CMPT 441-711: Homework 1

Instructor: Prof. Cenk Sahinalp

Fall 2014

Question 2.17

We can model this problem as a set of two recurrence relations. Let $V(t)$ be the number of viruses after t minutes. From the problem description we can obtain the following recurrence:

$$\begin{aligned} V(0) &= 1 \\ V(t) &= 2V(t-1) \end{aligned}$$

After examining a few values of $V(t)$, we can assume that $V(t) = 2^t$. To prove this assumption, we use induction:

1. For $t = 0$, we obviously have $V(0) = 1$. For $t = 1$ we have $V(1) = 2V(0) = 2 \times 1 = 2$.
2. Suppose that statement holds for $t = k$.
3. We need to prove that it holds for $t = k + 1$ as well. In this case, we have $V(k + 1) = 2V(k)$ and because of step (2) we can expand $V(k)$, so we have $V(k + 1) = 2V(k) = 2 \times 2^k = 2^{k+1}$.

Similarly, we can set up the recurrence for the number of bacteria after t minutes. Let it be $B(t)$, and we have:

$$\begin{aligned} B(0) &= n \\ B(t) &= 2(B(t-1) - V(t-1)) \end{aligned}$$

Using the closed form of the $V(t)$ we can simplify the previous statement to:

$$B(t) = 2B(t-1) - 2^t.$$

Again, after few tryouts, we can assume that $B(t) = 2^t(n - t)$. Now, we need to prove it by induction:

1. For $t = 1$, we have $B(1) = 2B(0) - 2^1 = 2n - 2$.
2. Suppose that statement holds for $t = k$.
3. To prove it for $t = k + 1$, we use second step to obtain $B(k + 1)$.
   $B(k + 1) = 2B(k) - 2^{k+1} = 2(2^k n - 2^k k) - 2^{k+1} = 2^{k+1}n - 2^{k+1}(k + 1) = 2^{k+1}(n - (k + 1))$.

Viruses will kill all the bacteria if there is a number $t \in \mathbb{N}$ such that $B(t) = 0$. Solving the last equation we obtain:

$$B(t) = 0 \Leftrightarrow 2^t(n - t) = 0 \Leftrightarrow t = n.$$

After N minutes there will be no bacteria inside Petri dish. Our algorithm for computing number of steps is pretty trivial: just output n. Complexity is obvious $O(1)$.

Question 2.19

If we have any column which contains a zero and is not a zero-column, then it is impossible to obtain zero matrix. Because every subtraction with one will lead to negative number, which cannot be increased by the given operations (subtraction/multiplication).

This observation gives us a general idea: try to subtract 1 from some column until some element of that column becomes 1. If all elements are 1, one subtraction will eliminate that column. If not, we will multiply any row which contains 1 in our column by two, and perform the subtraction again. As each multiplication will be reverted after a number of subtractions (remember, we just double rows with 1), in the next step we will have all other non-1 elements lowered by 1, eventually reaching the state of one-column (i.e. column which contains only ones). For example:

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 1 & 1 & 2 \end{bmatrix} \to \begin{bmatrix} 2 & 4 & 6 \\ 2 & 3 & 1 \\ 1 & 1 & 2 \end{bmatrix} \to \begin{bmatrix} 2 & 4 & 6 \\ 2 & 3 & 1 \\ 2 & 2 & 4 \end{bmatrix} \to \begin{bmatrix} 1 & 4 & 6 \\ 1 & 3 & 1 \\ 1 & 2 & 4 \end{bmatrix} \to \begin{bmatrix} 0 & 3 & 6 \\ 0 & 2 & 1 \\ 0 & 1 & 4 \end{bmatrix} \to \dots$$

Previous algorithm can be further refined if we notice that we spend too much time subtracting and doubling. In fact, it is better to make each column number as close as possible to the largest number m in the column. So we need to double each row as much as possible without exceeding largest number m. In fact, as we are multiplying by 2 it is sufficient to double each row $\left\lfloor \log_2 \frac{m}{a} \right\rfloor$ times (where a is the element in our column and in desired row).

We can state algorithm as following:

Input: Matrix $M[1..8, 1..8]$
if $\exists (i,j) : M[i,j] = 0$ **and** $\exists j' \neq j : M[i,j'] \neq 0$ **then**
  return;
end
for $c \leftarrow 1$ **to** 8 **do**
  while $\exists r : M[r,c] \neq 0$ **do**
    $m \leftarrow$ `MaximumElementInColumn` (c);
    for $r \leftarrow 1$ **to** 8 **do**
      `DoubleRows` (r) $\left\lfloor \log_2 \frac{m}{M[r,c]} \right\rfloor$ times;
      $n \leftarrow$ `MinimumElementInColumn` (c);
      `SubstractColumn` (i) $n - 1$ times;
    end
  end
end

If m is the maximum element in M, then in the worst case we need $O(m)$ operations to eliminate first column. Now, as we were multiplying some rows by 2 no more than $\lfloor \log_2 m \rfloor$ times, the largest possible number in the second column could not exceed $2^{\lfloor \log_2 m \rfloor} m = m^2$. Analogously, we obtain that for 8th row we need no more than $O(m^8)$ operations, and as the number of rows/columns is constant we conclude that complexity of the algorithm is $O(m^8)$.

Question 2.20

It is sufficient to analyze just the case in which all chameleons obtain the color k. Obviously, if $m = n$ then solution is simply interaction between remaining green and black chameleons. Also, we can represent current state as row vector $\begin{pmatrix} m & n & k \end{pmatrix}$. Using such notation we can represent our desired state as $\begin{pmatrix} 0 & 0 & m+n+k \end{pmatrix}$. Notice that each possible interaction could be represented as a $\begin{pmatrix} 2 & 2 & 2 \end{pmatrix}$ transition vector modulo 3 (e.g. if a black chameleon meets a green one, they will both become brown chameleons, so the transition vector is $\begin{pmatrix} -1 & -1 & 2 \end{pmatrix}$ which is the same as $\begin{pmatrix} 2 & 2 & 2 \end{pmatrix}$ modulo 3).

Using such a notation we can prove the following things:

- *If each element of the state vector yields a different remainder modulo 3, then there is no solution*
  Indeed, if the $\begin{pmatrix} 0 & 1 & 2 \end{pmatrix}$ modulo 3 is the state vector, then adding transition vector will result in

$\begin{pmatrix} 2 & 0 & 1 \end{pmatrix}$ modulo 3. Adding again will now result in $\begin{pmatrix} 1 & 2 & 0 \end{pmatrix}$, and next addition will return us to the initial state $\begin{pmatrix} 0 & 1 & 2 \end{pmatrix}$. As we cannot obtain the solution vector $\begin{pmatrix} 0 & 0 & q \end{pmatrix}$ modulo 3 (where q could be any possible remainder), we conclude that there is no solution.

- *If two elements of the state vector have a same remainder modulo 3, then we can obtain a solution*
  Suppose that $m \equiv n \pmod 3$. Then our vector is of form $\begin{pmatrix} a & a & q \end{pmatrix}$ modulo 3, where a and q are some of the possible remainders modulo 3. After a interactions between black and green chameleons we will obtain some vector $\begin{pmatrix} 0 & 0 & q+2a \end{pmatrix}$ modulo 3, so we have already obtained our solution vector **modulo 3**.
  Now, we just need to prove that any state vector $\begin{pmatrix} m & n & k \end{pmatrix}$, where $m \equiv n \equiv 0 \pmod 3$ can lead to the solution vector $\begin{pmatrix} 0 & 0 & m+n+k \end{pmatrix}$. If we perform $\min\{m,n\}$ interactions between black and green chameleons, we will obtain the vector $\begin{pmatrix} 0 & n-m & k+2m \end{pmatrix}$ (for simplicity we will suppose that $m \leq n$). We now need to somehow get rid of those $n-m$ black chameleons. Because $n-m$ is divisible by 3 and greater than zero (otherwise we would already obtained our solution), following two steps will obviously decrease the number of black chameleons without increasing the number of green chameleons:

  1. Arrange the meeting of brown and black chameleons, leading to the $\begin{pmatrix} 2 & n-m-1 & k+2m-1 \end{pmatrix}$ vector
  2. Arrange the meeting of two green and black chameleons, now leading to the $\begin{pmatrix} 0 & n-m-3 & k+2m+3 \end{pmatrix}$ vector (we are able to do this because $n-m \geq 3$)

Because $n-m$ is divisible by 3 and lower-bounded by 0, we can obtain the solution vector by merely repeating previous steps.

To summarize, it is possible to achieve a solution if and only if two of three numbers m, n and k have the same remainder modulo 3.

Question 4.11

Suppose that root is at level 0, and that every leaf is at level L (because each path between root and leaf is of length L). If we let $T(l)$ be the number of the children of some node v belonging to the level l (including that node), we can set up the following recurrence:

$$\begin{aligned} T(L) &= 1 \\ T(l) &= 1 + kT(l+1) \end{aligned}$$

Our solution is obviously $T(0)$. After evaluating the recurrence relation we obtain

$$T(0) = 1 + kT(1) = 1 + k(1 + kT(2)) = 1 + k + k^2T(3) = \ldots = \sum_{i=0}^{L} k^i = \frac{k^{L+1} - 1}{k - 1}$$

As we notice that last statement is in fact geometric progression, we obtain the closed-form expression using a formula for general geometric progression.

Question 4.12

We will use the following algorithm:

```
Input: Text string T[1..n], pattern P[1..m]
Output: Index of the first occurrence of the P in the T
for i ← 1 to n − m + 1 do
    found ← true;
    for j ← i to i + m − 1 do
        if T[i] ≠ P[j] then
            found ← false;
            exit for;
        end
    end
    if found then
        print i;
        exit for;
    end
end
```

Basically, we will check each position within T for the occurrence of the P. We check for the occurrence by simply comparing P with the each substring of length m within T. This algorithm will output nothing if there is no occurrence of P inside T. It will find first occurrence because it iterates through the set of m-substrings $\{T_i \mid 1 \leq i \leq n - m + 1\}$ incrementally (if we set T_i as the m-substring of T starting at the position i). As it needs to scan entire string T (in fact first $n - m + 1$ characters) in the worst case, and because for each character it needs to analyze next m characters, worst-case complexity of this algorithm is $(n - m + 1)(c + dm) = dmn - dm^2 + (d - c)m + cn + c \in O(mn)$ (for c, d constants and $m \ll n$).

You may also use KMP.

Question 4.13

We will use the following algorithm (similar to the previous one):

```
Input: Text string T[1..n], pattern P[1..m], k
Output: Index of the first substring of T for which its Hamming distance with P is less or equal than
        k
for i ← 1 to n − m + 1 do
    dist ← 0;
    for j ← i to i + m − 1 do
        if T[i] ≠ P[j] then
            dist ← dist +1;
        end
    end
    if dist ≤ k then
        print i;
        exit for;
    end
end
```

Analysis is similar to the previous algorithm. Complexity is again $O(mn)$.