2015 Final

Tuesday, July 16, 2019 11:11 PM

final_2015

Design and Analysis of Algorithms May 20, 2015
Massachusetts Institute of Technology 6.046J/18.410J
Profs. Erik Demaine, Srini Devadas, and Nancy Lynch Final Exam

Final Exam

- Do not open this exam booklet until you are directed to do so. Read all the instructions first.
- The exam contains 9 problems, with multiple parts. You have 180 minutes to earn 180 points.
- This exam booklet contains 18 pages, including this one.
- This exam is closed book. You may use three double-sided letter ($8\frac{1}{2}'' \times 11''$) or A4 crib sheets. No calculators or programmable devices are permitted. Cell phones must be put away.
- Do not waste time deriving facts that we have studied. Just cite results from class.
- When we ask you to "give an algorithm" in this exam, describe your algorithm in English or pseudocode, and provide a short argument for correctness and running time. You do not need to provide a diagram or example unless it helps make your explanation clearer.
- Do not spend too much time on any one problem. Generally, a problem's point value is an indication of how many minutes to spend on it.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Please be neat.
- Good luck!

Q	Title	Points	Parts	Grade	Q	Title	Points	Parts	Grade
1	True or False	56	14		6	Be the Computer	14	3	
2	Überstructure	10	1		7	Startups are Hard	20	3	
3	Meancorp	15	2		8	Load Balancing	15	2	
4	Forgetful Forrest	15	3		9	Distributed Coloring	20	3	
5	Piano Recital	15	3		Total		180		158

Name: George Ong

Problem 1. True or False. [56 points] (14 parts)

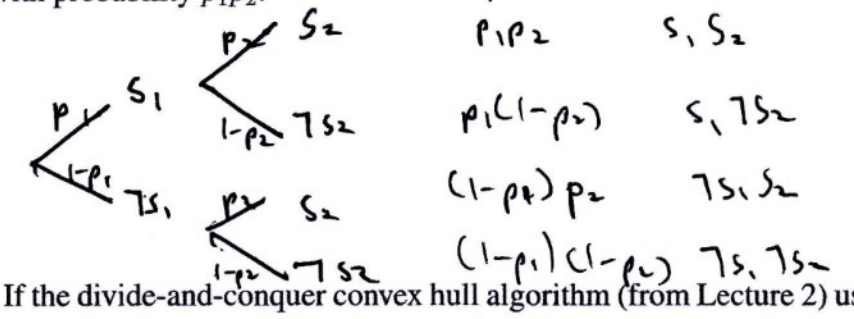
Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false and briefly explain why.

X

−4

Requires independence

(a) (T) F [4 points] Suppose algorithm $\mathcal{A}$ has two steps, and $\mathcal{A}$ succeeds if both the steps succeed. If the two steps succeed with probability p_1 and p_2 respectively, then $\mathcal{A}$ succeeds with probability $p_1 p_2$.



(b) T (F) [4 points] If the divide-and-conquer convex hull algorithm (from Lecture 2) used a $\Theta(n^2)$ strategy to discover the maximum and minimum tangents, the overall algorithm would run in $\Theta(n^2 \log n)$ time.

$T(n) = O(n^2) + 2T(n/2)$ Divide : $O(\lg n)$

Conquer : $2T(n/2)$

Combine : $O(n^2)$

Master's Theorem Say it is $\Theta(n^2)$

(c) T (F) [4 points] In order to get an expected $\Theta(n \log n)$ runtime for "paranoid" quicksort (from Lecture 3), we require the recursive divide step to split the array into two subarrays each of at least $\frac{1}{4}$ the size of the original array.

False, any fraction will work

T

(d) [4 points] A binary min-heap with n elements supports INSERT in $O(\log n)$ amortized time and DELETE-MIN in 0 amortized time.

We ~~can't amortize across methods~~ amortize delete costs by pre-paying during insert.

deletes are bounded by number of inserts/size

(e) T F [4 points] The hash family $H = \{h_1, h_2\}$ is universal, where $h_1, h_2 : \{1, 2, 3\} \to \{0, 1\}$ are defined by the following table:

	1	2	3
h_1	0	1	0
h_2	1	0	1

(For example, $h_1(3) = 0$.)

guaranteed collision for key pairs (1,3)

$Pr(h(1)=h(2)) = \frac{1}{2}(0+0) \leq \frac{1}{2}$

$Pr(h(2)=h(3)) = \frac{1}{2}(0+0) \leq \frac{1}{2}$

$Pr(h(1)=h(3)) = \frac{1}{2}(1+1) = 1 \not\leq \frac{1}{2}$

Universal hash family requires

$Pr_{h \in H}[h(k)=h(k')] \leq \frac{1}{2}$

However these paths are not guaranteed (min weight)

(f) T F [4 points] Recall the $O(n^3 \lg n)$ matrix-multiplication algorithm to compute shortest paths, where we replaced the matrix-multiplication operator pair $(*, +)$ with $(+, \min)$. If we instead replace the operator pair with $(+, *)$, then we compute the product of the weights of all paths between each pair of vertices.

+ operator gives weight of paths between vertex pair using some intermediate vertex

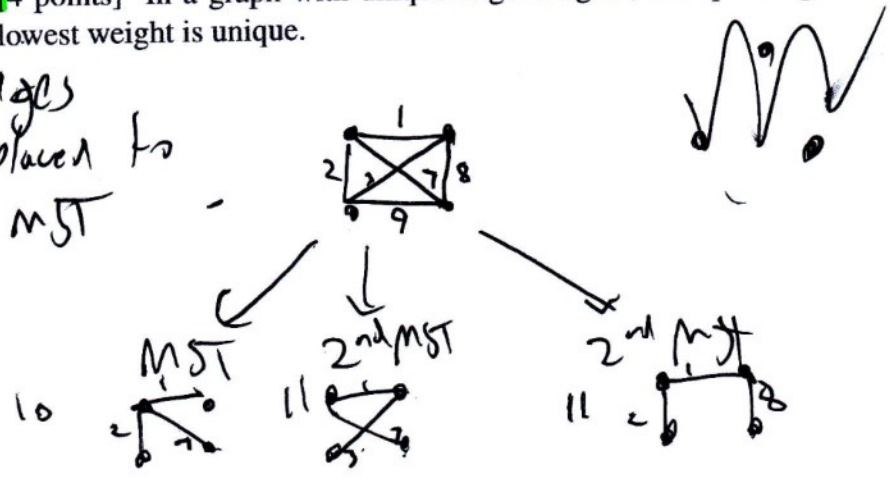
* operates on all possible intermediate vertices

(g) T F [4 points] Negating all the edge weights in a weighted undirected graph G and then finding the minimum spanning tree gives us the *maximum*-weight spanning tree of the original graph G.

tree algorithms don't have to worry about cycles,

(h) T F [4 points] In a graph with unique edge weights, the spanning tree of second-lowest weight is unique.

multiple edges can be replaced to create new MST.



(i) T **F** [4 points] In the recursion of the Floyd–Warshall algorithm:

$$d_{uv}^{(k)} = \min\{d_{uv}^{(k-1)}, d_{uk}^{(k-1)} + d_{kv}^{(k-1)}\},$$

$d_{uv}^{(k)}$ represents the length of the shortest path from vertex u to vertex v that contains at most k edges.

that contains only edges w/i the subset of edges whose indices are between 1 and k

(j) **T** F [4 points] Consider a network of processes based on an arbitrary undirected graph $G = (V, E)$ with a distinguished vertex $v_0 \in V$. The process at each vertex $v \in V$ starts with a positive integer x_v. The goal is for the process at v_0 to compute the maximum $\max_{v \in V} x_v$. There is an asynchronous distributed algorithm that solves this problem using $O(diam^2 d)$ time and $O(E + diam \cdot n)$ messages.

Every process can flood the network w/ its value

V_0 can run a convergecast query

(k) T **F** [4 points] Suppose a file server stores a hash of every file in addition to the file contents. When you download a file from the server, you also download the hash and confirm that it matches the file. This system securely verifies that the downloaded file has not been modified by an adversary, provided the hash function has collision resistance.

X

-4

CR implies it is hard to modify the file and still pass the hash check!

Generating a collision of authentic file is NOT necessary for a security breach.
Mallory can replace the file and re-hash the file!

(l) T F [4 points] Suppose Alice, Bob, and Charlie secretly generate a, b and c, respectively, and publish $g^a \bmod p$, $g^b \bmod p$, and $g^c \bmod p$, where p is a prime. Then, Alice, Bob, and Charles can each compute $g^{abc} \bmod p$ as a shared secret known only to the three of them.

No - one can calculate g^{abc} w/ the information they have

alice can calculate $g^{ab} \bmod p$ and $g^{ac} \bmod p$ w/ modular exponentiation, but can't calculate $g^{abc} \bmod p$

(m) T F [4 points] The number of memory transfers used by the best cache-oblivious algorithm is always at least the number of memory transfers used by the best external-memory algorithm for the same problem.

True. VPI: more information never hurts

(n) T F [4 points] If there is a time-optimal divide-and-conquer algorithm for a problem, then that algorithm is also optimal with respect to memory transfers in the cache-oblivious model.

Counterexample: binary search

Problem 2. Überstructure [10 points] (1 part)

Design a data structure that maintains a dynamic set S of n elements subject to the following operations and time bounds:

Operation	Effect	Time Bound	
1. INSERT(x, S)	Insert x into S.	$O(\log n)$	expected amortized
2. DELETE(x, S)	Delete x from S.	$O(\log n)$	expected amortized
3. SUCCESSOR(x, S)	Find the smallest element in S larger than x.	$O(\log n)$	worst-case
4. FIND-MIN(S)	Return the smallest element in S.	$O(1)$	worst-case
5. SEARCH(x, S)	Return TRUE if element x is in S.	$O(1)$	expected

Describe how the operations are implemented on your data structure and justify their runtime.

A binary search tree w/
1. tree-min referred to O(1) find-min
2. auxiliary dictionary for O(1) expected search
subtree size attribute at each node to support O(log n) rank + successor

Problem 3. Meancorp [15 points] (2 parts)

You are in charge of the salary database for Meancorp, which stores all employee salaries in a 2-3 tree ordered by salary. Meancorp compiles regular reports to the Department of Fairness about the salary for low-income employees in the firm. You are asked to implement a new database operation AVERAGE(x) which returns the average salary of all employees whose salary is at most x.

(a) [10 points] What extra information needs to be stored at each node? Describe how to answer an AVERAGE(x) query in $O(\lg n)$ time using this extra information.

For each child pointer, maintain

- total salary of employees w/ salary at most x in the subtree
- # of employee "

~~return root. total + valid keys in this node / root.num + salary of~~

$$\text{return } \frac{\sum_{\text{children}} \text{child.total} + \text{sum of salary of valid keys in node}}{\sum_{\text{children}} \text{child.num} + \text{num of valid keys in node}}$$

(b) [5 points] Describe how to modify INSERT to maintain this information. Briefly justify that the worst-case running time for INSERT remains $O(\lg n)$.

~~When searching for~~

insert and rebalance as normal.
Ascend to root and update meta data along each node.

Problem 4. Forgetful Forrest [15 points] (3 parts)

Prof. Forrest Gump is very forgetful, so he uses automatic calendar reminders for his appointments. For each reminder he receives for an event, he has a 50% chance of actually remembering the event (decided by an independent coin flip).

(a) [5 points] Suppose we send Forrest k reminders for each of n events. What is the expected number of appointments Forrest will remember? Give your answer in terms of k and n.

Forgets
Forrest [illegible] event i w/ probability 0.5^k. Forrest remembers w/ probability $1-0.5^k$
Let E_i be the IIRV forrest remembers event i. Let $R =$ Ex(Remembered Events) = Ex(R).
by L.E.: Ex(R) = $\sum$ Ex(E_i) = $\boxed{n(1-0.5^k)}$ EG:

(b) [5 points] Suppose we send Forrest k reminders for a *single* event. How should we set k with respect to n so that Forrest will remember the event with high probability, i.e., $1 - 1/n^\alpha$?

$1 - \frac{1}{n^\alpha} = 1 - 0.5^k$

$\frac{1}{n^\alpha} = 0.5^k$ $\Rightarrow$ $k = \log_{0.5}\left(\frac{1}{n^\alpha}\right)$

(c) [5 points] Suppose we send Forrest k reminders for each of n events. How should we set k with respect to n so that Forrest will remember *all* the events with high probability, i.e., $1 - 1/n^\alpha$?

Pr(Forrest Remembers All Events) = $\prod$ Pr(remember i'th event)
= $(1-0.5^k)^n = 1 - \frac{1}{n^\alpha}$

Let k = (a + 1) lg n
P(remember a event)
= 1 - 0.5^(lg n * a + 1)
= 1 - 1/ n^(a+1)

$\boxed{k = \log_{0.5}\left(1 - \left(1 - \frac{1}{n^\alpha}\right)^{1/n}\right)}$

-5

Problem 5. Piano Recital [15 points] (3 parts)

Prof. Chopin has a piano recital coming up, and in preparation, he wants to learn as many pieces as possible. There are m possible pieces he could learn. Each piece i takes p_i hours to learn.

Prof. Chopin has a total of T hours that he can study by himself (before getting bored). In addition, he has n piano teachers. Each teacher j will spend up to t_j hours teaching. The teachers are very strict, so they will teach Prof. Chopin only a single piece, and only if no other teacher is teaching him that piece.

Thus, to learn piece i, Prof. Chopin can either (1) learn it by himself by spending p_i of his T self-learning budget; or (2) he can choose a unique teacher j (not chosen for any other piece), learn together for $\min\{p_i, t_j\}$ hours, and if any hours remain ($p_i > t_j$), learn the rest using $p_i - t_j$ hours of his T self-learning budget. (Learning part of a piece is useless.)

(a) [6 points] Assume that Prof. Chopin decides to learn exactly k pieces. Prove that he needs to consider only the k lowest p_is and the k highest t_js.

PBC.

Suppose Chopin devises a plan using a piece outside of the k lowest, piece X.

Using cut and paste, Chopin can create an equally good plan by substituting piece X w/ a piece in the k lowest set. This is equally valid.

Some argument can be made for teachers (PBC using cut + paste

(b) [5 points] Assuming part (a), give an efficient greedy algorithm to determine whether Prof. Chopin can learn exactly k pieces. Argue its correctness.

Initialize T and P as k-maxheap of teachs and
While piece-heap is not empty k-maxheap of
In descending order of piece cost, k-lowest prices
if possible
recruit the most valuable teacher to help you
learn and spend however remaining hours needed to
learn.
If spent hours exceeds budgeted hours, return False.
otherwise True

Time:
min
$(O(n \log n)$,
$O($ n $)$
$+ k \log k)$

PoC: high value teachers used most effectively on highest value pieces. PBC for any other sln, we can swap higher value teacher to higher problem for a better sln

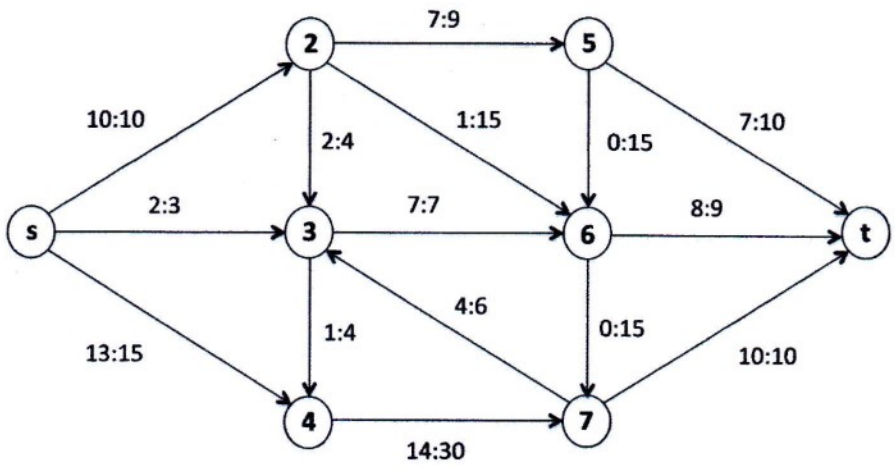
(c) [4 points] Using part (b) as a black box, give an ~~efficient algorithm that finds the~~ maximum number of pieces Prof. Chopin can learn. Analyze its running time.

Binary Search over number of pieces.

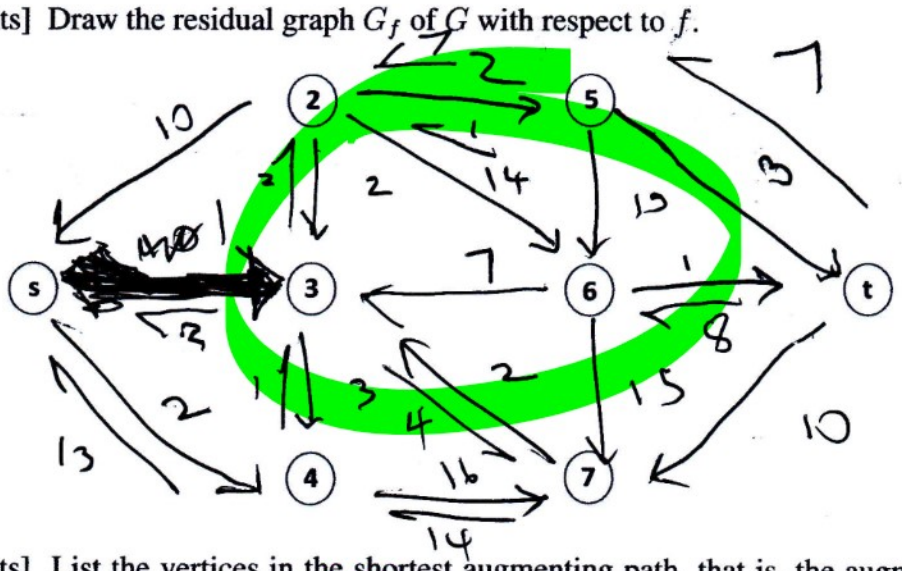
$O(\lg N \cdot T(\text{greedy Algorithm}))$

Problem 6. Be the Computer [14 points] (3 parts)

Consider the following flow network and initial flow f. We will perform one iteration of the Edmonds–Karp algorithm.



(a) [5 points] Draw the residual graph G_f of G with respect to f.



(b) [4 points] List the vertices in the shortest augmenting path, that is, the augmenting path with the fewest possible edges.

s 3 2 5 t

(c) [5 points] Perform the augmentation. What is the value of the resulting flow?

added flow is 1.

total flow is 26

n^{n-k}/k^k

Problem 7. Startups are Hard [20 points] (3 parts)

For your new startup company, *Uber for Algorithms*, you are trying to assign projects to employees. You have a set P of n projects and a set E of m employees. Each employee e can only work on one project, and each project $p \in P$ has a subset $E_p \subseteq E$ of employees that must be assigned to p to complete p. The decision problem we want to solve is whether we can assign the employees to projects such that we can complete (at least) k projects.

(a) [5 points] Give a straightforward algorithm that checks whether any subset of k projects can be completed to solve the decisional problem. Analyze its time complexity in terms of m, n, and k.

-2

Enumerate all k subsets of E, S_i.
For each S_i:
  initialize a size m bit vector to zero
  for each project in S_i:
    if employees needed is 1: continue
    else: set employees needed bits to 1
  return True
return False

$= O\left(\frac{n^k}{k!} \cdot k \cdot m\right)$

$O\left(\binom{n}{k} \cdot k \cdot m\right) = O\left(\frac{n!}{k!(n-k)!} \cdot k \cdot m\right)$

-5

(b) [5 points] Is your algorithm in part (a) fixed-parameter tractable? Briefly explain.

~~Yes. if we fix k,~~ No

then our algorithm is polynomial w.r.t. n (n^k)

(c) [10 points] Show that the problem is NP-hard via a reduction from 3D matching.

Recall the 3D matching problem: You are given three sets X, Y, Z, each of size m; a set $T \subseteq X \times Y \times Z$ of triples; and an integer k. The goal is to determine whether there is a subset $S \subseteq T$ of (at least) k disjoint triples.

A ∈ NP

A certificate can be checked by [redacted] in polynomial time for a set of k projects via employee collision checking.

We can reduce 3D matching into project assignment algorithm as follows.

E = X ∪ Y ∪ Z

P = T

If a certificate exists for project assignment on (E, P, k) then a certificate exists for 3D Matching Problem.

- P_c is a certificate for projects
- $|P_c| \geq k$
- $P_c \subseteq T$.
- P_c has no collisions

$\Rightarrow$ P_c is a certificate for 3D matching

$\Rightarrow$ Project Algorithm ∈ NP-Hard

$\Rightarrow$ ∈ NP-Complete

Problem 8. Load Balancing [15 points] (2 parts)

Suppose you need to complete n jobs, and the time it takes to complete job i is t_i. You are given m identical machines $M_1, M_2, \ldots, M_m$ to run the jobs on. Each machine can run only one job at a time, and each job must be completely run on a single machine. If you assign a set $J_j \subseteq \{1, 2, \ldots, n\}$ of jobs to machine M_j, then it will need $T_j = \sum_{i \in J_j} t_i$ time. Your goal is to partition the n jobs among the m machines to minimize $\max_i T_i$.

(a) [5 points] Describe a greedy approximation algorithm for this problem.

Initialize machines in a descending order of time heap.
Sort jobs items in descending order of costing heap
While jobs heap is not empty,
pop a job into the machine with least total time.

Runtime: $O(n(\lg n + \lg m) + m(\lg m))$

(b) [10 points] Show that your algorithm from part (a) is a 2-approximation algorithm.

Hint: Determine an ideal bound on the optimal solution OPT. Then consider the machine M_ℓ with the longest T_ℓ, and the last job i^* that was added to it.

Optimal solution is LB

$\frac{\sum t_i}{m} \leq OPT$ $\quad \frac{\sum t}{m} \leq OPT$

Before we add last job to M_ℓ it had the minimum load among all machines

which means it was at most $\frac{\sum' t}{m}$

The last added job is no longer than any job before it.

$\therefore$ at most this job costs $\frac{\sum_{j'} t}{m}$ $\quad \therefore$ $\boxed{T(M_\ell) \leq 2OPT}$

Problem 9. Distributed Coloring [20 points] (3 parts)

Consider an undirected graph $G = (V, E)$ in which every vertex has degree at most Δ. Define a new graph $G' = (V', E')$, the ***Cartesian product*** of G with a clique of size $\Delta + 1$. Specifically, V' is the set of pairs (v, i) for all vertices $v \in V$ and integers i with $0 \leq i \leq \Delta$, and E' consists of two types of edges:

1. For each edge $\{u, v\} \in E$, there is an edge between (u, i) and (v, i) in E', for all $0 \leq i \leq \Delta$. (Thus, each index i forms a copy of G.)
2. For each vertex $v \in V$, there is an edge between (v, i) and (v, j) in E', for all $i \neq j$ with $0 \leq i, j \leq \Delta$. (Thus each v forms a $(\Delta + 1)$-clique.)

Here is an example of this transformation with $\Delta = 3$:

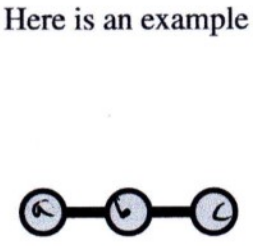


Figure 1: Graph G.

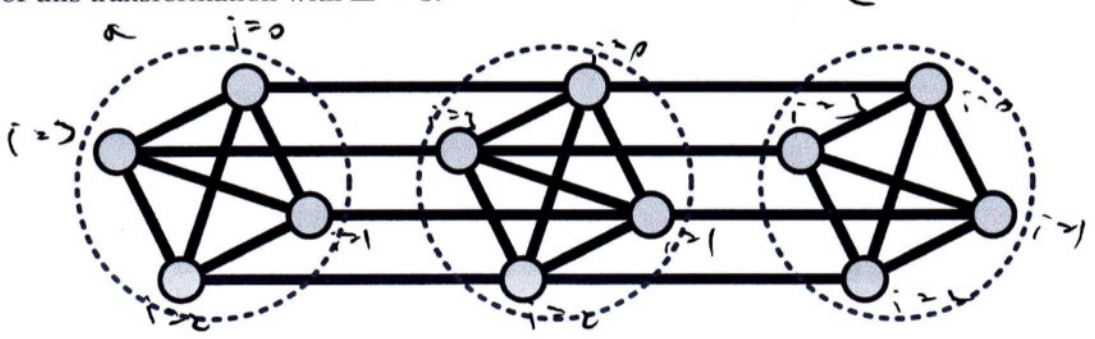


Figure 2: The Cartesian product G' of G and a clique of size 4.

(a) [8 points] Let S be any *maximal* independent set of G' (i.e., adding any other vertex to S would violate independence). Prove that, for each vertex $v \in V$, S contains exactly one of the $\Delta + 1$ vertices in V' of the form (v, i). *Hint:* Use the Pigeonhole Principle.

S contains at MOST 1 vertex (v, i) for each v

, vertices (u, i) and (v, i) are connected and cannot both be members in a MIS

$|V| \leq |S| \leq |V|$

S contains at LEAST 1 vertex (v, i) for each v

because $|\{v_0, v_1, v_2, v_3\}| = \Delta + 1$

For any v, we can have up to $|V|-1$ other nodes in MIS, not including (v, i). Up to Δ of the nodes can be excluded, leaving at least 1 valid node

(b) [8 points] Now consider a synchronous network of processes based on the graph G, where every vertex knows an upper bound Δ on the degree. Give a distributed algorithm to find a vertex $(\Delta + 1)$-coloring of G, i.e., a mapping from vertices in V to colors in $\{0, 1, \ldots, \Delta\}$ such that adjacent vertices have distinct colors. The process associated with each vertex should output its color. Argue correctness.

Hint: Combine part (a) with Luby's algorithm.

~~Use Luby's Algorithm to get every MIS.~~

While nodes are uncolored, find a MIS w/ Luby's Algorithm, give them a unique color. Set a flag that disables these nodes from participating in subsequent rounds.

If the $\Delta+1$ shares every S has every $v \in V$ $|S| = |V|$, number of MIS $= \Delta+1$

Algorithm uses $\Delta+1$ colors, only MIS share colors, which have no adjacency

(c) [4 points] Analyze the expected time and communication costs for solving the coloring problem in this way, including the cost of Luby's algorithm.

Luby's takes $|V|$ rounds

$O(\log_{\Delta}(\Delta|V|))$ w.h.p.