac{n}{2} \rfloor 2})$
    $(r_{21}, c_{21}), (r_{22}, c_{22}), (r_{23}, c_{23}) = $ ELECTION$(a_{\lfloor \frac{n}{2}+1 \rfloor 1}, a_{\lfloor \frac{n}{2}+1 \rfloor 2}, \cdots, a_{n1}, a_{n2})$
    for $p \in \{11, 12, 13\},\ q \in \{21, 22, 23\}$ do
        if $r_p = r_q$ then                                   ▷ same candidate in both groups
            $c_p, c_q = c_p + c_q$                            ▷ add count of votes
        end if
    end for
    return $\{(r_i, c_i)\}$ : list of (candidate, count) with more than $n/2$ votes     ▷ maximum three returns
end function
```

2. We devide the problem into two problems of size $n/2$. After that, $3 \times 3 = 9$ comparisons are made to find if there are any repeated candidates from both groups. Finally, we compare the (maximum) six candidates' count of votes with $n/2$ to find who to return. (An additional step to determine two winners out of three possible candidates is also needed too, but it isn't part of the recursive algorithm.)

$$T(n) = 2T\left(\frac{n}{2}\right) + 15$$

Therefore the recurrence relation is as the above, and the algorithm's complexity is :

$$T(n) = O\left(n^{\log_2 2}\right) = O(n)$$

Problem 3

1. Let's call the divided parts A, B, C, and D. Now two queries can be made, selecting $A \cup B$ and $A \cup C$.

$A \cup B$	$A \cup C$	True True	True False	False True	In Common
Yes	Yes	Always in A	Always in B	Always in C	Eliminate D
Yes	No	Always in B	Always in A	Always in D	Eliminate C
No	Yes	Always in C	Always in D	Always in A	Eliminate B
No	No	Always in D	Always in C	Always in B	Eliminate A

As shown in the table above, for all cases where the first person lies or not, we can eliminate one part, which is 1/4 of the entire set.

2. Since we showed that we can eliminate 1/4 of the elements by two queries, after that we should solve a problem with $3n/4$ numbers. Therefore the recurrence relation is:

$$f(n) = f\left(\frac{3}{4}n\right) + 2$$

Now since n is a multiple of 4, set $n = 4k$. Therefore

$$\begin{aligned}
f(n) &= f\left(\left\lceil \frac{3}{4}n \right\rceil\right) + 2 \\
&= f\left(\left\lceil \frac{3}{4}\left\lceil \frac{3}{4}n \right\rceil\right\rceil\right) + 2 \cdot 2 \\
&\le f\left(\left\lceil \frac{3}{4}\left(\frac{3}{4}n + 1\right)\right\rceil\right) + 2 \cdot 2 = f\left(\left\lceil \left(\frac{3}{4}\right)^2 n + \frac{3}{4} \right\rceil\right) + 2 \cdot 2 \\
&\le f\left(\left\lceil \left(\frac{3}{4}\right)^3 n + \left(\frac{3}{4}\right)^2 + \frac{3}{4} \right\rceil\right) + 2 \cdot 3 \\
&\le \cdots \le f\left(\left\lceil \left(\frac{3}{4}\right)^k n + \sum_{i=1}^{k-1} \left(\frac{3}{4}\right)^i \right\rceil\right) + 2k
\end{aligned}$$

When $k \approx \log_{4/3} n$,

$$\left\lceil \left(\frac{3}{4}\right)^k n + \sum_{i=1}^{k-1} \left(\frac{3}{4}\right)^i \right\rceil \approx \left\lceil 1 + 3 - 3\left(\frac{3}{4}\right)^{k-1} \right\rceil = 4$$

Therefore

$$f(n) \le \cdots \le f\left(\left\lceil \left(\frac{3}{4}\right)^k n + \sum_{i=1}^{k-1} \left(\frac{3}{4}\right)^i \right\rceil\right) + 2k = f(4) + 2k$$

when $k \approx \log_{4/3} n$. Therefore $f(n) = O(2k) = O(2\log_{4/3} n) = O(\log n)$.

Problem 4

Let U the set of all permutations, F, R, B the set of permutations that include *fish*, *rat* and *bird*. The number of permutations containing *fish* can be considered as ordering 23 objects: *fish*, *a*, *b*, $\cdots$, *z*. Other cases can be thought of as the same way.

Therefore, using the principle of inclusion-exclusion gives :

$$\begin{aligned}
\left|F^C \cap R^C \cap B^C\right| &= 1 - |F \cup R \cup B| \\
&= |U| - (|F| + |R| + |B|) + (|F \cap R| + |R \cap B| + |B \cap F|) - |F \cap R \cap B| \\
&= 26! - (23! + 24! + 23!) + (20! + 0 + 0) + 0 \\
&= 16419539410212005217738692971219157417
\end{aligned}$$

Problem 5

1. If $w_j > w$, item j cannot be chosen since that one element will exceed the maximum weight.

   If $w_j \le w$, we can think of two cases, whether j is included in the optimal solution or not. If j isn't included, the maximum weight will be same as $M(j-1, w)$. If it is included, we should choose the best set of items from $1, 2, \cdots, j-1$ not to exceed a total weight of $w - w_j$ The total weight then becomes $w_j + M(j-1, w-w_j)$. Therefore the maximum value between these two cases will be the answer.

2. Using the recurrence relation shown in 5-1, we can write the following pseudocode.

   Calculating MaxWeight(i, W) will give the maximum total weight.

Algorithm 2 Knapsack Problem

```
function MaxWeight($j$, $w$ : positive integers)
    if $w_j > w$ then
        return MaxWeight($j-1$, $w$)
    else
        $ex$ = MaxWeight($j-1$, $w$)
        $in = w_j$ + MaxWeight($j-1$, $w - w_j$)
        return $\max\{ex, in\}$
    end if
end function
```

3. From the same algorithm from 5-1 and 5-2, returning both the maximum weight and the chosen set of items will help find the subset of items we want to find.

   Calculating MaxWeight(i, W) will give the maximum total weight and the corresponding subset.

Algorithm 3 Knapsack Problem

```
function MaxWeightSet($j$, $w$ : positive integers)
    if $w_j > w$ then
        return MaxWeightSet($j-1$, $w$)
    else
        $ex$, $set_{ex}$ = MaxWeightSet($j-1$, $w$)
        $in = w_j$ + MaxWeightSet($j-1$, $w - w_j$).$weight$
        $set_{in} = \{j\}$ + MaxWeightSet($j-1$, $w - w_j$).$set$
        if $ex > in$ then
            return ($ex$, $set_{ex}$)          ▷ return both maximum weight and set of items
        else
            return ($in$, $set_{in}$)          ▷ return both maximum weight and set of items
        end if
    end if
end function
```

Problem 6

R is antisymmetric if and only if the following is true.

$$\text{If } (a,b) \in R \text{ and } (b,a) \in R,\ a = b.$$

Since we defined $R^{-1} = \{(b,a) \mid (a,b) \in R\}$, $(b,a) \in R$ is equivlent to $(a,b) \in R^{-1}$. Therefore the above statement is equivalent to

$$\text{If } (a,b) \in R \text{ and } (a,b) \in R^{-1},\ a = b. \Leftrightarrow \text{If } (a,b) \in R \cap R^{-1},\ a = b.$$

For $\forall (a,b) \in R \cap R^{-1}$, $a \in A$ and $b \in A$ since R is a relation on A. Also $a = b$ according to the above statement. Therefore $(a,b) = (a,a) \in \Delta = \{(a,a) \mid a \in A\}$, and $R \cap R^{-1} \in \Delta$.

Problem 7

Let's consider an initial relation R on a set A. Then its transitive closure is equal to the connectivity relation R^*. If we define $\Delta = \{(a,a) \mid a \in A\}$, the reflexive closure should contain Δ, so the reflexive closure of R^*

is $R^* \cup \Delta$. Now let $S = R^* \cup \Delta$. If we define $S^{-1} = \{(b,a) \mid (a,b) \in S\}$, the symmetric closure of S should contain S^{-1}, so the symmetric closure of S is $S \cup S^{-1}$. Now let $T = S \cup S^{-1}$.

Since T is a symmetric closure of a relation(S), it is symmetric itself. Also, since $T = S \cup S^{-1} = R^* \cup \Delta \cup S^{-1}$, it contains Δ and is consequently reflexive. Now for T to be an equivalence relation, it should be transitive.

However there is a counterexample, where $A = \{a,b,c\}$ and $R = \{(a,b),(c,b)\}$. Then $R^* = \{(a,b),(c,b)\}$ and $\Delta = \{(a,a),(b,b),(c,c)\}$, so $S = \{(a,a),(a,b),(b,b),(c,b),(c,c)\}$. Then $S^{-1} = \{(a,a),(b,a),(b,b),(b,c),(c,c)\}$. Therefore $T = \{(a,a),(a,b),(b,a),(b,b),(b,c),(c,b),(c,c)\}$. Now $(a,b),(b,c) \in T$ but $(a,c) \notin T$. Therefore T is not transitive, and is not an equivalence relation.

Problem 8

Let's say two graphs G, G' are isomorphic and a bijection $f \colon V \to V'$ satisfies the following.

$$\forall a,b \in V,\ a,b \text{ are adjacent in } G \text{ iff } f(a), f(b) \text{ are adjacent in } G'.$$

If G is bipartite, we can divide the vertices of G into two subsets V_1 and V_2, where there are no edges connecting vertices between each set. Then we can define :

$$V_1' := \{f(v) \mid v \in V_1\},\ V_2' := \{f(v) \mid v \in V_2\}$$

If G' is not bipartite, V_1' and V_2' cannot be a partition that don't have any edges in between. Therefore suppose an edge $e' \in G'$' exists which connects $v_i' \in V_1'$ and $v_j' \in V_2'$. Since G and G' are isomorphic, an edge e exists in G which connects $f^{-1}(v_i') := v_i$ and $f^{-1}(v_j') := v_j$. Since $v_i' \in V_1'$ and $v_j' \in V_2'$, $v_i \in V_1$ and $v_j \in V_2$ according to the definition of V_1' and V_2'.

However G is bipartite, and there shouldn't be an edge connecting vertices between V_1 and V_2. This is a contradiction, and G' should be bipartite. Therefore a graph being bipartite can be an isomorphic invariant.

References

[1] 'Divide-and-conquer: Determining the top two candidates and whether these two candidates received', *Computer Science Stack Exchange*, 2018. Available: `https://cs.stackexchange.com/questions/96960/divide-and-conquer-determining-the-top-two-candidates-and-whether-these-two-can`.

Usage of References

Problem 2 : I referred to the idea of the same problem's answer in [1], gave a new written solution.