Beckham Carver & Ben Wilkin

31 March 2023

COSC 3020

Dr. Kotthoff

Assignment 2: Algorithm Analysis and Graphs

1 Theory vs. Practice (9 points)

(a) List 3 reasons why asymptotic analysis may be misleading with respect to actual performance in practice. (3 points)

Big-O describes the growth of an algorithm at infinite input sizes, i.e. an algorithm that takes exactly 1 million steps per n is equivalent to one that takes 1 step per n. That is to say, 1 000 000n = O(n) and O(n) < O(n^2). So if an algorithm A takes n^2 steps per n, and B takes 1 million per, their asymptotic analysis says that O(A) < O(B) even though for all inputs less than 1,000,000 algorithm B is faster.

The speed of memory operations is not easily differentiated in asymptotic analysis. The same algorithm could have implementation A which uses a contiguous list in memory, while implementation B could be implemented as a linked-list split which can be split across memory. They both can have the same time complexity and memory complexity, though the performance of implementation A is likely able to be a lot faster.

Depending on the programming language, implementations of an algorithm can noticeably differ in performance between their recursive and iterative implementations. A recursive algorithm may take longer than predicted by asymptotic analysis as unnecessary stack frames are loaded and heap operations performed.

(b) Suppose finding a particular element in a binary search tree with 1,000 elements takes 5 seconds. Given what you know about the asymptotic complexity of search in a binary search tree, how long would you guess finding the same element in a search tree with 10,000 elements takes? Explain your reasoning. (3 points)

Finding an element in a binary search tree should take at worst O(H) time where H = $\log_2(n)$ if the tree is well spread out. Assuming the data was entered randomly, and the tree is well spread, searching a BST of 1,000 elements takes $\log_2(1{,}000) = 3$ operations. Therefore, each operation took 1.66 seconds if we assume startup and I/O operations took zero time.

Assuming again that startup and I/O operations took zero time, we should expect searching 10,000 elements to take $\log_2(10{,}000) = 4$ operations and therefore 6.66 seconds.

(c) You measure the time with 10,000 elements and it takes 100 seconds! List 3 reasons why this could be the case, given that reasoning with the asymptotic complexity suggests a different time. (3 points)

The data stored in the BST was most likely added to the BST in a sorted order. Meaning that the tree is a skewed and in the worst case, is simply a linked list. Searching then takes O(n) time as each element is in

order and must be traversed one at a time. This tracks with the time observed, because when the input increased 10x the time taken increased 20x which fits with a O(n) time complexity.

The implementation of the search could also be awful, simply checking every element in a breadth first search fashion, which takes O(|V|) time where |V| = n for both spread and skewed graphs.

The implementation of search could alternatively be a tree traversal, following a left-most or right-most path until it finds the value, this again takes O(n) which was the growth observed.

2 Graph Properties (9 points)

(a) Prove that if two graphs A and B do not have the same number of nodes, they cannot be isomorphic. (3 points)

Proof by contradiction

Definition of isomorphism:
G1 = (V1, E1) is isomorphic to G2 = (V2, E2) if there is a one-to-one and onto function (bijection) f : V1 → V2 such that (u, v) ∈ E1 iff (f(u), f(v)) ∈ E2.

Graphs A and B do not have the same number of nodes:

A = (V, E), |V| = n
B = (X, Y), |X| = k
where k != n

If A and B are isomorphic, and n < k:

f() is a one-to-one function and all v ∈ V have been mapped by f() to different x ∈ X
Therefore f() cannot be an 'onto' function because there exists an x ∈ X that has not mapped to and there are no more v ∈ V to map without having repeats.

If A and B are isomorphic, and n > k:

f() is an onto function and all x ∈ X have been mapped onto by different v ∈ V from f()
Therefore f() cannot be a 'one-to-one' function because exists a v ∈ V that has not mapped to any x ∈ X and doing so requires repeat mappings.

Contradiction:
f() cannot be both one-to-one and onto, a bijection f() does not exist between any graph A and B that do not have the same number of nodes, meaning that they cannot be isomorphic.

(b) Prove that if two graphs A and B have the same number of nodes and are completely connected, they must be isomorphic. (3 points)

Proof by example

Definition of isomorphism:
G1 = (V1, E1) is isomorphic to G2 = (V2, E2) if there is a one-to-one and onto function (bijection) $f : V1 \rightarrow V2$ such that $(u, v) \in E1$ iff $(f(u), f(v)) \in E2$.

Graphs A and B are completely connected and have the same number of nodes:

A = (V1, E1) where $(u1,v1) \in E1$ iff every u is connected to every v1, and $|V1| = n$
B = (V2, E2) where $(u2,v2) \in E2$ iff every u is connected to every v2, and $|V2| = n$

Function f:
Create a function f() that never returns the same result twice, and takes a $v1 \in V1$ and maps it to an arbitrary name from $v2 \in V2$ until all names have been used in which case the function maps v1 to nothing. Apply this function to all $v1 \in V1$. Because $|V1|=|V2|$ and that f() cannot repeat a mapping, f() is both one-to-one and onto.

Because pairs '$(u1,v1) \in E1$ iff every u is connected to every v1' is true before renaming, and after renaming all $v1 \in V1$ to different $v2 \in V2$ via our bijective function f().
'$(f(u1),f(v1)) \in E2$ iff every u is connected to every v2' also holds.

Therefore, any two graphs that are fully connected, and have the same number of nodes, a bijective function f() from $V1 \rightarrow V2$ exists.

(c) Prove that if two graphs A and B are isomorphic they do not have to be completely connected. (3 points) You need to give complete, formal proofs – state all your assumptions and make sure that you've explained every step of your reasoning. Hint: A good way to start is by writing down the definitions for everything related to what you want to prove.

Proof by example

Definition of isomorphism:
G1 = (V1, E1) is isomorphic to G2 = (V2, E2) if there is a one-to-one and onto function (bijection) f : V1 → V2 such that (u, v) ∈ E1 iff (f(u), f(v)) ∈ E2.

Graph A and B:

A = (V1, E1), V1 = {a, b, c}, E1 = {(a,b), (b,c)}, A is not a completely connected graph.
B = (V2, E2), V2 = {x, y, z}, E2 = {(x,y}, (y,z)}, B is not a completely connected graph.

Function f:

a -> x
b -> y
c -> z

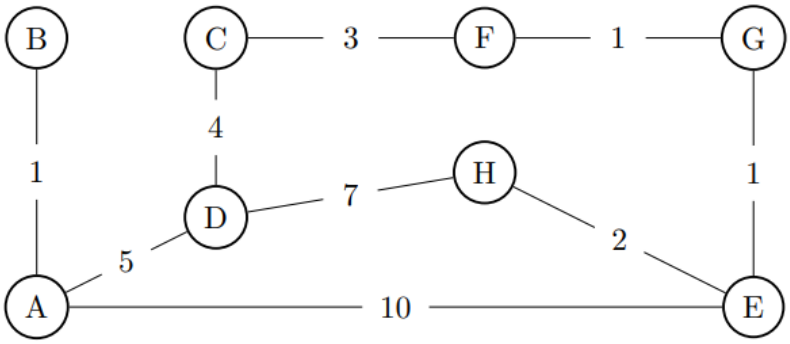
Apply function f to all V1:

V1 = {a, b, c}
V1* = {f(a), f(b), f(c)}
V1* = {x, y, z}
V1* = V2
…
E1* = E2

Therefore (u, v) ∈ E1 iff (f(u), f(v)) ∈ E2 is true.

Because A and B are not completely connected, and they are isomorphic, there exists at least one pair of graphs that are isomorphic but are not completely connected. Therefore graphs do not need to be completely connected to be isomorphic.

4 Kruskal's Algorithm (5 points)



Run Kruskal's algorithm on the above graph to produce a minimum spanning tree (MST). The weights are the numbers drawn on top of each edge. Draw the minimum spanning tree Kruskal's algorithm finds, indicate the order in which edges are added to it, and note the total weight of the MST. You may break ties arbitrarily. Provide the output, not code that produces the output.

The red numbers are the order of edges added. The orange edges won't be added because they will form a cycle. The total weight of the minimum spanning tree is 17. Below is the final tree.