12/17/2017

CS 5200 - Take Home Final

Name: Venkata Praveen Kumar. M

1) a) Definition of O:-

$f(n) \in O(g(n))$ when

$\exists\, c > 0$ & $\exists\, n_0 \in N$, $n_0 \geq 1$.

Such that, $|f(n)| \leq c|g(n)| \quad \forall\, n > n_0$.

Let's Call $f(n) = 81n^3 + 1300n^2 + 300n$.

& $g(n) = n^5 - 15000n^4 - 10n^3$

Let $c = 1$ & let $n_0 = 20,000$.

If $n > n_0 = 20000$,

$$f(n) \leq 81n^3 + \frac{1300n^2 \times n}{20000} + \frac{300n \times n^2}{(20000)^2} \quad \left(\text{As } n > 20,000, \frac{n}{20000} > 1\right).$$

$$\Rightarrow f(n) \leq 81n^3 + \frac{13}{200}n^3 + \frac{3n^3}{4 \times 10^6}$$

$$\Rightarrow f(n) \leq 82n^3$$ (As 2^{nd} and 3^{rd} term are less than n^3)

$$g(n) = n^5 - 15000n^4 - 10n^3$$

$$\Rightarrow g(n) \geq (20000)^2 \times n^3 - 15000 \times 20000 n^3 - 10n^3$$ (Take least n, change to '20')

$$\Rightarrow g(n) \geq (20000 \times 20000 - 15000 \times 20000 - 10) \times n^3$$

$$\Rightarrow g(n) \geq (5000 \times 20000 - 10) \times n^3$$

$$\Rightarrow g(n) \geq 99999990n^3 \geq 82n^3 \geq f(n)$$

As n is positive, for any n value, $g(n) \geq f(n)$

$$\Rightarrow f(n) \leq c|g(n)| \quad \forall\, n > n_0.$$

$\therefore$ we can say that $f(n) \in O(g(n))$.

Now, we need to Prove $g(n) \notin O(f(n))$

Proof by Contradiction:-

Assume $\exists C$ & $\exists n_0$ such that

$\forall n > n_0$, $|g(n)| \leq c|f(n)|$.

$\forall n > 20,000$ $g(n) \geq 0$, $f(n) \geq 0$.

$|g(n)| = g(n)$ & $|f(n)| = f(n)$.

$$n^5 - 15000n^4 - 10n^3 \leq c(81n^3 + 1300n^2 + 300n)$$

$$\Rightarrow n^5 \leq 15000n^4 + 10n^3 + 81 \cdot c \cdot n^3 + 1300 \cdot c \cdot n^2 + 300 \cdot c \cdot n$$

$$\Rightarrow \frac{n^5}{n^4} \leq \frac{15000n^4 + 10n^3 + 81 \cdot c \cdot n^3 + 1300 \cdot c \cdot n^2 + 300 \cdot c \cdot n}{n^4}$$

$$\Rightarrow \frac{n^5}{n^4} \leq 15000 + \frac{10}{n} + \frac{81c}{n} + \frac{1300 \cdot c}{n^2} + \frac{300 \cdot c}{n^3}$$

$$\Rightarrow n \leq 15000 + \frac{10}{n} + \frac{81 \cdot c}{n} + \frac{1300 \cdot c}{n^2} + \frac{300 \cdot c}{n^3} \quad \text{— } (1)$$

As c is constant, For large n ($n \geq 20000$) we can get the L.H.S value a small fraction more than 15000. (like 15000.53)

This contradicts the equation (1) as $n \geq 20000$. So the ~~statement in~~ equation (1) does not hold True. Thus, our Assumption is not true.

$\therefore g(n) \notin O(f(n))$.

3) a) The linear relationship between q and sq is:

$$q(n) = sq(n-2) + 7 \quad \forall\, n > 0.$$

b) Proof by Induction:-

1) step-1, Define the Problem:-

Given definitions of q and sq are:

```
def q(n):
    if n <= 0:
        return 1
    elif n < 2:
        return 7
    else:
        return q(n-1) + q(n-2)

def sq(n)
    if n < 0:
        return 0
    else:
        return sq(n-1) + q(n)
```

We need to Prove that $q(n) = sq(n-2) + 7 \;\forall\, n > 0$.

That means the Domain is all +ve integers. So, the base Case will Start with 1.

```
def P(n):
    return q(n) == sq(n-2) + 7.
```

2) Step-2 - Check base case and two other values:

$n=1$: $q(1) = 7$

$sq(-1)+7 = 0+7 = 7$

$\therefore q(1) = sq(-1)+7$ ($n=1$ is satisfied)

$n=2$: $q(2) = 8$

$sq(0)+7 = 1+7 = 8$

$\therefore q(2) = sq(0)+7$ ($n=2$ is satisfied)

$n=3$: $q(3) = 15$

$sq(1)+7 = 8+7 = 15$

$\therefore q(3) = sq(1)+7$ ($n=3$ is satisfied.

We got $P(n)$ as true for base case and next two values.

3) Step-3 - for all $n>0$, Prove that $P(n)$ is true if $P(n-1)$ is true.

If $P(n-1)$ is true, $q(n-1) = sq(n-3)+7$. — ①

From the definition of $q(n)$, $^{(n>2)}$ $q(n) = q(n-1) + q(n-2)$ — ②

From the definition of $sq(n)$, $sq(n-2) = sq(n-3) + q(n-2)$

$\Rightarrow q(n-2) = sq(n-2) - sq(n-3)$ — ③

Substituting ③ in ② $\Rightarrow q(n) = q(n-1) + sq(n-2) - sq(n-3)$

from eqn ① $\Rightarrow q(n) = \cancel{sq(n-3)} + 7 + sq(n-2) - \cancel{sq(n-3)}$

$\Rightarrow q(n) = sq(n-2) + 7$.

Thus, if $P(n-1)$ is True, we got $P(n)$ as True. Since n was arbitrary, it follows that for all $n>0$, n inherits P from $n-1$.

4) Step-4 - Invoke Induction: From step 2 it follows that $P(1)$ is true, while from step 3 it follows for all $n>0$. Since, the hypothesis of Simple Induction are true, the conclusion.

4) a) vec_3 has each vector of length 3. Each vector can have one of $\{0,1,2\}$ at one of the three spaces. As each vector length is 3, we will have $3^3 = 27$ vectors.

So, Sample Space $S = \{(0,0,0), (0,0,1), (0,0,2), (0,1,0), (0,2,0), (0,1,1), (0,1,2), (0,2,2), (0,2,1), (1,0,0), (1,0,1), (1,0,2), (1,1,0), (1,2,0), (1,1,1), (1,1,2), (1,2,2), (1,2,1), (2,0,0), (2,0,1), (2,0,2), (2,1,0), (2,2,0), (2,1,1), (2,1,2), (2,2,2), (2,2,1)\}$.

Total possible quirks from all the 27 vectors $= 27 \cdot \left(3^3 \times {}^3C_2 \times \frac{{}^3C_2}{3^2}\right)$

Average number of quirks in members of $vec_3 = \dfrac{\text{Total quirks}}{\text{Number of vectors}}$

$= \dfrac{27}{27} = \underline{1}$.

b) Generalizing:-

Average number of quirks in members of vec_n. For this, we need to find the probability of a pair being a quirk on an average case. For this, the sample space for different pairs in vector of length n is:

$$\frac{\text{Total Possible different quirks}}{\text{Total possible different pairs}}$$

Total possible pairs $= n^2$ because each one can be any value from 0 to $n-1$.

Total possible quirks $= (n-1) + (n-2) + (n-3) + \cdots + 3 + 2 + 1 + 0. = \dfrac{n(n-1)}{2}$.

The above equation is because, when first element is $(n-1)$ we can have any element from $(n-2)$ to 0 (count is $n-1$ elements). Similarly if the first element in pair is $(n-2)$, we can place any value from $(n-3)$ to 0 to form a quirk.

If we go on like that, we will end up with the summation shown.

$\therefore$ Probability of a Pair to be quirk $= \dfrac{\frac{n(n-1)}{2}}{n^2} = \dfrac{\cancel{n}(n-1)}{2n^{\cancel{2}}} = \dfrac{n-1}{2n}$.

Now, In each vector, we can have ${}^nC_2$ Pairs Possible. This is because we can select 2 elements from n elements (vector length) in ${}^nC_2$ ways.

$${}^nC_2 = \frac{n(n-1)}{2}.$$

$\therefore$ Average number of quirks in vector $\in Vec_n = \underbrace{\dfrac{n(n-1)}{2}}_{\text{Total Possible Pairs}} \times \underbrace{\dfrac{n-1}{2n}}_{\text{Prob. of Pair to be quirk}}$

$= \dfrac{(n-1)^2}{4}$.

As there are n^n vectors in Vec_n, Total Possible quirks in all vectors of $Vec_n = n^n \times \dfrac{(n-1)^2}{4}$.

Example:- Take $n = 7$.

Average number of quirks in a vector $\in Vec_7 = \dfrac{(7-1)^2}{4} = \dfrac{36}{4} = 9$.

Total Quirks in all Vectors $\in Vec_7 = 7^7 \times 9$.

6) There will be two Cases for the TABLE-DELETE

i) When there is no table Contract and

ii) When there is a table Contract.

i) When there is no table Contract,

T.Size remains same in step i and step $i-1$.

$$\therefore T.Size_i = T.Size_{i-1} \quad — (1)$$

$T.num_i$ is less than $T.num_{i-1}$.

$$\therefore T.num_i = T.num_{i-1} - 1. \quad — (2)$$

Amortized cost, $\hat{C}_i = C_i + \phi_i - \phi_{i-1}$

$$= 1 + |2 \times T.num_i - T.Size_i| - |2 \times T.num_{i-1} - T.Size_{i-1}|$$

$$= 1 + |2 \times T.num_i - T.Size_i| - |2 \times (T.num_i + 1) - T.Size_i| \quad (\because \text{from } (1) \,\&\, (2))$$

$$= 1 + \cancel{2 \times T.num_i} - \cancel{T.Size_i} - \cancel{2 \times T.num_i} + 2 - \cancel{T.Size_i}$$

$$= 1 + 2$$
$$= 3.$$

$\therefore$ when table is not Contracted,

Amortized Cost $\hat{C}_i = 3$.

Now, we need to see the second Condition where the table Can be Contracted.

ii) when the table Contracts,

$T.Size_{i-1}$ will be Contracted to $\frac{2}{3}^{rd}$ in step$_i$

$$\therefore T.Size_i = \frac{2}{3} T.Size_{i-1} \quad — ①$$

This Contraction happens when we delete an element and then the load factor goes below 1/3.

So, before deletion, the size of Table is $\frac{1}{3}$.

$$\Rightarrow T.num_{i-1} = \frac{1}{3} T.Size_{i-1} \quad — ②$$

As we are deleting an element from table,

$$T.num_i = T.num_{i-1} - 1 \quad — ③$$

In previous Case, $C_i = 1$ because we have only deleted but when resizing happens we need to delete (1 operation) and move the rest of elements ($T.num_i$ operations).

$$\therefore C_i = T.num_i + 1 \quad — ④$$

Amortized Cost, $\hat{C}_i = C_i + |2 \times T.num_i - T.Size_i| - |2 \times T.num_{i-1} - T.Size_{i-1}|$

from equation ①,②,③,④ $= T.num_i + 1 + \left|2\times(T.num_{i-1}-1) - \frac{2.T.Size_{i-1}}{3}\right| - \left|2\times\frac{T.Size_{i-1}}{3} - T.Size_{i-1}\right|$

$$= (T.num_{i-1}-1) + 1 + \left|2\times\left(\frac{T.Size_{i-1}}{3}-1\right) - \frac{2.T.Size_{i-1}}{3}\right| - \left|2\times\frac{T.Size_{i-1}}{3} - T.Size_{i-1}\right|$$

$$= \left(\frac{T.Size_{i-1}}{3} - 1\right) + 1 + \left|\cancel{\frac{2.T.Size_{i-1}}{3}} - 2 - \cancel{\frac{2.T.Size_{i-1}}{3}}\right| - \left|\frac{2.T.Size_{i-1}}{3} - T.Size_{i-1}\right|$$

$$= \frac{T.Size_{i-1}}{3} \cancel{-1} \cancel{+1} + |-2| - \left|-\frac{T.Size_{i-1}}{3}\right|$$

$$= \cancel{\frac{T.Size_{i-1}}{3}} + 2 - \cancel{\frac{T.Size_{i-1}}{3}}$$

$$= 2$$

$\therefore$ when table is Contracted, Amortized cost, $\hat{C}_i = 2$

That means TABLE-DELETE for this strategy is always bounded by a Constant.

7) a) Greedy Algorithm:-

1) Set P = Pennies sum and used = false for all denominations

2) Take Largest denomination d_i with used = false

3) Find $Int(P/d_i)$ which gives the number of d_i to get the sum.

4) $P = P \bmod d_i$

5) set used = True for d_i

6) If $P = 0$ Goto END

7) else Goto step 2.

8) END

This Greedy Algorithm produces optimal values every time for the denomination set {1 cent, 6 cents, 18 cents} but for some cases it will not give optimal result for the set {1 cent, 8 cents, 20 cents}. As we are starting from higher denominations, we are assigning them if $P > D_i$ without checking divisibility with other denominations. In the set {1 cent, 6 cents, 18 cents} every denomination is multiple of the lesser denominations which is helping it to give optimal solution in all the cases unlike {1 cent, 8 cents, 20 cents}.

Example where {1 cent, 8 cents, 20 cents} fails to give optimal solution is $P=35$.

For such P, our Algorithm will give $(20\times1, 8\times1, 1\times7)$ total of 9 coins. But the optimal is $(8\times4+1\times3)$ 7 coins.

7) b) Algorithm:-

1) Arrange all activities in Ascending order of the start time.

2) Build a table with 4 rows and columns number = number of Activities

3) First row is start, second row is end time, Third row is number of Intersections and fourth row is Prior Intersections (Number of activities that start before and have an intersection) for all Activities

4) Any Activity is given (Prior Intersections + 1)th room. If that room is filled, give the least room number less than (Prior Intersections + 1) for the given activity. This is because prior Intersections is the number of previous activities going on when the present activity starts. So it needs a maximum of (Prior Intersections + 1) rooms at that time.

5) Follow the step 4 repeatedly until all the activities ~~elements~~ are done. Maximum value of fourth ~~Third~~ row ~~column~~ plus one (max (row 4) + 1) will give the results of the Maximum rooms used at any time.

Row 3 is not used in the Algorithm so we can ignore it.

Example:-

	A_1	A_2	A_3	A_4	A_5
Start	1	2	4	5	8
END	10	4	8	9	11
Prior Intersections	0	1	1	2	2

Time	Room 1	Rooom2	Room 3	Room 4
1	A_1	-	-	-
2	A_1	A_2	--	-
4	A_1	A_3	-	-
5	A_1	A_3	A_4	-
7	A_1	A_5	A_4	

Maximum rooms used = 3, which satisfies the formula Max(row3)+1 = 2+1 = 3

8) a) If we can get Independent sets, we can check that in Polynomial time whether 4 colors can be usable to color the graph. We need to prove that 4 coloring of a graph is np-complete. For this, we need to take the already known np-complete Problem 3-color problem and reduce it to 4-color problem. By this way we can prove that 4-coloring is np-complete Problem.

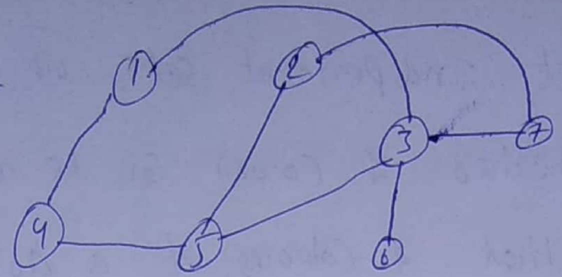
Procedure:-

1) Take the graph and consider a new node n.
2) Draw an edge to each vertex in the existing graph
3) This makes the new node n to get a different color (additional color).
4) Now proving the new graph is 4-colorable is same as proving the old graph is 3-colorable.
5) Thus we ended up reducing a 3-color Problem to 4-color Problem.

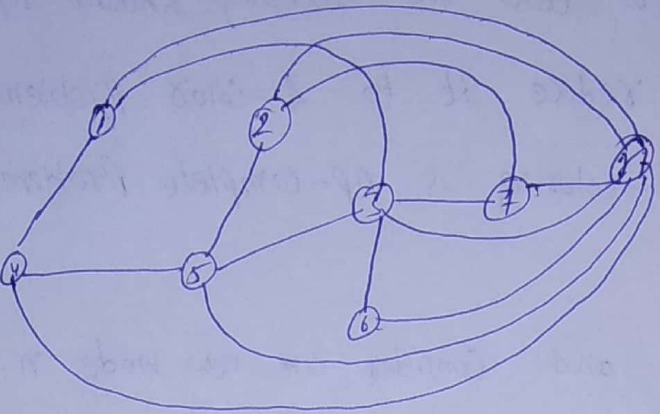
Now, from the properities of np-complete Problems, we can conclude that 4-colorable problem is in np-complete.

b) Like the way we added a new node and connecting each existing vertex to the new node by adding edges will force the vertex to take a new color. By this way, we can say that we can add x vertices to the graph one by one with the above stated condition proves that Problem with $K = x+3$, proves that $K(??)$ colorable is in np-complete.

Example:- Graph G =



add vertex n:



n is Connected to each existing vertex. Now proving graph - n is 3 colorable which is in np-complete proves that the new graph is 4 Colorable. Repeating this process of adding new vertices will prove that for K>4 and determing whether graph is K-colorable is an np-complete problem.

---

② The Program (Python) has been included in the zip file with the name 2.py.

---

⑤ a) Python Programs 5a.py and 5a2.py are implemented for recursive and dynamic Programming respectively.

5a.py uses ~~unti~~ repetetive calls to G(n) and won't show up the result due to so many recursive calls. But 5a2.py stores each value in the table and shows the result of G(500) almost immediately.

5) b) The Program 5b.py is Normal recursive call which will not result in any output for H(500) because of too many recursive calls.

5b2.py stores all the outputs of each recursive call and thus avoids the calculation each time.

5b3.py stores only last 3 recursive call outputs as they are enough to calculate the next value. So, it is the efficient (space) of implementing 5b2.py.

---

1) b) Given,

$$f(x) = 2\sin^2(x) - 4\sin^4(x) \quad \&$$

$$g(x) = \cancel{2\cos^2(x)}\ 2\cos^4(x) + 5\cos(x)$$

From the definition of O:

$f(n) \in O(g(n))$ when

$\exists C > 0 \ \& \ \exists n_0 \in N,\ n_0 \geq 1$

such that $|f(n)| \leq C|g(n)| \quad \forall x \geq x_0$.

Let us assume that $f(x) \in O(g(x))$ (Proof by Contradiction)

$\therefore |f(x)| \leq C|g(n)|$ for some $C > 0$, $n > n_0 \in N$.

As $f(x), g(x)$ are in $\sin x$, $\cos x$ terms, they will give different results from 0° to 360° only for x value. If x is more than 360°, they will repeat the values in the same order. So, lets see the function holds in 4 different intervals $0^\circ, 90^\circ, 180^\circ, 270^\circ$.

for 0°, $|f(0)| = 0$, $C|g(0)| = 7.C$

for 90°, $|f(90)| = 2$, $C|g(90)| = 0$

for 180°, $|f(180)| = 0$, $C|g(180)| = 3.C$

for 270°, $|f(270)| = 6$, $C|g(270)| = 0$

From the above values, we can say that the inequality:

$$|f(x)| \leq c\,|g(x)|$$

will not hold for any value of x, c as they are overlapping and this will continue for any value of x. for any x, $|f(360 \times n + 90)| \geq c|g(360 \times n + 90)|$ for any n.

So, by contradiction, $f(x) \notin O(g(x))$.

Now, assume $g(x) \in O(f(x))$

$\Rightarrow |g(x)| \leq c\,|f(x)|$ for some $c > 0$, $x > x_0$.

Take $0°, 90°, 180°, 270°$ for x and see the comparision. If x is increased by $360°$, the values are repeated.

for $0°$, $|g(0)| = 7$, $c|f(0)| = 0$

for $90°$, $|g(90)| = 0$, $c|f(90)| = 2 \cdot c$

for $180°$, $|g(180)| = 3$, $c|f(180)| = 0$

for $270°$, $|g(270°)| = 0$, $c|f(270)| = 6 \cdot c$

from the above values, we can say that $|g(x)| \leq c|f(x)|$ will not always hold. take for example $|g(n \times 360)| \geq c|f(n \times 360)|$. This will stay true for any n.

So, by contradiction,

$$g(x) \notin O(f(x)).$$

```
C:\Users\mprav\Desktop\CS 5200 final python>py 8a2.py
21354641739573893477279411178449377737569899092639453775895870570975006915873346065610819405691957713899246806099708361З
22154885470915

C:\Users\mprav\Desktop\CS 5200 final python>py 8b2.py
6

C:\Users\mprav\Desktop\CS 5200 final python>py 8b3.py
6

C:\Users\mprav\Desktop\CS 5200 final python>
```