**CS2040 Tips**

Disclaimer: These are just my personal takes, so please take it as a pinch of salt!

**Graph**

1. Graph Traversal Algorithm is just an algorithm. You need to **apply it accordingl**y to solve a problem, meaning that with only the algorithm itself it cannot solve the problem, but you have to **use that algorithm to aid your solution** to the problem

* Apply accordingly doesn’t mean to change the algorithm, but to use it as a part of another solution.
* This is the same for all the other data structures and algorithms. A data structure or algorithm is just a **“tool”** to aid the success of your solution, i.e. with the tool itself we cannot solve the problem, but it must be **used as a part of a solution** so that we can solve a problem.
* Basically this means that you need to develop a solution and use the suitable data structure and algorithm within your solution to aid the success and efficiency of your solution.
* One tip which you can use is **start off with a raw graph which models your current problem first**. Then **determine what algorithm** has the potential to solve your problem. Then, if necessary, **change your graph structure** along the way so that it fits the algorithm (or vice versa, i.e you can first change the graph structure first, then decide which algorithm to use). Or if necessary, do some slight modification to the algorithm to fit your solution.

1. When there is some form of ranking in your graph problem, consider using topological sort.
2. Also, if your problem has **some form of relationships** between entities or **jumping or moving around**, then the problem may be able to converted into a graph problem to be solved.
3. To solve a graph problem, first model the problem as a graph first (i.e. transform the **data** in the problem into a graph)
4. To model a problem as a graph, **first identify** what should the vertex, edge, edge weight, **etc.** represent.
5. Then, determine the nature of the problem: topological ordering? SSSP? ASSP?
6. Then, choose the suitable algorithm according to the nature of the problem **and** the graph data structure we have (esp need to consider what kind of edge: weighted? Directed? Undirected?)
7. Sometimes you need not need to use the graph given to you. You can **transform** it into another graph so that it can speed up your graph traversal / SSSP algorithm or simply make it easier to be implemented.

**Exam**

1. Keep your answer as **succinct** as possible.
2. Even though you can’t give an efficient solution, it’s fine. As long as you give sth that works, then it would be good enough
3. When you face a problem you don’t know how to do, first try to figure out **what topic that may be related** to this problem. Then, think whether if anything that you have learnt in that topic can be used to solve the problem effectively.
4. Try to make the problem size smaller if the problem size is too big to solve.
5. Make sure that your solution is **general,** i.e. can cater to all the cases!
6. If complexity is given, then it is trying to give you hint on the approach to the problem
7. Sometimes when given a problem, you can first think of the brute force method first, then try to improve your solution
8. The only **connected undirected** graph that **doesn’t have a cycl**e is a **tree** (which has V – 1 edges). If the connected undirected graph has >= V edges, then it has at least one simple cycle with 3 or more vertices.
9. “Yes, writing pseudocode is fine. You are not required to write in Java syntax, so you can use pseudocode if you are more comfortable with that, as long as the idea behind your algorithm is clear.”
10. Remember you can always **transform the graph** given to you in order to solve it more efficiently!
11. One last important tip is to **be creative**, think of whatever that is possible as long as the question didn’t say you can’t do it.

**Some terms to keep your answer succinct**

1. **Store x keyed using y**

* Used in BST. It means that you store a pair (x, y) into the BST, but the pair is **ordered according to y.**

**Linear Data Structure**

1. Usually, those kinds of weird questions will require either a stack or queue to solve, but usually it is **stack**
2. **General** rule of thumb when dealing with stack question

* If the stack is **empty**, you **push**
* If the stack is **not empty**, you either **push or pop**

**Graph**

1. If a problem cannot be solved linearly (i.e looping linearly) and there is jumping around, then the problem may be a graph problem.
2. Or if the problem involves **connections**

**Complexity Analysis**

1. **Very IMPORTANT!**

* If let’s say in a complexity analysis question, it also mentions another unknown, say k, e.g. find the kth smallest integer, note that k is a **variable** and hence should be included in big O, unless it is not the leading term!

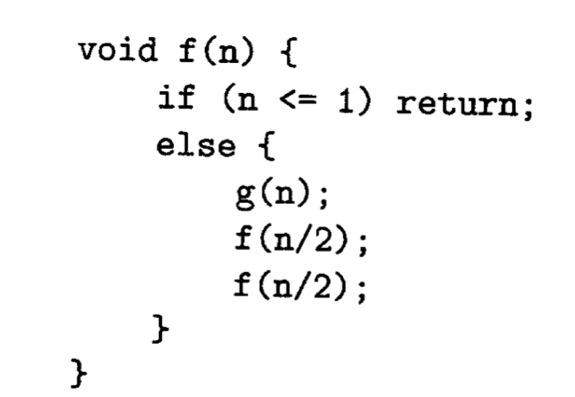
1. **If you see a gp, the last term is the big O notation**
2. If you see a big O of O(xlogn ), where x is a constant, it can be changed to a big O of O(n) as both of them are similar in terms of big O

* E.g. O(2^log n) = O(2^log

1. The analogy of using a rectangle, square or a triangle when counting the number of loops is very effective

* E.g. let’s say you have 1 nested for loop. The outer for loop will run for n times, and for each iteration of the outer for loop, the inner for loop will run for k times
* So the total number of operations is n \* k
* This is similar to a square with length of k and width of n
* Another case is let’s say your outer loop will run for n times, but the inner loop it will run for n times at first, but at every subsequent iteration the number of times it run will be decreased by 1.
* This is similar to finding the area of the half triangle within a square with length n.
* i.e. (n \* n)/2
* The key here is imagine that you have a lot of rows in the square or triangle
* How we calculate area is that starting from one line, assuming that it covers m units, then let’s say we have k similar lines and we arrange them side by side, and then the total area covered by all the lines is the area of the square or rectangle. Since we have k lines and each line covers m units, the total area covered is km, which is exactly the area of the rectangle.
* For the triangle case is let’s say we have an k lines which covers m units, but for each subsequent line it decreases by 1 unit. This is similar to how we form a triangle by using the two opposite vertices of a rectangle or a square. How to imagine this is that imagine you put back the number that is decreased into the rectangle, i.e. starting from second line you put 1 unit, then third line you put back 2 units, and so on. And eventually this will form a rectangle. This means that the area “lost” of the rectangle when forming the triangle is half of the area of the rectangle (as by assembling all the units together we get a half triangle and by fitting this with the another triangle we get a rectangle)

1. **Hot Questions (often come out in exam)**

A close up of text on a whiteboard

Description automatically generated

Relationship between the complexity of g(n) and f(n) in this type of question

|  |  |
| --- | --- |
| **g(n)** | **f(n)** |
| O(1) | O(n) |
| O(n) | O(nlogn) |
| O(n2) | O(n2) |

\*Note that above table only works if the code given is exactly the same or just only the name of the methods or variables changes.

\* Note that O(2^log2 n) = O(2^log n)

\* Note thatO(2^log2 n) = O(n)