Computer Science COMP-251A
Midterm, Oct 17, 2016, 14:35-15:55.
O P E N • B O O K S • / • O P E N • N O T E S

1) Consider the following algorithm:

Expo(A,e)

IF e=0 **THEN RETURN** ▌ (*the identity matrix of same size as A*)
ELSE IF e is even **THEN RETURN** Expo(A^2,e/2)
ELSE RETURN A*(Expo(A^2, (e-1)/2))

[30%]

Given an nxn matrix A, consider a recursion T(n) describing the running time of the above algorithm with e=n,
i.e. T(n) = (worst case) running-time of computing A^n using Expo(A,n).

a) Write T(n) under the assumption that matrix products are computed by the (brute force) standard definition of matrix product. Use the master theorem to solve T(n).

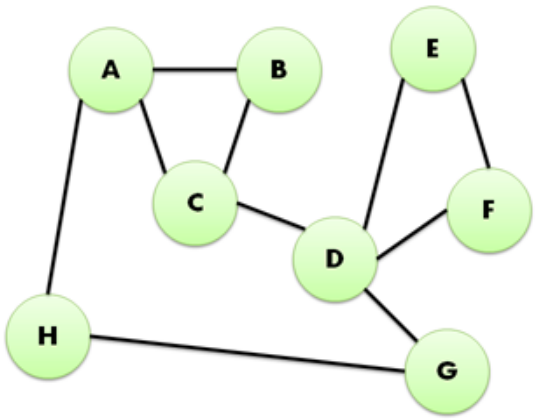
b) Rewrite T(n) under the assumption that Strassen's algorithm is used for matrix products. Use the master theorem to solve T(n).

2) For each of the two graphs below

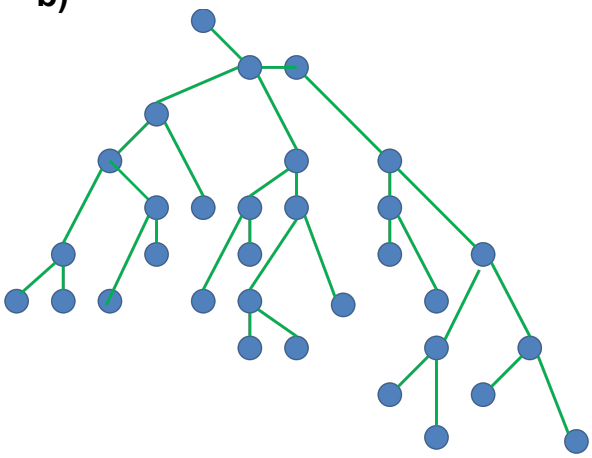
[50%]

say if they are bipartite or not and explain why.

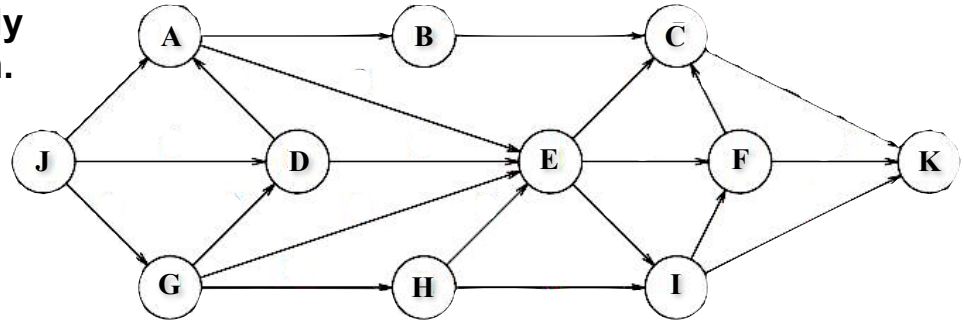
a)



b)



c) Topologically sort this graph. Provide a step-by-step run.

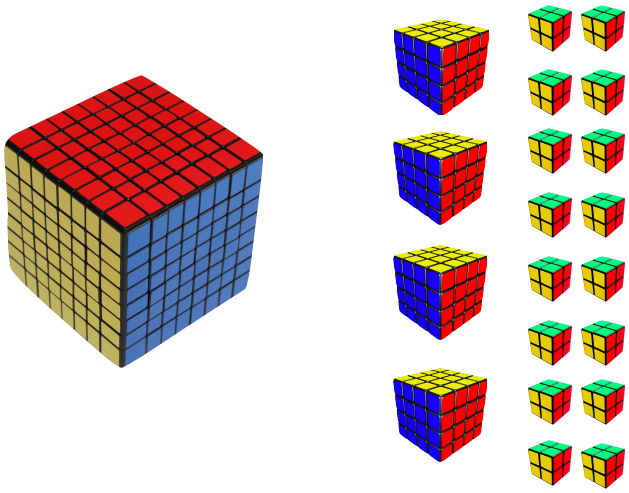


Computer Science COMP-251A
Midterm, Oct 17, 2016, 14:35-15:55.

d) Provide an instance of the Stable-Matching problem with 3 men (A,B,C) and 3 women (X,Y,Z) that would lead both men-optimal and women-optimal versions of Gale-Shapley to the same (unique) solution. Explain why the solution must be both men-optimal and women-optimal.

e) I wrote in the course slides that solving an nxnxn Rubik's cube can be solved optimally in $\Theta(n^2/\log n)$ rotations. If I asked you to find the number of rotations used to perform the following task (Work(n)), why will you have a problem giving me an exact asymptotic answer ?

$$\text{Work}(n) = \begin{cases} \text{nothing needs to be done} & \text{IF } n=1 \\ \text{solve an } n\text{x}n\text{x}n \text{ cube and run 4 times Work}(n/2) & \text{IF } n>1 \end{cases}$$



[+5%] **BONUS:** argue that the solution would be $\Theta(n^2 \log \log n)$ rotations.

[30%] **3)** Explain how using **RED-BLACK** Trees you may implement a priority queue (containing *n* elements) such that both related operations (see below for a reminder) are at most $O(\log n)$-time.

A **priority queue** must at least support the following operations:

insert_with_priority: add an element to the queue with an associated priority.

pull_highest_priority_element: remove the element from the queue that has the highest priority, and return it.