CS 161 W19: Recitation 4 Solutions

February 2019

Exercise 0

Recall from class that a universal hash family H is a family of functions from a universe U to $[1, \ldots, n]$ that satisfies

$$P_{h \in H}\left(h(u_i) = h(u_j)\right) \leq \frac{1}{n} \qquad \forall u_i, u_j \in U,\ u_i \neq u_j$$

where n is the number of buckets. Which of the following are universal hash families?

(a) Pick a prime $p \geq |U|$. $H = \{(ax + b \bmod p) \bmod n \mid a \in \{1, \ldots, p-1\}, b \in \{0, \ldots, p-1\}\}$.

(b) $H = \{x \bmod n\}$ (recall n is the number of buckets and is fixed).

(c) $H = \{ax \bmod n \mid a \in \{1, \ldots, n-1\}\}$.

(d) $H = \{ax + b \bmod n \mid a \in \{1, \ldots, n-1\}, b \in \{0, \ldots, n-1\}\}$.

Solution 0

(a) This is a universal hash family—it's the one we saw in class.

(b) This is not a universal hash family. H consists of only 1 function, and for instance $u_i = 1, u_j = n+1$ are mapped to the same bucket with probability 1.

(c) This is not a universal hash family. Let $u_i = 1, u_j = n + 1$. Since $a \equiv an + a \equiv a(n + 1) \mod n$, these are mapped to the same bucket with probability 1.

(d) This is not a universal hash family. The parameter b has no effect on whether two elements are mapped to the same bucket, it only permutes the buckets. I.e., $au_i + b \equiv au_j + b \mod n$ if and only if $au_i \equiv au_j \mod n$. Thus the same argument as in part (c) applies.

Exercise 1

In section 2, we came up with a few different algorithms that take in n pairs of integers (x_i, y_i)—where for all i, j we have $x_i \neq x_j$ and $y_i \neq y_j$—and find the maximum cardinality subset of collinear points. Two points uniquely define a line, $y = mx + b$, with slope m and intercept b, and a set of points S is *collinear* if they all fall on the same line; that is, for all $(x_i, y_i) \in S$, $y_i = mx_i + b$.

In section 2, we asked you not to use hashing and you were able to arrive at a $O(n^2 \log n)$ time solution. Now, using hashing, come up with an $O(n^2)$ solution. Assume that given two points, you can compute the corresponding m and b for the line passing through them in constant time, and you can compare two slopes or two intercepts in constant time.

Solution 1

Create a hash table with n^2 buckets. For each pair of points, compute the corresponding (m, b) tuple and hash it into the hash table. If that tuple is not yet present in the corresponding bucket, add it with a counter initialized to 1. Otherwise, find the existing tuple and increment its counter. Finally, iterate over the buckets to find the largest counter.

Exercise 2

Consider a hash table with n buckets and n elements. We assume that each element has a distinct key and they are hashed using a uniformly random hash function, i.e., each element has equal probability of hashing into any of the n buckets independently of other elements.

(a) For a particular bucket, what is the probability that the bucket has exactly one element? How about k elements?

(b) Let X_i be a random variable for the number of elements in bucket i. What is the expected value of X_i?

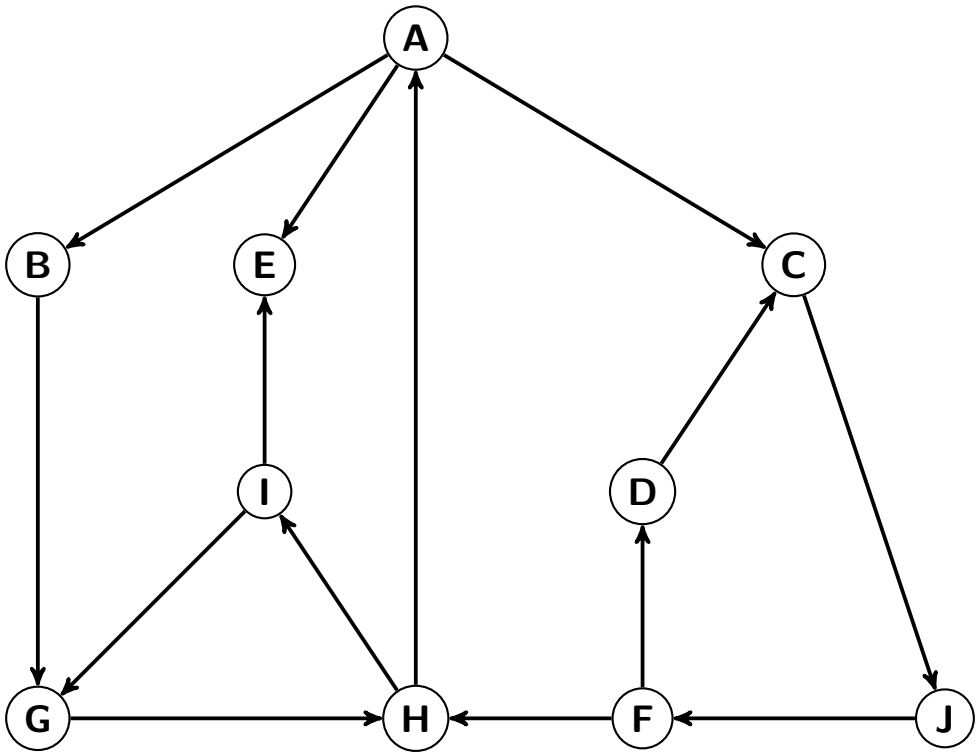
Solution 2

(a) Consider bucket i. Each element is hashed into bucket i with probability $\frac{1}{n}$ independently of other elements. Let X_i be a random variable that denotes the number of elements hashed into bucket i. Then, X_i is a binomial random variable with parameters n and $\frac{1}{n}$. Hence,

$$P(X_i = k) = \binom{n}{k} \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k}$$

(b) Since $X_i \sim B\left(n, \frac{1}{n}\right)$, $E[X_i] = n \cdot \frac{1}{n} = 1$.

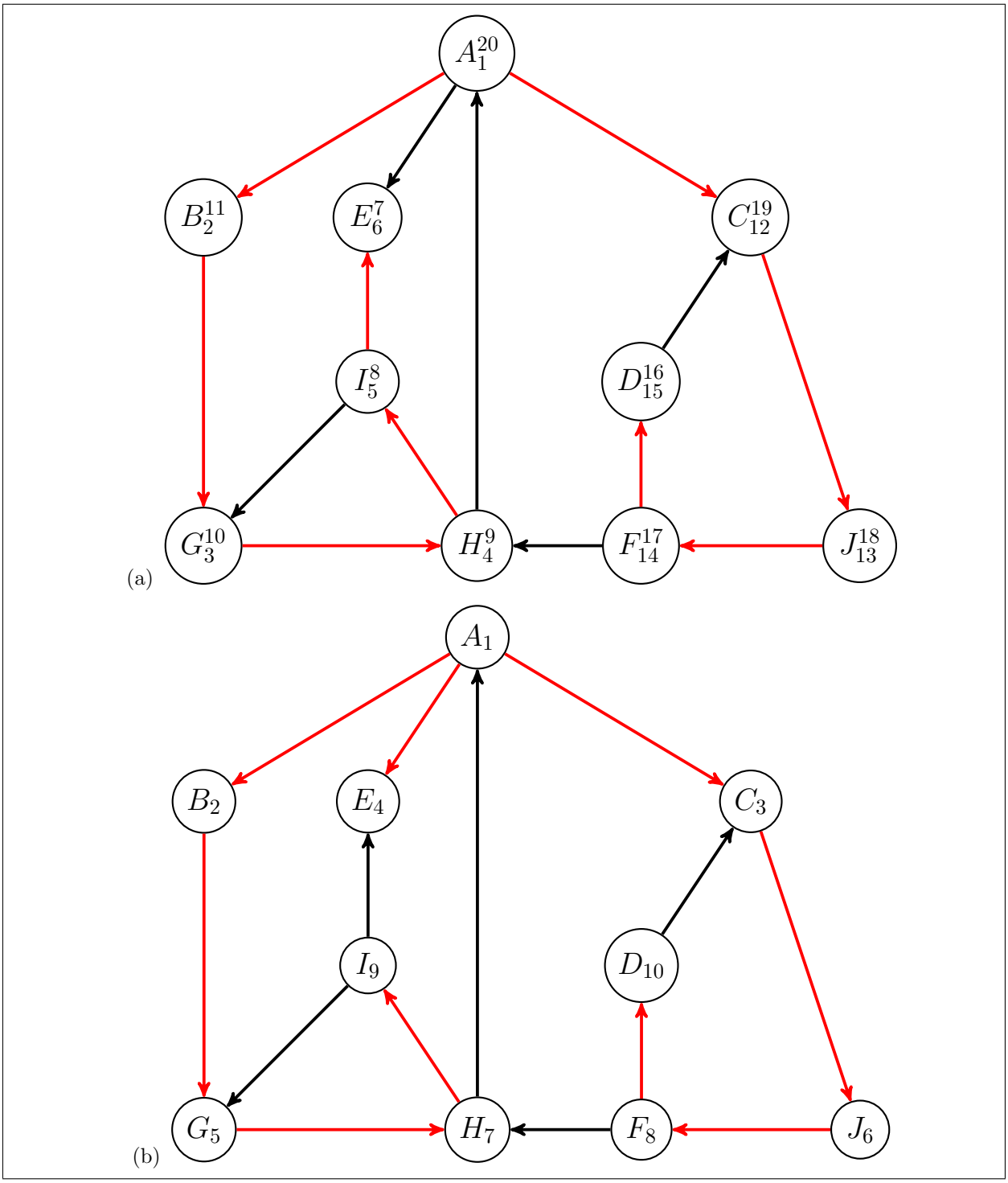
Exercise 3



(a) Perform DFS on the graph above starting from vertex A. Use lexicographical ordering to break vertex ties. As you go, label each node with the start time and the finish time. Highlight the edges in the tree generated from the search.

(b) Perform BFS on the graph above starting from vertex A. Use lexicographical ordering to break vertex ties. As you go, label each node with the discovery order. Highlight the edges in the tree generated from the search.

Solution 3


A_1^{20}
B_2^{11}
E_6^7
C_{12}^{19}
I_5^8
D_{15}^{16}
G_3^{10}
H_4^9
F_{14}^{17}
J_{13}^{18}
(a)
A_1
B_2
E_4
C_3
I_9
D_{10}
G_5
H_7
F_8
J_6
(b)

Exercise 4

A bipartite graph is an undirected graph whose vertices can be divided into two independent sets, U and V, such that every edge (u, v) connects one vertex in U and one vertex in V (or vice versa). Equivalently, a graph is bipartite if and only if the vertices can be colored using two colors such that no edge connects two vertices of the same color.

(a) Design an algorithm using BFS to determine whether or not a graph is bipartite.

(b) Design an algorithm using DFS to determine whether or not a graph is bipartite.

Solution 4

(a) Assuming the graph is connected, we simply perform a search, coloring vertices as we visit them and checking each edge as we explore it. Except for the first vertex, which we color RED without loss of generality, when we encounter a vertex there will be only one possible way to color it while satisfying bipartite-ness. Since search checks every edge, the graph is bipartite if and only if this search-based coloring succeeds.

Algorithm 1: isBipartiteBFS(G)

```
set L_i = [] for i ∈ [1, n]
pick a starting node s
L_0 = [s]
mark s as visited and color it RED
for i ∈ [0, n − 1] do
    next color = BLUE if i is even, else RED
    for u ∈ L_i do
        for each neighbor v of u do
            if v hasn't been visited then
                mark v as visited and color it next color
                add v to L_{i+1}
            else
                check v is colored next color, otherwise return False
return True
```

(b) This algorithm is essentially the same except using DFS. Call isBipartiteDFS(G, s, RED) on any starting vertex s.

Algorithm 2: isBipartiteDFS(G, s, current color)

```
mark s as visited and color it current color
for each neighbor v of s do
    if v hasn't been visited then
        run isBipartiteDFS(G, v, opposite of current color)
        if the recursive call is False, return False
    else
        check v is colored the opposite of current color, otherwise return False
return True
```

Exercise 5

Suppose you have an undirected, connected graph $G = (V, E)$ and two nodes $s, t \in V$. You then take a *random walk* on G: start at node $v_0 = s$, and at every step v_i, pick an edge coming out of v_i uniformly at random, and set v_{i+1} to be the other node on that edge.

Prove that with probability 1, your random walk will hit t: there exists some i such that $v_i = t$.

Solution 5

Pick any path from s to t. The length of that path is at most n (since there are no repeated vertices in a path), so the probability that the walk takes *exactly* those steps at the start is at least $1/n^n$. This worst-case scenario is based on the possibility that all nodes are connected to all other nodes in the graph.

If the walk of length $\leq n$ didn't take exactly that path, start where it ended up, and repeat the same argument: with probability at least $1/n^n$, we'll take some exact path that leads to t.

So we divide time into epochs of length at most n, where in each epoch we reach t with probability at least $1/n^n$.

The probability that we never reach t in any of these epochs is $\prod_{i=1}^{\infty}(1 - 1/n^n) = 0$.