Paris Hom SID 862062330
Ajeet Kokatay SID 862083784

CS141 Homework 4

Problem 1: A CS student is about to take n final exams, and he has only h hours left to study. The probability that the student fails course $i(1 \leq i \leq n)$ if he spends j hours studying for it is $p_{j,i}(0 \leq j \leq h)$. Assume that the probability that failing all courses is the product of the probability of failing each individual course. Give a dynamic programming algorithm to allocate hours to courses with the objective of minimizing the probability of him failing all courses. Analyze the space- and time-complexity of your solution.

Solution 1: Let us initialize a table V as such:

y / x	0	1	2	...	h
0	1	1	1	1	1
1	1				
2	1				
3	1				
...	1				
n	1				

The values in column x represent how many course choices we have made for that row; The values in row y represent how many hours we have consumed for that column. The first row and column are 1 because you have a 100% probability of failing if you do not allocate any hours or make any exam choices.

We can recursively fill out the table as such:

$$V[x, y] = min((V[x-1, y]), (p_{x,j} * V[x-1, y-j]))$$

With the caveat that for each course we must iterate through the possible amount of hours j in $p_{x,j}$ to minimize $p_{x,j} * V[x-1, y-j]$.

Time complexity is $O(n_j * h * n)$, where n_j is the total number of $p_{i,j}$ entires.

Space complexity is $O(h * n)$

Problem 2: *Pouring water.* We have three containers whose sizes are 10 pints, 7 pints, and 4 pints, respectively. The 7-pint and 4-pint containers start out full of water, but the 10-pint container is initially empty. We are allowed one type of operation: pouring the contents of one container into another, stopping only when the source container is empty or the destination container is full. We want to know if there is a sequence of pourings that leaves exactly 2 pints in the 7- or 4-pint container.

(a) Model this as a graph problem: give a precise definition of the graph involved and state the specific question about this graph that needs to be answered.

(b) What algorithm should be applied to solve the problem?

(c) Find the answer by applying the algorithm.

Solution 2:

a) The graph involved will be the search space of the problem. The children of each node will be each operator that is performed on the current node. The question about the graph that needs to be answered is whether it is possible that we can get to a node that leaves exactly 2 pints in the 7 or 4 pint container by performing operations on the inital state (7-pint and 4 pint containers are full, but 10-pint is empty)

b) BFS should be applied to solve the problem. We should not use DFS, since we can easily fall into a loop and will never be able to get out of it using DFS, for example if we accidentally keep performing an operation to get back to the parent state and then back, like pour content of 4pint into 7pint, then perform pour content of 7pint into 4pint. We will perform BFS and expand each node by doing every operation that we are allowed on that node, so this means that possible children are: pouring all content of 7-pint into 10-pint or 2-pint, pouring all content of 2-pint into 10-pint or 7-pint, pouring 10-pint content into 2-pint or 7-pint (until source container is empty or the destination container is full). Then, we will check each state of the node and see if it matches the state we are looking for, that is the state in which there are exactly 2 pints in the 7 or 4 pint container. Then, if we want, when we found this goal state, we can go back up the graph from the goal vertex back to the root to find the sequence of pourings, but we would need to have a pointer to the parent node in each child if we wanted to do this..

c) the answer will be in the end, the containers will be: 4pint container holds 2 pints, 7pint container hold 7 pints, and the 10pint container hold 2 pints. To get to this, we will perform the following operations/sequence of actions. We will denote the container and the amount of each with: x/4 means 4pint container hold x pints, y/7 means 7pint container hold y pints, and z/10 means 10pint container hold z pints:

1. pour contents of 4pint into 10pint: 0/4, 7/7, 4/10
2. pour contents of 7pint into 10pint: 0/4, 1/7, 10/10
3. pour contents of 7pint into 4pint: 1/4, 0/7, 10/10
4. pour contents of 10pint into 4pint: 4/4, 0/7, 7/10
5. pour contents of 4pint into 7pint: 0/4, 4/7, 7/10
6. pour contents of 7pint into 10pint: 0/4, 1/7, 10/10
7. pour contents of 10pint into 4pint: 4/4, 1/7, 6/10
8. pour contents of 4pint into 7pint: 0/4, 5/7, 6/10
9. pour contents of 10pint into 4pint: 4/4, 5/7, 2/10
10. pour contents of 4pint into 7pint: 2/4, 7/7, 2/10

Problem 3: Two paths in a graph are called edge-disjoint if they have no edges in common. Show that in any undirected graph, it is possible to pair up the vertices of odd degree and find paths between each such pair so that all these paths are edge-disjoint.

Solution 3: We will prove using induction that in any undirected graph, it is possible to pair up the vertices of odd degree and find paths between each such pair so that all these paths are edge-disjoint.

Base Case: We have a undirected graph G with 2 vertices v_1 and v_2 that are connected, then each vertex has a degree of 1, which is odd. Let us find a path between vertices v_1 and v_2 that is disjoint. Then, it is obvious that since G only has vertices v_1 and v_2, and they are connected then there is only one path that we can use to get from v_1 to v_2, which is the edge connecting v_1 and v_2. This one path is edge disjoint, since there is no other edge we can use to create a path from v_1 to v_2.

Inductive hypothesis: Suppose that we have a undirected graph G with $n-1$ vertices. Suppose that graph G has vertices of odd degrees. Then, it is possible to pair up these vertices of odd degrees and find paths between each such pair so that all these paths are edge-disjoint.

Inductive step: We will show that undirected graph G with n vertices, which has vertices of odd degrees, it is possible to pair up the vertices of odd degrees and find paths between each such pair that all the paths are edge disjoint.

Then, suppose that graph G is the set of vertices $V = \{v_1, v_2, v_3, v_4, v_5, ..., v_n\}$. We will remove v_n and call this graph G'. Then, graph G' has $n-1$ vertices. We said that in a graph with $n-1$ vertices, we are able to pair up vertices of odd degrees and find paths between each such pair so that all these paths are edge disjoint from our inductive hypothesis. So, Then, we will add back vertex v_n. Thus, all the paths between vertices of odd degrees are edge-disjoint. Then, add back in our vertex v_n back into G' to recreate graph G which then has n vertices. Then, we just need to find the vertices that v_n shares an edge with to those vertices with edge disjoint paths found from G', which will then create edge disjoint paths between the odd vertices in G to v_n, since the edge connecting v_n to a vertex v_i that v_n shares an edge to will be that edge between v_n to v_i, which is disjoint demonstrated by the base case. Thus proves that in any undirected graph, it is possible to pair up the vertices of odd degree and find paths between each such pair so that all these paths are edge-disjoint..

Problem 4: If there is ever a decision between multiple neighbor nodes in the BFS or DFS algorithms, assume we always choose the letter in alphabetical order.

a) Consider the following adjacency matrix.

	A	B	C	D	E	F	G	H	I
A	0	1	1	1	0	0	0	0	0
B	1	0	1	0	1	0	1	0	0
C	1	1	0	1	0	0	0	0	1
D	1	0	1	0	0	0	0	0	0
E	0	1	0	0	0	0	0	1	1
F	0	0	0	0	0	0	1	1	0
G	0	1	0	0	0	1	0	1	0
H	0	0	0	0	1	1	1	0	1
I	0	0	1	0	1	0	0	1	0

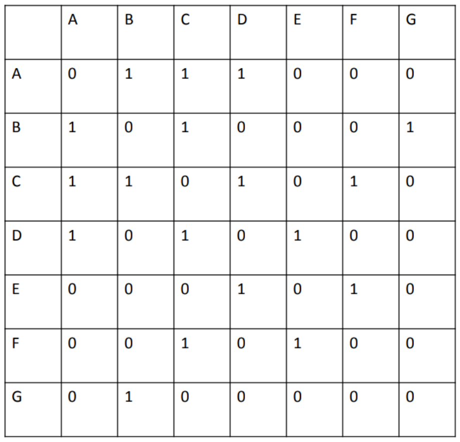
a.1) Draw the graph that is represented by the matrix.

a.2) In what order will the nodes be visited using Breadth First Search? In what order will the nodes be visited using Depth First Search?

b)If you have the 0/1 adjacency matrix of a graph, and you take the matrix to the N^{th} power, then the $(i,j)^{th}$ entry of the result tells how many paths of length N there are from vertex i to vertex j (here the length is measured in number of edges traversed). For example, in a) according to the adjacency matrix there is a path of length 1 edge between vertices A and B.

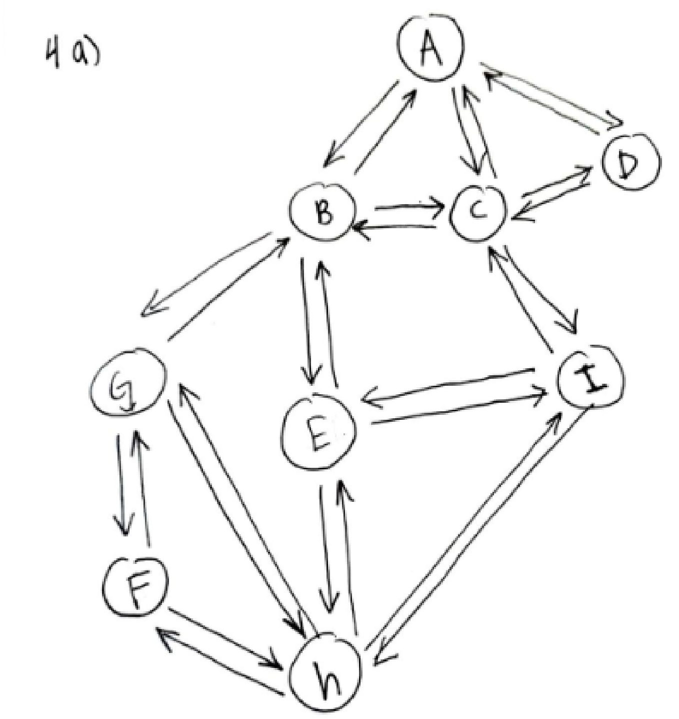
Let $G = (V, E)$ be an undirected graph. A triangle in G is a cycle consisting of exactly three vertices (or, equivalently, three edges). Suppose that G is represented as an adjacency matrix. Give an algorithm to determine whether G contains any triangle in $O(n^{\log_2 7})$ worst-case time.

c) Using matrix multiplications, find all the triangles in the graph, represented by the following adjacency matrix.



	A	B	C	D	E	F	G
A	0	1	1	1	0	0	0
B	1	0	1	0	0	0	1
C	1	1	0	1	0	1	0
D	1	0	1	0	1	0	0
E	0	0	0	1	0	1	0
F	0	0	1	0	1	0	0
G	0	1	0	0	0	0	0

Solution 4:
4a1) The graph that is represented by the matrix is:



4a2)
For DFS (break ties between nodes by going with the one alphabetically smaller): A → B → C → D → I → E → H → F → G
For BFS (break ties between nodes by going with the one alphabetically smaller): A → B → C → D → G → E → I → F → H
4b) Let M be our adjacency matrix. The matrix M^3 tells how how many walks of length 3 there are between two vertices. The algorithm is to simply look for vertices that have at least one third degree connection with themselves. This means we return true if we find diagonal matrix cells that are of value ≥ 1 in the matrix M^3. Since we can use Strassen's algorithm to multiply matrices in $O(n^{log_2(7)})$, the total algorithm runtime is

$(2 * O(n^{log_2(7)}) + O(n)$, since we use Strassens' algorithm twice, then iterate through the third degree matrix in n time where n is the number of vertices. $O(n^{log_2(7)})$ eclipses the term $O(n^2)$, so this algorithm runs at $O(n^{log_2(7)})$ time.

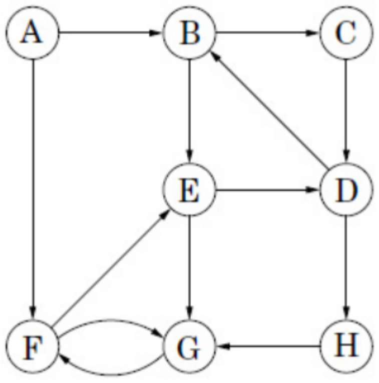
4c)

$$A^3 = \begin{pmatrix} 4 & 6 & 6 & 6 & 2 & 3 & 1 \\ 6 & 2 & 7 & 2 & 3 & 1 & 3 \\ 6 & 7 & 4 & 8 & 1 & 6 & 1 \\ 6 & 2 & 8 & 2 & 5 & 1 & 2 \\ 2 & 3 & 1 & 5 & 0 & 4 & 0 \\ 3 & 1 & 6 & 1 & 4 & 0 & 1 \\ 1 & 3 & 1 & 2 & 0 & 1 & 0 \end{pmatrix}$$

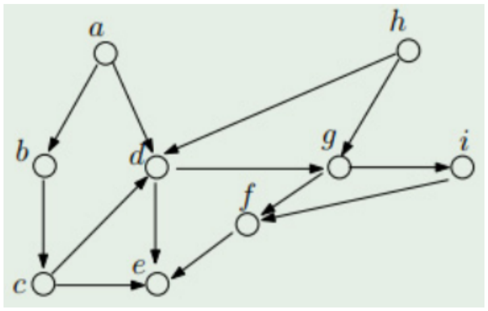
Thus, the the triangles in the graph are: A, B, C and A, C, D, since the diagonals of the matrix are ≥ 1 on the cells for A, B, C, D

Problem 5:

a) Perform depth-first search on the following graph. Classify each edge as a tree edge, forward edge, back edge, or cross edge, and give the pre and post number of each vertex.

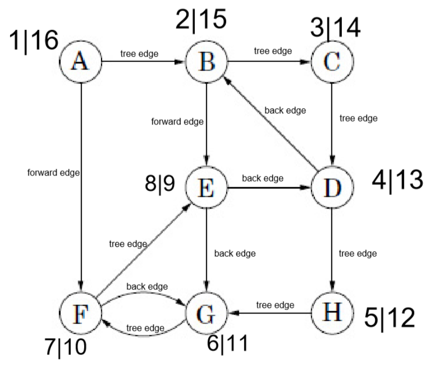


b) Run the topological ordering algorithm on the following directed graph G. Give the post order of every vertex. Show a topological ordering of the graph.



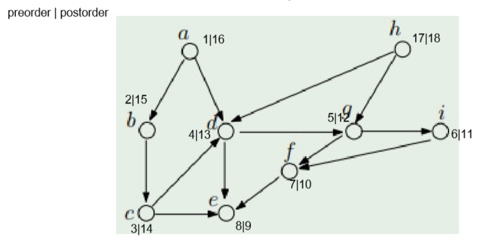
Solution 5:

a) We will perform DFS search on the graph, with the pre and post number of each vertex denoted:

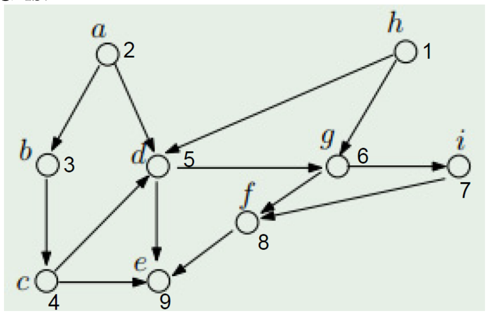


preorder — postorder above the vertex.

b) we will now run the topological ordering on G, and give the preorder and postorder of every vertex above the vertex, denoted by preorder—postorder.



Now we can give a topological ordering of G using the postorder of each vertex. A topological ordering of G is:



As we can see, a topological ordering of G can be: h → a → b → c → d → g → i → f → e