German University in Cairo
Department of Computer Science
Dr. Nourhan Ehab

Introduction to Artificial Intelligence, Winter Term 2024
Project 1: Water Sort

Due Friday, October 18th by 23:59

1. **Project Description** In this project, you will be implementing a search agent that solves the water-sort puzzle. The puzzle consists of several bottles each consisting of several layers of liquids of different colours. The objective is to make it so that in every bottle all layers are of the same colour. The agent can achieve that by pouring from one bottle to another.

   Each bottle has a given capacity (maximum number of layers it can hold) and is assigned an identifier number (0 to n) according to its order in the given initial state. The only action the agent can do is pour from one bottle to another. In a given instance of the game, all bottles have the same capacity but this capacity and the number of bottles vary across instances of the problem. There are only 5 possible colors: red, green, blue, yellow and orange.

   Given an initial description of the state, the agent should be able to search for a plan (if such a plan exists) to achieve its objective. In the following sections there are more details on the agent, actions and the implementation requirements.

2. **Actions:**

   The agent can perform one action: `pour(i, j)`. This action results in pouring from bottle `i` to bottle `j`. To be able to perform this action:

   - There must be at least 1 empty layer in bottle `j`.
   - Bottle `i` must not be empty.
   - Either the topmost layers in each bottle are of the same colour, or bottle `j` is empty.

   As a result of performing this action, some layers from bottle `i` will be transferred to bottle `j`. The number of poured layers will be the minimum between the empty spaces in bottle `j` and the topmost layers in bottle `i` sharing the same colour.

   The optimality criterion is to minimize the number of layers poured.

3. **Implementation:** In this project, you will implement a search agent that tries to find a sequence of actions that achieves the goal. The agent has succeeded when each bottle has all its layers of the same color. Optimally, you would like to minimise total amount of liquid layers transferred to achieve this. Several search strategies will be implemented and each will be used to plan the agent's solution to this problem. The search is to be implemented as described in the lectures (Lecture 2, Slide 15):

   a) Breadth-first search.
   b) Depth-first search.

c) Iterative deepening search.

d) Uniform cost search.

e) Greedy search with at least two heuristics.

f) A* search with at least two *admissible* heuristics. A trivial heuristic (e.g., $h(n) = 1$) is not acceptable.

Different solutions should be compared in terms of run-time, number of expanded nodes, memory (RAM) utilisation and CPU utilisation. **You are required to implement this agent using Java.**

Your implementation should have the following classes:

- `GenericSearch`, which has the generic implementation of a search problem (as defined in Lecture 2).
- `WaterSortSearch`, which extends `GenericSearch`, implementing the "Water Sort" search problem.
- `Node`, which implements a search-tree node (as defined in Lecture 2).

You can implement any additional classes you want. Inside `WaterSortSearch` you will implement `solve` as the key function which will be the basis for testing :

- `solve(`*initialState*`,`*strategy*`,` *visualize*`)` uses search to find a sequence of steps to help separate all the colours if such a sequence exists.
  - `initialState` a provided string that defines the parameters of the instance of the problem. It gives the initial content of each bottle. It is a string provided in the following format:

    $numberOfBottles$;
    $bottleCapacity$;
    $color_{0,1}, color_{0,2}, ...color_{0,k}$;
    $color_{1,1}, color_{1,2}, ...color_{1,k}$;
    .
    .
    .
    .
    .
    $color_{n,1}, color_{n,2}, ...color_{n,k}$;

    Where

    - $numberOfBottles$ is the number of bottles in this problem.
    - $bottleCapacity$ is the maximum number of layer each bottle can take (they do not all have to be full initially)
    - $color_{n,k}$ refers to the color of the kth layer in the nth bottle. Bottle IDs start at 0. Layer indices start from 0 being the top layer and the last being the bottom. The colours are given as the first letter of the color's name (e.g. r for red, g for green...etc.). Empty layers will be denoted with e.

    Note that the string representing the initial state does not contain any spaces or new lines. It is just formatted this way here to make it more readable.

- `strategy` is a string indicating the search strategy to be applied:
  - `BF` for breadth-first search,
  - `DF` for depth-first search,
  - `ID` for iterative deepening search,
  - `UC` for uniform cost search,
  - `GR`i for greedy search, with $i \in \{1, 2\}$ distinguishing the two heuristics.
  - `AS`i for A* search with $i \in \{1, 2\}$ distinguishing the two heuristics.
- `visualize` is a Boolean parameter which, when set to `true`, results in your program's side-effecting displaying the state information as it undergoes the different steps of the discovered solution (if one was discovered). *A GUI is not required, printing to the console would suffice.* The main value of this part is to help you debug and understand.

The function returns a `String` of 3 elements, in the following format:
`plan;pathCost;nodesExpanded`
where:

- `plan`: the sequence of actions that lead to the goal (if such a sequence exists) separated by commas.
  For example:

  ```
  pour_0_3,pour_0_4,pour_1_3,pour_1_4,pour_0_1,pour_0_3,
          pour_2_4,pour_2_1,pour_2_3,pour_2_4
  ```

- `pathCost`: the total cost the sequence of actions from the initial state through the path to the goal.
- `nodesExpanded`: is the number of nodes chosen for expansion during the search.

  If there is no possible solution, the string 'NOSOLUTION' should be returned. Please make sure you use the exact string formats specified for the tests to pass.

4. Examples:

```
String init = "5;4;" + "b,y,r,b;" + "b,y,r,r;" +
"y,r,b,y;" + "e,e,e,e;" + "e,e,e,e;";
```

The search tree for the problem with this initial state could look something similar to that shown in 1. This example is also to illustrate the possible actions, their effects and their costs. One possible solution the agent could find is:

```
pour_0_3,pour_0_4,pour_1_3,pour_1_4,pour_0_1,pour_0_3,pour_2_4,pour_2_1,
                                                    pour_2_3,pour_2_4;10;7493
```

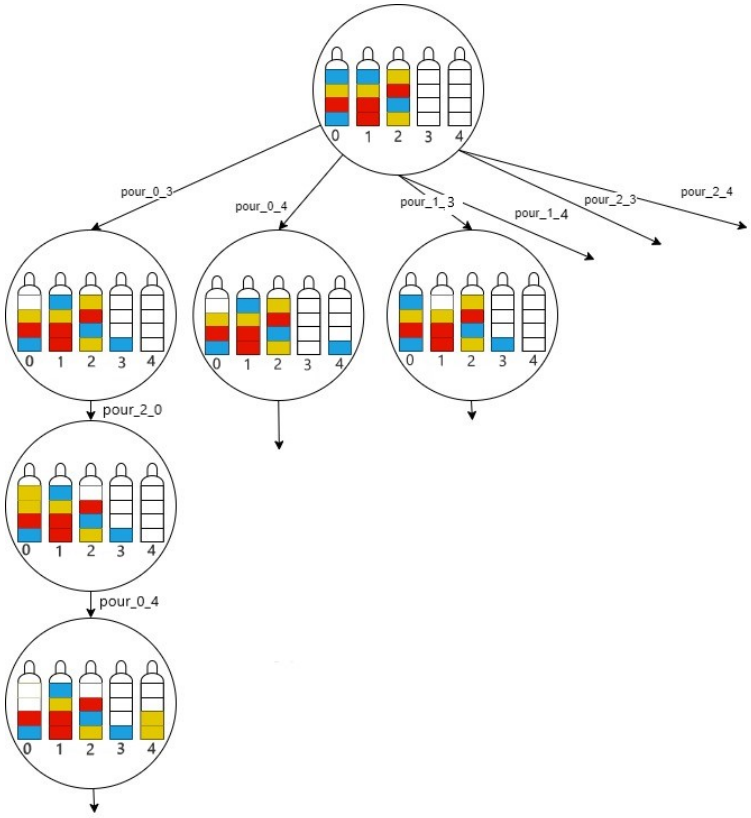

Figure 1: Please note that this example illustrates only one possible sequence of actions and not necessarily one that leads to the goal. The representation of the state also is just for illustration and not necessarily one you must follow.

5. Groups: You may work in groups of at most four.

6. Deliverables

a) Source Code

- You should implement an abstract data type for a search-tree node as presented in class. (*This does not necessarily mean that your implementation should include Java abstract classes.*)

- You should implement an abstract data type for a generic search problem, as presented in class.
- You should implement the generic search procedure presented in class(Lecture 2 slides), taking a problem and a search strategy as inputs.You should make sure that your implementation of search minimises or eliminates redundant states and that all your search strategies terminate within the time limits of the automated public test cases that will be posted.
- You should implement a WaterSortSearch subclass of the generic search problem. This class must contain `solve` as a static method.
- You should implement all of the search strategies indicated above (together with the required heuristics).
- Your program will be subject to both public and private test cases.
- Your source code should have two packages. A package called `tests` and a package called `code`. All your code should be located in the `code` package and the test cases (when posted) should be imported in the `tests` package.

b) Project Report, **not exceeding 10 pages** and including the following.

- A class diagram showing the main classes and functions in your implementation.
- A discussion of how you implemented the various search algorithms.
- A discussion of the heuristic functions you employed and, in the case of greedy or A*, an argument for their admissibility.
- A comparison of the performance of the different algorithms implemented in terms of **completeness, optimality, RAM usage, CPU utilization, and the number of expanded nodes**. You should comment on the differences in the RAM usage, CPU utilization, and the number of expanded nodes between the implemented search strategies.
- Proper citation of any sources you might have consulted in the course of completing the project.
- If you use code available in library or internet references, make sure you *fully* explain how the code works and be ready for an oral discussion of your work.
- If your program does not run, your report should include a discussion of what you think the problem is and any suggestions you might have for solving it.

7. Grading Criteria

The following are *some* of the main points regarding how the submission is evaluated:

- The report is a major component of the grade (30-40)%
- The public and private tests are the remaining grade.
- The incorrect implementation of search and each strategy (not using search or incorrectly implementing the strategies will result in nullifying the corresponding test grades)
- Optimal strategies should result in optimal solutions with respect to path cost.
- The tests posted will provide some initial states and expect a solution if one exists. They will succeed if the solution represents a valid sequence of steps that would lead to the goal. However, they will not be checking the optimality or the correct implementation of the strategy, these are up to you to verify as they are part of the grade.

8. Important Dates

Team Submission Make sure you submit your team member details by October 4th at 23:59 using the following link `https://forms.gle/HV9LcD1ruRShgadC7`. Only one team member should submit this for the whole team. After this deadline, we will be posting on the CMS a team ID for each submitted team. You will be using this team ID for submission. Any student enrolled in the course that does not have a submitted team will be considered to work alone. No random assignment will be performed.

Source code and Project Report On-line submission by October 18th at 23:59. The submission details will be announced after the team submission deadline.

Brainstorming Session. In tutorials.