University Of Calgary

Winter 2020

CPSC 413

FINAL ASSESSMENT

Minh Hang Chu

3007 4056

Due : Saturday, April 18 12PM

① Short answer questions

a) If $f(n) = 4^{\sqrt{n}}$ and $g(n) = 2^{\sqrt{n}}$, then $f(n) \in o(g(n))$

This statement is false.

In class, we use a fact: $f(n) \in o(g(n)) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

We compute:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{4^{\sqrt{n}}}{2^{\sqrt{n}}} = \lim_{n\to\infty} \left(\frac{4}{2}\right)^{\sqrt{n}}$$

$$= \lim_{n\to\infty} 2^{\sqrt{n}}$$

$$= \infty$$

|| Thus, $f(n) \notin o(g(n))$.

b) Consider the recurrence:

$$T(n) = 16\,T(n/b) + n^2 \lg n$$

Give value for constant $b > 1$ such that:

i) $T(n) \in \Theta(n^4)$

Using Master Theorem, we have

$a = 16$ $\quad b > 1$ $\quad f(n) = n^2 \lg n$

We want $T(n) \in \Theta(n^4)$, which means we want to find b that satisfies the first case. Since case 2 & 3, we will get $T(n) \in \Theta(n^2 \lg n)$ - not what we want.

Hence, we want to find b so that:

$f(n) = n^2 \lg n \in O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$

We choose $b = 2 \Rightarrow \log_b a = \log_2 16 = 4$

Then we choose $\varepsilon = 1 \Rightarrow n^{\log_b a - \varepsilon} = n^{4-1} = n^3$

We check if $f(n) \in o(n^3)$:

$$\lim_{n \to \infty} \frac{f(n)}{n^3} = \lim_{n \to \infty} \frac{n^2 \lg n}{n^3} = \lim_{n \to \infty} \frac{\lg n}{n} = 0$$

Thus, $f(n) \in o(n^3) \Rightarrow f(n) \in O(n^3) = O(n^{\log_b a - \varepsilon})$

This condition satisfies, then $T(n) \in \Theta(n^{\log_b a}) = \Theta(n^4)$

Hence, we choose $\boxed{b = 2}$ to satisfy $T(n) \in \Theta(n^4)$

according to the first case in Master Theorem.

ii) The Master Theorem cannot be used to determine an asymptotic tight bound for $T(n)$

We still have $a = 16$ $b > 1$ $f(n) = n^2 \lg n$

We choose $b = 4 \Rightarrow \log_b a = \log_4 16 = 2$

- Condition 1: $f(n) = n^2 \lg n \in O(n^{2-\varepsilon})$

No. $f(n) > n^{2-\varepsilon}$ for some $\varepsilon > 0$

- Condition 2: $f(n) = n^2 \lg n \in \Theta(n^2)$

No. This case does not apply

- Condition 3: $f(n) \in \Omega(n^{2+\varepsilon})$

No. This case also does not apply since

$n^2 \lg n < 2^{2+\varepsilon}$ for all $\varepsilon > 0$

Hence, Master Theorem tells us nothing

$\rightarrow$ Master Theorem cannot be used to determine asymptotic tight bound for $\boxed{b = 4}$.

c) True or False: If Vertex Cover $\in P$, then $P = NP$

True.

We know that Vertex Cover and Clique are equivalent problems

Clique $\in$ NP-Complete $\Rightarrow$ Vertex Cover $\in$ NP-Complete

From Theorem 1 in Topic 7, we have (slide 40)

let Vertex Cover $\in$ NP-Complete Then

- Vertex Cover $\in P \Rightarrow P = NP$.

Thus, the statement is true.

d) True or False: If a decision problem $A \in NP$, then a certificate that a given instance of A is a "yes" instance can be computed in polynomial time.

False.

$A \in NP$ means there exists a polynomial-time verification algorithm does not mean we can compute ~~th~~ certificate in polynomial time.

② Divide and Conquer. $T(n) = \begin{cases} 2T(\lfloor n/5 \rfloor) + 2T(\lceil n/5 \rceil) + cn^2 & n \\ d & \text{otherwise} \end{cases}$

a) Use Master Theorem to guess a tight asymptotic bound

We can drop $\lceil \; \rceil$ and $\lfloor \; \rfloor$ to get an approximation

We have:

$$T(n) = 2T\left(\frac{n}{5}\right) + 2T\left(\frac{n}{5}\right) + cn^2$$

$$= 4T\left(\frac{n}{5}\right) + cn^2$$

Apply Master Theorem, we have $a=4$ $b=5$ $f(n)=cn$

$\Rightarrow \log_b a = \log_5 4 < 1$

We show the third case applied: $f(n) \in \Omega(n^{\log_b a + \epsilon})$

for some $\epsilon > 0$ and $af(n/b) \leq c' f(n)$ for some $c' < 1$

- $f(n) = cn^2 \in \Omega(n^{\log_5 4 + \epsilon})$ choose $\epsilon = -\log_5 4 + 1$.

  $f(n) \in \Omega(n)$

- $4c\left(\frac{n}{5}\right)^2 \leq c' cn^2 \Rightarrow \frac{4n^2}{25} \leq c' n^2$

  Choose $c' = \frac{1}{2}$ $\rightarrow$ this condition also satisfy.

$\Rightarrow$ Thus, $T(n) \in \Theta(f(n))$

$\Rightarrow T(n) \in \Theta(n^2)$

Our guess for the tight asymptotic bound for $T(n)$ is

$\Theta(n^2)$

b) Consider the recurrence

$$T(n) = \begin{cases} 2T\left(\lfloor \frac{n}{4} \rfloor\right) + cn^2 & \text{if } n > 4 \\ d & \text{otherwise} \end{cases}$$

Prove by induction that $T(n) \leq an^2 \quad \forall n \geq 1$

<u>Base case</u>: $1 \leq n \leq 4$

$T(n)$ is bounded by a constant d, so if we pick $a \geq d$, we have $T(n) \leq d \leq an^2$.

<u>Inductive Hypothesis</u>: Assume that $T(K) \leq aK^2$ for $1 \leq K < n$

<u>Inductive step</u>: We show $T(n) \leq an^2$:

$$T(n) = 2T\left(\lfloor \frac{n}{4} \rfloor\right) + cn^2$$

$$\leq 2a\left(\frac{n}{4}\right)^2 + cn^2 \qquad \left(\text{by inductive hypothesis } \frac{n}{4} < n\right)$$

$$= 2a\frac{n^2}{4^2} + cn^2$$

$$= \frac{an^2}{8} + cn^2$$

$$= n^2\left(\frac{a}{8} + c\right)$$

Now, we want $T(n) \leq an^2 \Rightarrow$ we want $\frac{a}{8} + c \leq a$

$$\Rightarrow c \leq \frac{7a}{8} \Rightarrow \frac{8c}{7} \leq a$$

Then: $T(n) \leq n^2\left(\frac{a}{8} + \frac{7a}{8}\right)$

$$= an^2 \Rightarrow \text{Choose } a = \max\left(d, \frac{8c}{7}\right)$$

Hence, $T(n) \leq an^2 \quad \forall n \geq 1$.

③ Dynamic Programming

a) Give recursive expression (optimal substructure)

Let $M[i]$ denote the maximum amount of money that can be earned over i days.

We have:

$M[0] = 0$ → no working day

$M[1] = r \times h_1$

$M[2] = r \times h_1 + r \times h_2$

} can only work for A, job at company B needs 3 days at least

$M[3] = \max(M[0] + 3c \,;\, M[2] + rh_3)$

⋮

$M[n] = \max(M[n-1] + rh_n \,;\, M[n-3] + 3c)$

We have:

$$M[n] = \begin{cases} 0 & \text{if } i = 0 \\ rh_i + M[i-1] & \text{if } 1 \leq i \leq 2 \\ \max(M[n-1] + rh_i \,;\, M[n-3] + 3c) & \text{otherwise} \end{cases}$$

b) Give a polynomial time dynamic programming algorithm

- 1 to compute the optimal value
- 1 to recover job schedule..

```
Job Schedule(r, c, H, n)
    M[0] = 0
    M[1] = r x h1
    P[1] = A
    M[2] = r x h1 + r x h2
    P[2] = A
    for i = 3 ... n do
        if M[i-1] + rhi > M[i-3] + 3c
            M[i] = M[i-1] + rhi
            P[i] = A
        else
            M[i] = M[i-3] + 3c
            P[i] = B
        endif
    end for
    return M[n], P

JobScheduleRecover(P, n)
    j = n
    while j ≥ 1 do
        if P[j] = A then
            Plan[j] = A
            j = j - 1
        else
            Plan[j] = B
            Plan[j-1] = B
            Plan[j-2] = B
            j = j - 3
        endif
    end while
    return Plan
```

c) State the asymptotic running time of your algorithm

First, we consider JobSchedule :

- From line 2 - 6, the execution takes constant steps c_1.
- We have a for loop that iterates $n-2$ times at line 7. In the loop body, we can see that all steps take constant time c_2. The total steps : nc_2.
- Finally the return takes constant step: c_3

⇒ This function rus in $c_1 + nc_2 + c_3 \in \Theta(n)$

Next we consider JobScheduleRecover :

- set j takes constant time at line 18 ; c_1
- the while loop iterate n times. The loop body takes constant steps. The total steps for this loop is nc_2.
- The return statement takes constant time : c_3

⇒ Hence, JobScheduleRecover takes $c_1 + nc_2 + c_3 \in \Theta(n)$

Therefore, both JobSchedule & JobScheduleRecover are in $\Theta(n)$

④ NP - Completeness

★ Prove Sales Route Selection (SRS) $\in$ NP

- State the input to the verification algorithm:
  - A directed graph $G=(V,E)$, a set of Route requests $R_1, R_2 \dots R_s$ (each R_i is a simple path) and an integer $K \geq 1$. (input of SRS)
  - A set of routes $P = \{P_1, P_2 \dots P_u\}$, subset of $R_1, R_2, \dots R_n$ (certificate)
- Certificate is polynomial in size to the remaining input.
  - Simple path of G is a sequence of zero or more edges. Let denote an edge (v_a, v_b) that $v_a, v_b \in V$. Each $R_i \in$ Route are simple path, each R_i will have at most $|V|$ edges, which means $2|V|$ vertices. Route has s elements, the size of Route is $2s|V|$
  - A set of routes $P \subseteq$ Route will also have the size at most $2u|V|$, polynomial in size of input.
- Give the verification algorithm

```
if K > u
    return no
endif
for each Pi in P1 ... Pu
    if Pi has no edge
        do nothing and go to next iteration
    else
        for each edge ej in Pi
            match ej = (a,b)
```

```
10              if j = 1
11                  if a or b are colored then
12                      return "no"
13                  else
14                      color a & b are visited
15              else
16                  if b is colored them
17                      return "no"
18                  else
19                      color b as visited.
20              endif
21          end for
22      endif
23  end for
24  return "Yes"
```

- Prove that the algorithm is correct
  - If G, K, Route is a "Yes" instance of SRS, there there are no routes in P has same vertices. First it checks if the number of routes in P is less than number routes requested (line 1-3).
    - Then we check every routes in P and each edge in that route. If route does not have any edge, move on to check next routes
    - If route does has edges, check vertices are visited or not. We check both vertices of the first edge in the route, if they are not visited (colored), we color them to mark as visited. Since this is ~~is~~ a yes instance, both vertices are not visited and will be colored (line 10-15)

(line 15-19)

+ Then we check the rest edges in that route. Since we are considering Yes instance, means no vertices are repeated, and vertices are not colored. We mark those vertices as visited and keep looping for all iterations.
+ After all iterations, vertices are colored, the execution exited the loop and return yes at line 24

- If S, k, Route are a No instance of SRS, means there are routes share the same vertex.
  + First, if P has more routes than Route, return no since $P \subseteq$ Route.
  + Then we check the first edge, if vertices are visited, the algorithm returns no at line 11-12
  + Then the algorithm check if the next vertex in the route ~~as~~ is colored / visited. Since this is a No instance, at some point, one vertex will appear to be visited, the algorithm returns no at line 16-17.

• Show that the verification algorithm runs in polynomial time:

We found that each iteration marks the vertices visited in each route using loop. If it visits a vertex twice, it stops. Hence, the worst case, the algorithm will visit each nodes once.

Hence, the loop will iterate $|E|$ times.

We also see that inside the loop body, all steps take constant steps.

$\Rightarrow$ The run time is in $O(|E|)$, which is polynomial in the size of input

Therefore, $SRS \in NP$

Next, we prove that 3-Dimentional Matching (3DM) $\leq_p$ SRS

knowing 3DM is in NP-Complete

- Give an algorithm to transform the input 3DM to input of SRS

```
TransformtoSRS (X, Y, Z, T)
    V = X U Y U Z
    E = []
    Route = []
    for each (x, y, z) in T:
        E.add ((x,y))
        E.add ((y,z))
        Route.add ((x,y),(y,z))
    K = X.size()
    return G= (V, E), S, K
```

- Prove that the transformation algorithm runs in polynomial time.
  - First, union X, Y, Z to get all vertices. Each set has size n, we assume that this takes $3c_1 n$ steps
  - Next, at line 3 + 4, we will need to initialize E and Route. This step takes constant time. c_2.
  - Next, we have for loop for all elements in T. Since T is the cross product of X, Y, Z, T will have at most n^3 elements. Hence, the ~~foor~~ for loop will iterate at most n^3 times. The body loop takes constant steps c_3. Hence, the total steps this loop take is $n^3 c_3$
  - Line 9 and 10 will return the input for SRS, which takes constant time

Hence, put everything together, we will get the run time of the algorithm is in $\Theta(n^3)$

- Let s be an input to 3DM and s' be the transformed input to SRS. Prove s is a Yes instance of 3DM if and only if s' is a yes instance of SRS

- Suppose (X, Y, Z, T) is a yes instance of 3DM, we see that $V = X \cup Y \dot{\cup} Z$ and T has (x, y, z), E has (x, y) + ~~case~~ (y, z) and $R_i = ((x, y)(y, z))$ - which is a simple path because x, y, z are distinct.

When SRS runs with G, Route & $K = n$ is a Yes instance of 3DM. We know that we have n ~~tuples~~ triples (x, y, z) and they are all distinct, Then a subset of the triple in T will cover all elements of the 3 sets exactly once Then we will also have n paths that no 2 selected routes go through the same towns

- Suppose (X, Y, Z, T) is a no instance of 3DM, means there does not exist a set of n triples in T. so that each element of $X \cup Y \cup Z$ is contained in exactly one of these triples, then SRS does not have $k = n$ routes so that no 2 of the selected routes go through the same ~~time~~ town. SRS will output no on input $G=(V, E)$ k and Route.

Hence, we have shown all conditions for SRS $\in$ NP-Complete

$\square$

THANK YOU!