Artificial Intelligence
Assignment - 3

Name: Tanish Pagaria
Roll No.: B21AI040

RESOLUTION-REFUTATION AS SEARCH

Problem Statement
To implement a propositional logic theorem prover based on resolution refutation algorithm.

Provided Constraints / Rules
- There is no space between two consecutive operators or variables.
- The following characters are used to denote different operators:
  - OR : **|**
  - AND : **&**
  - NOT : **!**
  - IMPLICATION: **>**
  - IFF (BICONDITIONAL / BIDIRECTIONAL) : **=**
  - OPENING BRACKET : **(**
  - CLOSING BRACKET : **)**

Assumptions
- The variables (literals) are upper-case or lower-case letters from the English alphabet
- While printing the resolution steps, we have printed the corresponding knowledge base obtained after appending the resolvent to the previous knowledge base while performing the search algorithm.
- We have calculated the total number of nodes explored during the search as well as the final path (solution) that takes us to the goal state.

Conversion to Conjunctive Normal Form (CNF)
The following algorithm was employed in order to convert a given formula to its CNF form:
- Eliminating Biconditional (⇔) using the relation: α⇔β ≡ (α⇒β) ∧ (β⇒α)
- Eliminating Implication (⇒) using the relation: α⇒β ≡ ¬αVβ
- Moving all Negations (¬) inwards using:
  - Double Negation Elimination: ¬(¬α) ≡ α
  - De Morgan's Laws: ¬(αVβ) ≡ ¬α∧¬β and ¬(α∧β) ≡ ¬αV¬β
- Applying Distributivity of V over ∧: α∧(βVɣ) ≡ (α∧β) V (α∧ɣ)

Resolution-Refutation
The following steps were used for performing resolution-refutation:
- Adding the negation of the query to the knowledge base
- Converting all the sentences in the knowledge base to CNF
- Showing that the resolution refutation eventually leads to an "Empty Clause" **(Proof by Contradiction)**

Resolution Rule of Inference

$$\frac{\alpha \vee \beta, \neg\beta \vee \gamma}{\alpha \vee \gamma}$$

Algorithm for performing Resolution-Refutation as a Search

The following analogy is used for solving the resolution-refutation problem as a search problem:

- **Start State:** The initial knowledge base (which also includes the negation of the query)
- **Goal State:** The new knowledge base created after resolution-refutation that encounters the addition of an empty clause
- **State Space:** The knowledge bases created after the addition of the resolvent obtained from a resolution-refutation step
- **Successor Function:** Applying resolution-refutation and appending the resolvent to a knowledge base
- **Solution:** The path obtained after successfully reaching the goal state, if it is possible (the query is entailed by the original knowledge base)

The overall steps for the algorithm:

- Initially, we convert the formulas in the knowledge base to their CNF form and then append the CNF form of the negation of the query to the knowledge base.
- We use a data structure (fringe) for applying the search algorithm. The choice of the data structure is based on the type of search method:
    - **Uninformed Search**
        - Breadth First Search (BFS) : Queue
        - Depth First Search (DFS) : Stack
    - **Greedy Search** : Priority Queue (based on the minimum heuristic)
- We use another data structure (a list) to save the pair of clauses that have already been resolved before so that we do not end up performing the same resolution again and again, leading to an infinite loop.
- We push the knowledge base (which we created in the initial step) to the fringe. In a push operation, we push a dictionary containing the knowledge base as well as an array containing the path to reach that knowledge base. This path will get updates for all the successor states (knowledge bases). This provides an easy way to store the path so that no backtracking is required afterward.
- A loop is then initialized that keeps on running until the fringe gets empty (the query is not entailed by the knowledge base).
- We pop the top or front of the fringe (depending on the data structure). Now, we iterate through this knowledge base and take all the pairs of formulas in it, and check if the pair can be resolved or not. If the pair is resolvable and has not been visited before, we push it to the fringe along with the updated path in a dictionary.
- If, at any point during the loop, we encounter the resolvent ending up as an empty clause, we stop the iteration and return that the query is entailed by the knowledge base, the corresponding solution path, the total number of nodes explored, and any other information required, for e.g., total execution time of the algorithm, etc.
- If the fringe gets empty without encountering any empty clauses, we return that the query is not entailed by the knowledge base.

EXPERIMENTAL DETAILS

Programming language used: Python3

Conversion to Conjunctive Normal Form

- The initially mentioned algorithm was employed to carry out the task.
- We defined several helper functions in order to approach the task in a stepwise manner.
- These included functions to handle the placement of the brackets and find the overall terms before and after a given operator. This included a stack-based approach to finding the correct brackets and the corresponding terms.

- We defined functions to deal with eliminating biconditional into implication, implication to disjunction, etc., and handling negation and distributivity.
- Error handling was also ensured up to a certain point. We removed any unnecessary spaces (if present in the input).

Resolution

- We performed the conversion of all the formulae (including the negation of the query) into a knowledge base by splitting the formulas by the separator '&' (AND operator) into a list of smaller constituent formulae. For each of these formulae, the brackets were removed so that the resultant expression would contain only literals connected via '|' (OR operator). The obtained list was used in further operations.
- The resolution was performed by splitting a given pair of formulas, each into a list of literals separated by '|' (OR operator).
- We also sorted the literals based on their ASCII values (in an attempt to maintain uniformity).
- We iterated through both lists, covering all the possible pairs of literals, one from each. If we encountered a pair of literals in which one was a negation of the other, we would stop the iteration and return the new formula after combining the lists, removing that pair of variables, and concatenating the literals using '|' (OR operator).
- We ensured that no literals were repeated in the resultant resolvent expression.
- In case there was no pair found, we return a null value (none).

Search Algorithm

- We defined a single search function to handle the search algorithm for all the types of searches (uninformed: BFS + DFS and greedy) by passing the required method and sub-method as parameters to the function.
- These parameters would divide the workings of the fringe based on the type of search.
- The function would carry out the algorithm following the same steps mentioned above.
- The function would print the required resolution steps (based on the input) and a 1 or 0 representing true or false based on whether the query is entailed by the knowledge base.
- The function would return a dictionary containing the following information:
    - a boolean value, whether the query is entailed or not
    - a list representing the solution path (if query is entailed)
    - the total number of nodes explored (states visited)
    - the execution time of the overall algorithm

Heuristic function (for Greedy Search)

- Heuristic: The count of literals in the resolvent clause added to the knowledge base.
- The reasoning (behind choosing this heuristic and its admissibility):

  Smaller resolvents

  - Clauses with fewer literals are likely to result in a more significant reduction in the search space because they tend to produce smaller resolvents.
  - If C_1 or C_2 are clauses, and $|C_1| < |C_2|$,

    then the resolvent $|R| = |(C_1 \setminus p) \vee (C_2 \setminus \neg p)|$

    will have fewer literals than clauses C_1 or C_2, where p is the literal being resolved.

  Efficient search space reduction

  - Smaller resolvents mean a smaller number of new clauses generated at each step, potentially reducing the number of clauses to explore.
  - If the heuristic consistently selects clauses with $O(k)$ literals, where k is a constant, the resulting resolvents are likely to have $O(k)$ literals, contributing to a reduced search space.

RESULTS & OBSERVATIONS

Applying the search algorithms to the example input given in the assignment:

```
n, m = 4, 1
formulas = ['P|(Q&(R>T))', 'P>R', 'Q>T', 'Q>(R=T)']
query = 'R'
```

Uninformed Search (BFS)

```
[17]: res = solve(n, m, formulas, query, method='uninformed', submethod='BFS')
```

```
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', 'P|!P|T', 'P|!P|!Q|R', '!P|!Q|R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', 'P|!P|T', 'P|!P|!Q|R', 'P|!P|!Q|T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', 'P|!P|T', 'P|!P|!Q|R', 'P|!P|!Q']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', 'P|!P|T', 'P|!P|!Q|R', 'P|!P|!Q|R|T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', 'P|!Q|!R|R', 'P|!Q|!R|T', 'P|!P|!Q|T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', 'P|!Q|!R|R', 'P|!Q|R|!T', 'P|R|!T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', 'P|!Q|!R|R', 'P|!Q|R|!T', 'P|!Q|R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', 'P|!Q|!R|R', 'P|!Q|R|!T', 'P|!Q|!T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', 'P|!Q|!R|R', 'P|!Q|R', 'P|!R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', 'P|!Q|!R|R', 'P|!Q|!R', 'P|!P|!Q']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', 'P|!Q|!R|R', 'P|!Q|!R', 'P|!Q|!T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!P|!Q|T', '!P|!Q|R', 'P|!P|R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!P|!Q|T', '!P|!Q|R', 'P|!P|!Q|T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!P|!Q|T', '!P|!Q|R', '!P|!Q']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!P', '!R|T', '!P|T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!P', '!R|T', '!Q|!R|R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|R', 'P|R', 'P|T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|R', 'P|R', 'R']
!R, R resolved to Empty Clause
1
```

```
[18]: print_results(res)
```

```
total number of nodes explored: 101
execution time: 0.014796 seconds

number of nodes in the path: 5
path:
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|R', 'P|R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|R', 'P|R', 'R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|R', 'P|R', 'R', '']
```

Uninformed Search (DFS)

```
[19]: res = solve(n, m, formulas, query, method='uninformed', submethod='DFS')
```

```
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q', 'P|!Q|R', '!P|!Q|T', 'P|!P|!T', 'P|!Q|R|T', 'P|!P|!Q|T', 'P|!Q|!R|
T', 'P|!Q|!R|R', 'P|!Q|R|!R|T', 'P|!P|!Q|R|T', 'P|!P|!Q|R', 'P|!Q|R|!T', 'P|!Q|R', 'P|!Q|R|T', 'P|!Q|T', '!Q|R|T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q', '!P|!Q|R', '!P|!Q|T', 'P|!P|!T', '!P|!Q|R|T', 'P|!P|!Q|T', 'P|!Q|!R|
T', 'P|!Q|!R|R', 'P|!Q|R|!R|T', 'P|!P|!Q|R|T', 'P|!P|!Q|R', 'P|!Q|R|!T', 'P|!Q|R', 'P|!Q|R|T', 'P|!Q|T', '!Q|R|T', '!Q|R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q', '!P|!Q|R', '!P|!Q|T', 'P|!P|!T', '!P|!Q|R|T', 'P|!P|!Q|T', 'P|!Q|!R|
T', 'P|!Q|!R|R', 'P|!Q|R|!R|T', 'P|!P|!Q|R|T', 'P|!P|!Q|R', 'P|!Q|R|!T', 'P|!Q|R', 'P|!Q|R|T', 'P|!Q|T', '!Q|R|T', '!Q|R', 'P|!Q']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q', '!P|!Q|R', '!P|!Q|T', 'P|!P|!T', '!P|!Q|R|T', 'P|!P|!Q|T', 'P|!Q|!R|
T', 'P|!Q|!R|R', 'P|!Q|R|!R|T', 'P|!P|!Q|R|T', 'P|!P|!Q|R', 'P|!Q|R|!T', 'P|!Q|R', 'P|!Q|R|T', 'P|!Q|T', '!Q|R|T', '!Q|R', 'P|!Q', 'P']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q', '!P|!Q|R', '!P|!Q|T', 'P|!P|!T', '!P|!Q|R|T', 'P|!P|!Q|T', 'P|!Q|!R|
T', 'P|!Q|!R|R', 'P|!Q|R|!R|T', 'P|!P|!Q|R|T', 'P|!P|!Q|R', 'P|!Q|R|!T', 'P|!Q|R', 'P|!Q|R|T', 'P|!Q|T', '!Q|R|T', '!Q|R', 'P|!Q', 'P', 'P|T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q', '!P|!Q|R', '!P|!Q|T', 'P|!P|!T', '!P|!Q|R|T', 'P|!P|!Q|T', 'P|!Q|!R|
T', 'P|!Q|!R|R', 'P|!Q|R|!R|T', 'P|!P|!Q|R|T', 'P|!P|!Q|R', 'P|!Q|R|!T', 'P|!Q|R', 'P|!Q|R|T', 'P|!Q|T', '!Q|R|T', '!Q|R', 'P|!Q', 'P', 'P|T', 'R|T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q', '!P|!Q|R', '!P|!Q|T', 'P|!P|!T', '!P|!Q|R|T', 'P|!P|!Q|T', 'P|!Q|!R|
T', 'P|!Q|!R|R', 'P|!Q|R|!R|T', 'P|!P|!Q|R|T', 'P|!P|!Q|R', 'P|!Q|R|!T', 'P|!Q|R', 'P|!Q|R|T', 'P|!Q|T', '!Q|R|T', '!Q|R', 'P|!Q', 'P', 'P|T', 'R|T', 'T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q', '!P|!Q|R', '!P|!Q|T', 'P|!P|!T', '!P|!Q|R|T', 'P|!P|!Q|T', 'P|!Q|!R|
T', 'P|!Q|!R|R', 'P|!Q|R|!R|T', 'P|!P|!Q|R|T', 'P|!P|!Q|R', 'P|!Q|R|!T', 'P|!Q|R', 'P|!Q|R|T', 'P|!Q|T', '!Q|R|T', '!Q|R', 'P|!Q', 'P', 'P|T', 'R|T', 'T', '!Q']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q', '!P|!Q|R', '!P|!Q|T', 'P|!P|!T', '!P|!Q|R|T', 'P|!P|!Q|T', 'P|!Q|!R|
T', 'P|!Q|!R|R', 'P|!Q|R|!R|T', 'P|!P|!Q|R|T', 'P|!P|!Q|R', 'P|!Q|R|!T', 'P|!Q|R', 'P|!Q|R|T', 'P|!Q|T', '!Q|R|T', '!Q|R', 'P|!Q', 'P', 'R']
!R, R resolved to Empty Clause
1
```

```
[20]: print_results(res)
```

```
total number of nodes explored: 35
execution time: 0.033786 seconds

number of nodes in the path: 27
path:
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q', '!P|!Q|R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q', '!P|!Q|R', '!P|!Q|T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q', '!P|!Q|R', '!P|!Q|T', 'P|!P|!T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q', '!P|!Q|R', '!P|!Q|T', 'P|!P|!T', '!P|!Q|R|T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q', '!P|!Q|R', '!P|!Q|T', 'P|!P|!T', '!P|!Q|R|T', 'P|!P|!Q|T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|!T', '!Q|!R', 'P|!R', 'P|!Q|!T', 'P|!Q|R', 'P|!P|!Q', '!P|!Q|R', '!P|!Q|T', 'P|!P|!T', '!P|!Q|R|T', 'P|!P|!Q|T', 'P|!Q|!R|
T']
```

Greedy Search

```
[21]: res = solve(n, m, formulas, query, method='greedy')
```

```
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!P']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!P', 'Q']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!P', 'Q', 'T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', 'Q|R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', 'Q|R', 'Q']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', 'P|T']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|R', '!Q']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|R', '!Q', 'P']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|R', '!Q', 'P', 'R']
!R, R resolved to Empty Clause
1
```

```
[22]: print_results(res)
```

```
total number of nodes explored: 11
execution time: 0.001151 seconds

number of nodes in the path: 6
path:
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|R', '!Q']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|R', '!Q', 'P']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|R', '!Q', 'P', 'R']
['P|Q', 'P|!R|T', '!P|R', '!Q|T', '!Q|R|!T', '!Q|!R|T', '!R', '!Q|R', '!Q', 'P', 'R', '']
```

Note: *Truncated (scrollable) outputs are shown here due to their large display size; the complete outputs are available to view in the notebook code file.*

Observations based on the example input

Search Algorithm	Total nodes explored	Total nodes in the path	Total execution time (seconds)
Breadth-First Search	101	5	0.014796
Depth-First Search	35	27	0.033786
Greedy Search	11	6	0.001151

- BFS provided the most optimal solution path.
- Greedy Search provided the solution in the least execution time and explored the least number of nodes.
- DFS explored less number of nodes but took more execution time, and the resultant solution had the most number of nodes (non-optimal).

Observations for inputs of different sizes and complexities

n	Breadth-First Search			Depth-First Search			Greedy Search		
	Total nodes explored	Total nodes in path	Total execution time	Total nodes explored	Total nodes in path	Total execution time	Total nodes explored	Total nodes in path	Total execution time
2	3	3	0.000199	12	3	0.000592	3	3	0.000271
3	22	5	0.002245	5	5	0.000419	6	5	0.001116
4	101	5	0.014796	35	27	0.033786	11	6	0.001151
5	86	not entailed	0.019166	106	not entailed	0.031076	86	not entailed	0.01554
6	216	6	0.063814	20	20	0.008618	11	7	0.002034

Note: *These results should be viewed differently (independently) for each value of n since the test examples differed for different values of n. The total execution time in the above table is given in seconds.*

- Greedy Search showed the lowest execution time for almost all values of n.
- The number of nodes explored was maximum in the case of BFS for most values of n, but it also gave the most optimal path to the solution in all the cases.
- For smaller values of n, all the search methods gave the same solution path.
- The execution time for DFS was the highest for most values of n, while for some, BFS took a higher execution time since it explored more nodes.
- The inputs played an important role in deciding the overall results between BFS and DFS. Greedy Search worked similarly for all the cases.

ANALYSES & CONCLUSION

Optimality of solution paths

- BFS produces optimal solution paths despite exploring a larger number of nodes.
- DFS yields non-optimal paths due to its depth-first nature, exploring deep branches before backtracking.

Exploration efficiency

- BFS explores a larger number of nodes exhaustively, ensuring the most direct solution path.
- DFS explores fewer nodes but might miss optimal paths due to its depth-first exploration.

Execution time

- BFS generally takes more time due to exhaustive exploration. DFS might take longer in some cases.
- Greedy Search demonstrates the shortest execution time, guided by an efficient heuristic.

Search method comparison

- Uninformed methods are reliable but computationally expensive.
- Greedy Search efficiently balances exploration and execution time.

Impact of problem size

- For smaller problems, all methods yield similar results.
- Differences in efficiency and execution time become more pronounced with larger and more complex problems.
