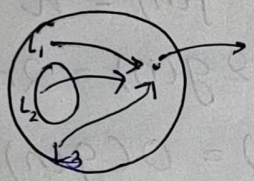
Q1

i) if a decision problem in P is NP-complete then P = NP. Recall that P denotes a decision problem which can be solved in polynomial time. (YES).

As we know NP-complete problems (P) are those problem

i) which lies in NP.

ii) All problems of L of NP $L \leq_P$ problem (P)



Claim: if p can be solved in polynomial time then P=NP.

$\Rightarrow$ $p \in P$ (given).

$\forall L_{in\ NP} \Rightarrow L \leq_P P$ (from def^n above).

$\Rightarrow$ all problems in NP are polynomial time reducible to p, Also as p can be solved in polynomial time L can be solved in polynomial time too.

$\Rightarrow$ there will be no problem in NP which can not be solved in polynomial time (NP - P = Null)

$\Rightarrow$ P = NP. (as $P \subseteq NP$ & $NP - P = \phi$).

ii) safe vertex if removal of v does not disconnect G.

(NO).

Let's take following example

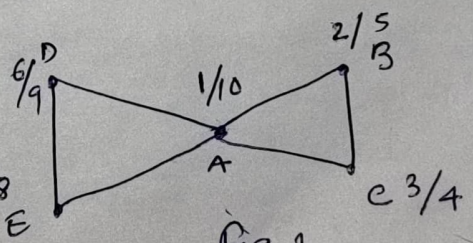


Fig 1

G has DFS Tree as.

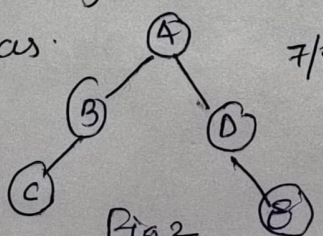


Fig 2

As from Fig 1 we can say that- removal of A disconnect the graph But- in Fig 2 DFS has more than 2 child

iii). if $f(n) = O(g(n))$ then $2^{f(n)}$ is $2^{O(g(n))}$ (NO)

$f(n) = O(g(n)) \Rightarrow \exists c > 0$ such that- $f(n) < c\,g(n)$ for all $n > n_0$

$\Rightarrow f(n) \leq c\,g(n)$ let's take $\left.\begin{array}{l} f(n) = n \\ 2g(n) = 2n \end{array}\right]$ A

we know $f(n) = O(g(n))$.

now if we take power on both side it will be

$\left.\begin{array}{ll} 2^{f(n)} & 2^{n} \\ \leq 2^{g(n)} & 2^{2n} \end{array}\right]$ B.

- In case A ratio between two is constant which can be used to set c.
- but in case B ratio between these two is 2^n which can't be written using c

$\Rightarrow$ above given statement is false-

2)

Given S: $n+1$ numbers. (distinct-) (unsorted)
A: n numbers from S (unsorted)

$O(n)$ time DC algorithm to find the number which is not in S.

- As we know if $T(n) = T(n_1) + T(n_2) + O(n)$
  where $n_1 + n_2 \leq \alpha n$
  then the sol^n of this recurrence is $O(n)$
- So we need to divide & conquer using throwing ~~sum~~ some part of the array.
- we need to find out the median for both arrays A & S and based on the position of median we can say that which part has missing element.
- if we refine the solution more we can use the partitioning algorithm to set the position of an element in S and A and for next iteration select the ~~size of~~ sides of S & A which has unbalance sizes.

Algorithm: Find_missing (A, S)

O(1) - // check base cond^n here (defined next)
O(1) - select random number e from A
O(n) - find the position of e in S
O(n) - set e as pivot in both A & S and do partitioning as discussed in class
this gives us A_1 e A_2 and S_1 e S_2

if ($|A_1| \; != |S_1|$)

call find_missing (A_1, S_1).

else

Call find_missing (A_2, S_2)

} // END of algorithm.

base condition (Should be at start of recursion) [if ~~size~~ $|S_3| = 1$ and $|A_3| = 0$ return the element of S as answer.]

Complexity: ~~As discussed in class~~ we are calling only one subarray. so ~~as the~~ it has complexity of selection algorithm where one part of the array is thrown off and work done in each iteration is $O(n)$.

our recurrence eqn is also like

$$T(n) = T(n_1) + O(n) \quad (\text{where } n_1 < \alpha n)$$

Since e_1 is chosen randomly we can say that it has decreasing the size of problem as a factor

$$\Rightarrow \quad T(n) = O(n).$$

Proof of correctness:

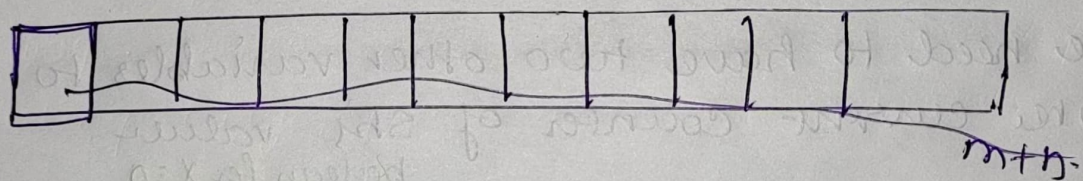
As we are selecting e in A $\Rightarrow$ e will be in S too, now if we do partition elements less than e are in first part & greater than are in second part of array S e T

Since all elements are distinct we can simply compare the size of first & second part to find out which part has missing number (which is easy embalance) //

3) Shuffel of two strings

a) given 3 strs $A[1,..n]$ $B[1..m]$, $C[1..m+n]$ we give a DP ~~that~~ to check whether C is a shuffle of A & B.

this problem is same as ~~graph~~ string edit-distance where at each point we should check if ith element of C is in order elements of either A or B.



Shuffel (~~A, B, C~~ Ai, Bj, Ck, value) {

base cond.

if ($C_k = A_i$)

~~value shuffel = shuffel~~

value ~~=~~ = ~~value~~ 1 + shuffel (A_{i+1}, B_j, C_{k+1}, value).

if ($C_k = B_j$)

value = 1 + shuffel (A_i, B_{j+1}, C_{k+1}, value).

}

base cond^n.

if $C_k \neq A_i$ & $C_k \neq B_j$ then

return no ~~value~~ valid shuffel

if value = m+n

return C is valid shuffel.

Idea is to count valid matches using A, B to C and incrementing pointer appropriately.

complexity time ⇒ $O(m+n)$

space ⇒ $O(1)$ only one variable value is used.

b) smooth shuffle, no more than two shuffles are done

X Y Z.

we need to have two other variables to store current counter of shi value

boolean for X = 0, Y = 0

S_shuffel (X_i, Y_j, Z_k, sh, count, value).

{

if value = m+n.
return valid shuffel

if $Z_{k+1} = X_i$ and $Z_{k+1} = Y_j$
return invalid shuffel.

if counter > 2
return invalid shuffel.

if $Z_k = X_i$

value = 1 + S shuffel (X_{i+1}, Y_j, Z_{k+1}, 0, (count+1).

if $Z_k = Y_{j+1}$

value = 1 + S shuffel (X_i, Y_{j+1}, Z_{k+1}, 1, (count+1))

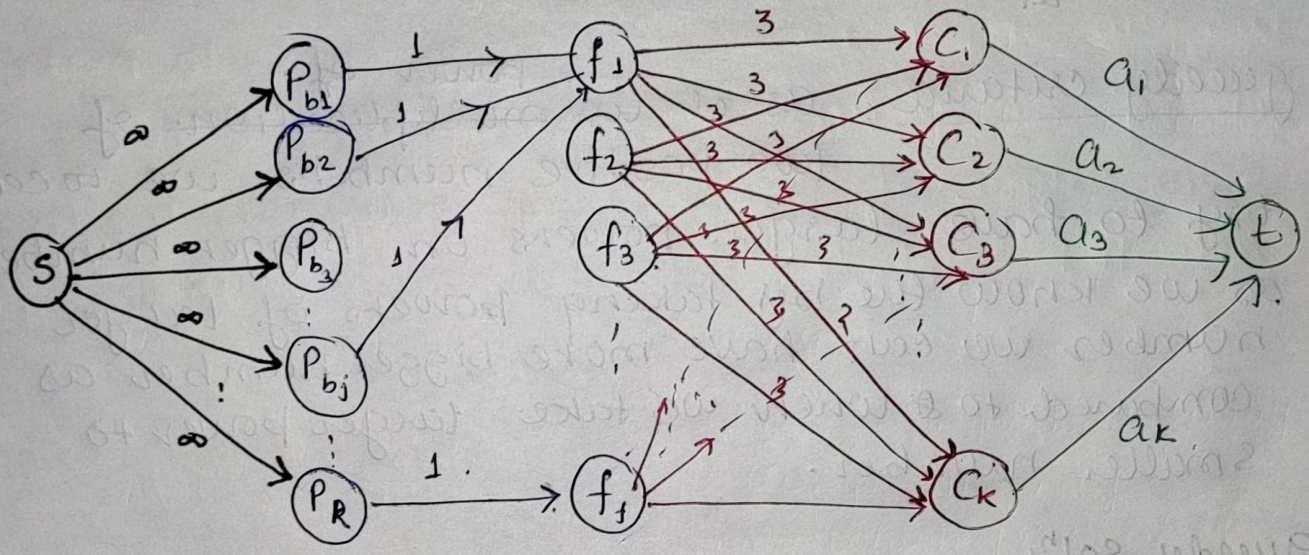
4) f: families

K: clubs. ith club needs - a_i number of memb.
- can take at most 3 from a family

each person can belong to at most 1 club.

jth family has b_j members.

1- each member has to be in a family. (atmost 1).



source can be connected to each person with capacity ∞ or any other capacity $C > 1$	each p_i is connected to f_j if $p_i \in f_j$ by a directed edge of capacity 1.	each f is connected to each c with a directed edge of capacity 3.	each club i is connected by t with capacity a_i
constrain that is satisfied: None	each person can be atmost in one family	each club can have atmost 3 members from a family	Club has to take a_i members in total

Q.5.

Near clique ${}^kC_2$ or ${}^kC_2 - 1$ edges

To prove clique $\leq_p$ Near clique.

Clique $\xrightarrow{\text{polynomial time reducible to}}$ Near clique.

$G(V, E)$ $G'(V, E')$.

we can simply remove exactly one edge in G to make it G' at a time and ask if it has a near clique of size k. if answer is yes to all $|E|$ graphs G' then we can say that clique of size k existed in graph G.

Proof :- if G has a clique of size k then ~~all~~ $\forall\, G'_i(V,E)$ $i \in [0,E]$ will ~~answer~~ yes have near clique

This is easy to prove that if G has a clique of size k then G' can ~~have~~ only have a clique or near clique of same size (k), as our algorithm for finding clique is called $O(E)$ times which is polynomial we can say clique is polynomial time reducible to near clique.

- if all $G'_i(V,E)$ s.t $i \in [0,E]$ has near cliques then G has a clique.

as similar to above G' only has one edge missing at a time $\Rightarrow$ if near clique is not-present then the clique of size k will no longer be present in G. & conversely if for all G' if near clique present $\Rightarrow$ G will have a clique

6) given X: n +ve int
Y: "
arrange the element in X & Y in some order
x_i ith element & y_i ith element in this order
arrange such that-

$$\prod_{i=1}^{n} x_i^{y_i} = x_1^{y_1} \times x_2^{y_2} \times x_3^{y_3} \times \cdots \times x_n^{y_n}.$$

Greedy criteria: as it is power of multiplication of two positive numbers we can try to have larger powers on bigger numbers as we know that by taking powers of larger number we can have more bigger number as compared to when we take larger power to smaller number.

Greedy soln:
X' = sort X in decreasing order
Y' = sort Y in decreasing order.
calculate the $\prod_{i=1}^{n} x_i^{y_i}$,

Proof of correctness.
in first step greedy select the first two elements of X' & Y'. & optimal takes any other 2 i & k for Y' & X'
Base step:
greedy X'[1] Y'[1]
optimal X'[k] Y'[i]

as we can say.

$x[i]^{r'[i]}$ is the max^m possible product

⇒ greedy is ahead of optimal in first step.

Also let's assume greedy matches $x[k]$ to $y'[k]$ & $x[i]$ to $y'[i]$

⇒ $x'[i]^{r[i]} \times x'[k]^{y'[k]}$ ~~$x'[t]^{[t]}$~~ is the (g1)

- partial product involve in greedy

on the other hand optimal taken.

$x'[k]^{[i]} \times x'[i]^{[t]}$ ~~$\times$~~ —— (o1)

the multiplication of (o1) will be less than (g1) always. as the values here are encreasing for taking larger power of larger elements.

Induction:

Claim: if greedy solution agrees to optimal in $n-1$ step then we should prove the it will agrees for n steps too

proof: let

Greedy $O_2 \quad O_3 \quad --- \quad O_n$

Optimal $G_2 \quad G_3 \quad --- \quad G_n$ (as per IH)

then by combining prove in above case for $n=1$ we can say that-

Greedy $O_1 \quad | \quad O_2 \quad O_3 \; -- \; O_n$

optimal $G_1 \quad | \quad G_2 \quad G_3 \; -- \; G_n$

agrees with each other.