CSE 101: Homework #1

Due on Apr 10, 2024 at 23:59pm

Professor Jones

Ray Tsai

A16848188

Problem 1

Let T be defined by the recurrence relation:

$$T(0) = 1 \;, T(1) = 4 \qquad T(n) = T(n-1) + 2T(n-2) + (3)(2^{n-1}) \;\; for \;\; all \;\; n \geq 2$$

(a) Prove that $T(n) = \Omega(2^n)$ using induction.

Proof. Pick $n_0 = 0$. We show that $T(n) \geq 2^n$ for all $n \geq n_0$ by induction on n. The base cases are trivial, as $T(0) = 1 \geq 2^0$ and $T(1) = 4 \geq 2^1$. Suppose $n \geq 2$. By induction,

$$\begin{aligned} T(n) &= T(n-1) + 2T(n-2) + 3 \cdot 2^{n-1} \\ &\geq 2^{n-1} + 2 \cdot 2^{n-2} + 3 \cdot 2^{n-1} = \frac{5}{2} \cdot 2^n > 2^n, \end{aligned}$$

and we are done. $\square$

(b) Prove that $T(n) = O(n2^n)$ using induction.

Proof. Pick $c = 2$ and $n_0 = 1$. We show that $T(n) \leq cn2^n$ for $n \geq n_0$ by induction on n. Since $T(1) = 4 \leq c \cdot 2 = 4$ and $T(2) = 12 \leq c \cdot 2 \cdot 2^2 = 16$, the base case is done. Suppose $n > 2$. By induction,

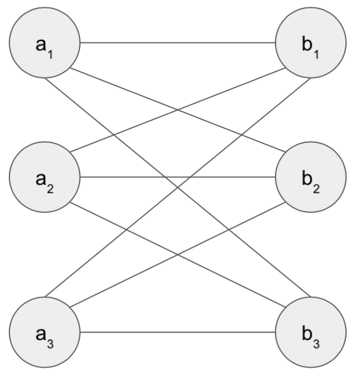
$$\begin{aligned} T(n) &= T(n-1) + 2T(n-2) + 3 \cdot 2^{n-1} \\ &\leq c(n-1)2^{n-1} + 2c(n-2)2^{n-2} + 3 \cdot 2^{n-1} \\ &= (4n-3)2^{n-1} = 2\left(n - \frac{3}{4}\right)2^n \leq c \cdot n2^n, \end{aligned}$$

and we are done. $\square$

Problem 2

Let $B(n)$ be the nth *Complete Balanced Bipartite Graph* on $2n$ vertices. $B(n)$ has $2n$ vertices n on each side. One side has vertices labeled $a_1, \ldots, a_n$ and the other side has vertices labeled $b_1, \ldots, b_n$. There is an edge connecting a_i and b_j for all $1 \le i, j \le n$

Below is the graph of $B(3)$:



(a) How many edges does $B(n)$ have?

Proof. Since each vertex is of degree n and there are $2n$ vertices, $e(B(n)) = n^2$ by the Handshake Lemma. □

(b) Let $H(n)$ be the number of Hamiltonian paths of $B(n)$ that start from an a_i vertex and ends at a b_j vertex (Hamiltonian paths are paths that go through each vertex exactly once.) Prove that $H(n) = (n!)^2$.

Proof. Since $B(n)$ is a complete balanced bipartite graph, we may go from any a_k to any desired b_l, and vice versa. Hence, counting the number of Hamiltonian paths in $B(n)$ is equivalent to counting the orderings of all vertices, where vertices of the same part are adjacent and the starting vertex being some a_i. Since there the vertices of each part have $n!$ orderings, $H(n) = (n!)^2$. □

(c) Let $P(N)$ be the number of Hamiltonian paths of a Complete Balanced Bipartite Graph on N vertices. Determine the big-Theta bound of $P(N)$.

Proof. We have already know $P(N) = (\frac{N}{2}!)^2$ when N is even. Applying Stiring's formula, we get

$$P(N) = \left(\frac{N}{2}!\right)^2 \sim \left(\sqrt{\pi N}\left(\frac{N}{2e}\right)^{N-1/2}\right)^2 = 2^{\log \pi + \log N + N\log N - N\log 2e}$$

Hence, $P(N) = 2^{\Theta(N \log N)} = \Theta(N!)$. □

Problem 3

A *triangle* in an undirected, simple graph is a set of three distinct vertices x, y, z such that all pairs are connected by an edge.

(a) Consider the following algorithm that takes as input an adjacency matrix of a simple undirected graph G and returns True if there exists a triangle and returns False if there is not a triangle.

Triangle1(G) (G, an undirected simple graph with n vertices in adjacency matrix form.)

```
for i = 1, ..., n:
    for j = 1, ..., n:
        if G[i, j] == 1 :
            for k = 1, ..., n:
                if G[i, k] == 1 and G[j, k] == 1:
                    return True
return False
```

Show that the runtime for this algorithm is $O(|V|^2 + |V||E|)$.

Proof. The algorithm finds edges by iterating through pairs of vertices, which takes $O(|V|^2)$ time. Upon finding an edge, it proceeds to iterating through vertices to look for the potential vertex that completes the triangle, which takes additional $O(|V|)$ time per edge. Hence, the runtime for *Triangle1* is $O(|V|^2 + |V||E|)$. □

(b) In order for *Triangle1* to return True, what needs to happen and why does this correspond to a triangle?

Proof. *Triangle1* returns true only if it finds an edge between some vertices u, v and there exists another vertex w which is adjacent to both u and v. In this case, since u, v, w are pair-wise adjacent, they form a triangle. □

(c) Consider the following algorithm that takes as input an adjacency matrix of a simple undirected graph G and returns True if there exists a triangle and returns False if there is not a triangle.

Triangle2(G) (G, an undirected simple graph with n vertices in adjacency matrix form.)

```
Compute H = G × G.
for i = 1, ..., n:
    for j = 1, ..., n:
        if G[i, j] == 1 AND H[i, j] > 0:
            return True
return False
```

Assuming that matrix multiplication between two $n \times n$ matrices takes $O(n^{2.81})$ time, calculate the runtime of this algorithm.

Proof. Iterating through pairs of vertices take $O(|V|^2) = O(n^2)$ time. Together with the runtime for matrix multiplication, the runtime for this algorithm is $O(n^{2.81} + n^2) = O(n^{2.81})$. □

Problem 3 continued on next page...

(d) In order for *Triangle2* to return True, what needs to happen and why does this correspond to a triangle? (In particular, what does it mean for $H[i, j] = 1$ or $H[i, j] = 2$?)

Proof. *Triangle2* returns True only when $H[i, j] > 0$ and $G[i, j] == 1$. Note that $H[i, j]$ records the number of length 2 paths from vertex i to vertex j. Hence, $H[i, j] > 0$ and $G[i, j] == 1$ indicates that i, j are both adjacent to some vertex k and i, j are also adjacent to each other, which makes i, j, k a triangle. □

(e) Is *Triangle1* or *Triangle2* more efficient? (Justify your answer.) (Hint: think about dense and sparse graphs.)

Proof. In dense graphs, $|E|$ is close to n^2, which makes the runtime for *Triangle1* around $O(n^3)$. But in sparse graphs, $|E|$ far less than n^2, which makes the runtime for *Triangle1* $O(n^2)$ in this case. Hence, *Triangle1* is more efficient than *Triangle2* when G is sparse, and the other way around when G is dense. □

Problem 4

Given a directed graph G with vertex weights $w_v \in \{0, 1, 2\}$ (in other words, each vertex is either labeled with 0, 1 or 2), and vertices s and t. Determine if there is a path in G from s to t such that the ternary sequence of vertex weights in the path does not repeat the same number twice in a row.

Consider the following algorithm that claims to solve this problem:

> **Algorithm Description:**
>
> Input: a $\{0, 1, 2\}$-labeled directed graph G, a vertex s of G and a vertex t of G.
>
> Create a graph G' by removing all edges (u, v) such that $w(u) == w(v)$.
>
> Run graphsearch on G' starting from s. If t is visited then return TRUE. Otherwise return FALSE

Prove that this algorithm is correct.

Proof. Suppose there exists a path P from s to t without repeating consecutive numbers, say $v_1 v_2 \ldots v_n$, where $v_1 = s$ and $v_n = t$. Since $w(v_i) \neq w(v_{i+1})$ for all i, all edges of P remains in G', and thus $P \subseteq G'$. It follows that t can be visited from s with graphsearch on G' via P, so the algorithm returns TRUE.

Suppose not. We may assume there exists a path P from s to t in G, otherwise t is also not reachable from s in $G' \subseteq G$ and we are done. Then, P must include some edge (v_i, v_{i+1}) such that $w(v_i) = w(v_{i+1})$. But then $(v_i, v_{i+1}) \notin E(G)$, so any paths from s to t in G do not remain in G', and thus the algorithm return FALSE.

Therefore, the algorithm is correct. $\square$