Task6: Backtracking approach (20 marks)

This task is about how to apply backtracking to the problem of generating a graph for bordering enclosures.

Part A: Chosen approach (5 marks)

Prepare a written discussion of the manner in which you intend to apply backtracking to this problem. Be sure to respond to the following in your discussion

1. what represents a partial solution, what represents a complete solution

Partial solution can be got when you have not reached to the complete solution and it is the solution we have got so far and then you can generate possible solutions to complete the solution. After that, Possible solution can be more than one because it will generate all the possibilities to move on and end up with the complete solution.

While, complete solution would be your final solution. You don't need anything once you get complete solution, it will represent as the last thing you get to solve the problem.

2. how you know that it will generate a valid graph

The condition is that the graph would be generated as soon as you succeed in getting complete solution. First, you must check whether graph is connected. There will be possible options generated in partial solutions form once you start moving on towards having complete solution. If it keeps producing partial solutions and gets to last partial solution. When that last solution is equal to the length of the list, it would be the complete solution. After that, we can say the graph is valid.

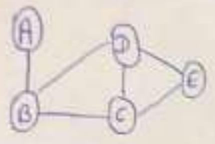
Part B: Explanation (15 marks)

Describe a backtracking algorithm (you do not need to write code for this section) that accepts an animal list and a maximum number of edges (for example the 6 – 5 case discussed in figure 2) and creates a graph representing the optimal solution to this problem. Any graph generated should always be a valid solution to the problem (see background). Draw the backtracking tree using an example that demonstrates your algorithm.

Backtracking will generate a list of all the possible solutions but unlike brute force, it would avoid using unnecessary candidates and would use partial solution as well.

In backtracking, you need to keep track of partial solution, build a set of positions selected. And then you have to keep track of possible next positions and be able to undo steps and check whether partial solution is complete solution. If partial solution is valid then we extend it with next component, else we backtrack to previous component and try another partial solution.

⇛ GRAPH :-



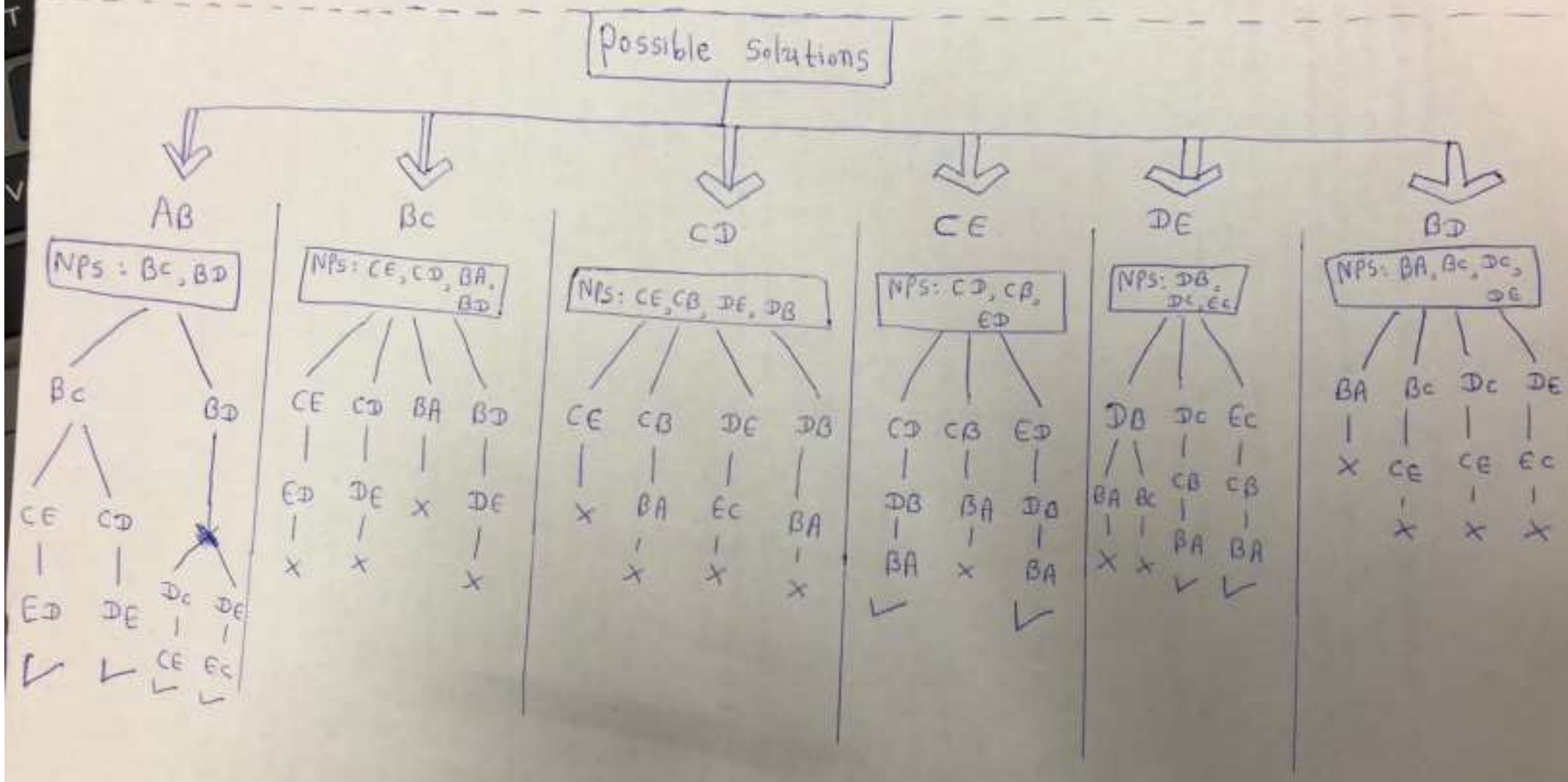
Vertices : A, B, C, D, E

Edges :- AB, BC, CD, CE, DE, BD

NPS : next possible solution

× : can't be solution

✓ : is solution



As given example above, there is a tree made of 5 vertices (A, B, C, D, E) which represents 6 valid edges AB, BC, CD, CE, DE, and BD. We are creating all possible solution without unnecessary components using backtracking. The solution will be made when you travel all vertices through edges. There shouldn't be any vertices repeated in your list of solution. We can say from the number of vertices that partial solution is complete solution if and only if the current elements in the list are equal to number of vertices-1 (4, here). There will be six edges useful to generate complete solutions.

As it can be seen in the diagram, edge AB will look for another connected edge. From the graph above, there are two edges connected with AB which are BC and BD. So, it will go ahead and check for BC first whether it is connected to other edge. It's connected to CE and CD, then first it would go on to check whether CE is connected to any other edge. It's joined with ED. In between all steps, it would keep checking whether list of partial solution is equal to 4. And now, it's equal to 4 so the above step would lead us to complete solution. After that, it would backtrack to BC through CE. We can see BC has two branches so it would go into another branch only to ensure if CD is connected with anything. It is connected with DE so now; the size of that list is 4 as well. It would return the partial solution as complete solution. After that, it would backtrack again. This time, it would go back to AB edge which has another branch BD. We can see that BD edge is joined with edge DC and DE so it would do the same thing as it did in last branch of BC. And BD would be connected with DC and DE which is connected to CE and EC, respectively. So, we would have two more complete solutions because these two lengths would also be 4.

Next, it would take another edge BC which is connected with CE, CD, BA, BD edges. It would first get into CE edge which is joint to ED but now ED is connected to only BA and BC. But partial solution can't take those edges as they are already used so it can't repeat vertices in list so with edge BC, going through edge CE is not a solution. The same thing would happen for every other edges in BC. Edge CD and BD are both connected to DE, but they do not have any other non-repeated edge joint. Outside those two edges, BA is not joint with anyone so in the end, we won't have any solution using BC edge as a starter.

It would backtrack again for good and check for edges in CD. Edge CD is connected to CE, CB, DE and DB. Edge CB, DE and DB are only connected to BA, EC, and BA, respectively. After that, these three edges are disconnected so they won't give any output. Apart from that, CE is disconnected so CD edge won't give any complete solution.

For CE edge, it would be connected with CD, ED, and CB. CD and ED both are joint with DB and DB is joint to BA so these two edges would give us two complete solutions. However, when it backtracks to CB edge, it is connected to BA and BA is joint nowhere so CB does not have complete solution then there will be two complete solutions if you use CE as your starting edge.

The next edge is DE which is joint with DB, DC, and EC valid edges. First, it would check into DB which is connected to BA and BC. Then it would check in BA which is joint nowhere and same stands for BC when it goes to check BC after backtracking. It would then backtrack all the way back to DE and check for DC and EC edges. These two edges are connected with CB and CB is joint with BA so DE edge would also give us two more complete solutions.

The last edge to look for is BD which is connected to BA, BC, DC, and DE. First, BA is connected nowhere so it would backtrack and go on to look into BC edge which is connected to CE, but CE will not give any connection, so it will not be the solution. Nonetheless it would backtrack and look for joint edge in DC which is also connected with CE. As we know CE does not give connection so it would come back and check for edges in DE, which is also connected with EC, but it will not be able to go ahead so it will stop there. In the end, there would be complete solutions for BD edge as starter.

In short, Edge CE, DE and AB give us two, two and four complete solutions respectively. Hence; there are eight of total solutions generated for the graph imported above.

Task7: Comparison of approaches (5 marks)

Prepare a discussion comparing your greedy and backtracking approaches together and their results. Your discussion should address the following points:

1. Scalability of each approach (which is faster for larger lists of animals?)

2. Quality of results for each approach (which produces better quality results) - make sure you explain exactly what it means to have better quality results in this context.

As we discussed earlier in greedy approach that greedy doesn't give optimal solution, but backtracking does.

Greedy only considers best value (minimum/maximum) when it comes to adding current item to the list which is not worthy every time. In other words, greedy can be useful for smaller amounts or range of list. But for larger animal of list, backtracking would be more appropriate method to use.

Backtracking takes care at every single step about efficiency. It never generates unnecessary elements or items in the list. If any item does not exist or is invalid for partial solution in any particular branch, then it backtracks itself and goes on to check for next branch for creating partial solution. So, it does not waste time and number of elements so backtracking would be more efficient approach.

Backtracking doesn't generate unnecessary elements or items in the list. It would only care about the current step and do what is going right now. While in greedy, it only cares about future steps so it only takes maximum and minimum value. That's why greedy ends up having non-optimal solution especially for larger list of elements so Backtracking stands tall for both efficiency and quality result.