German University in Cairo
Department of Computer Science
Assoc. Prof. Haythem O. Ismail

Introduction to Artificial Intelligence, Winter Term 2023
Project 1: Live Long and Prosper

Due Friday, November 10th by 23:59

1. Project Description

There is a town that requires some resources (food, materials, energy) for the citizens inside or for the establishment of new buildings. The town has a prosperity level indicating how well its people are doing. Buildings need to be established to make the town more prosperous and these buildings require some resources. For this project, your task is to design a search agent responsible for finding a plan that will help this town reach a prosperity level of 100. Different actions can affect the prosperity and resources differently.

You have a budget of 100,000 to spend and there is no additional source of income. There is a limit for the amount of resources you can store in this town at any time (50 units per resource). Resources deplete with every action the agent does. To increase the level of a resource, a delivery has to be requested. Consuming resources also results in spending money which is the cost of these resources.

Given an initial description of the state and the possible actions, the agent should be able to search for a plan (if such a plan exists) to get this town to prosperity level 100. In the following sections there are more details on the agent, actions and the implementation requirements.

2. Actions:

The agent can perform the following actions.

- **RequestFood, RequestMaterials, RequestEnergy:** These actions request a delivery of a resource to the town. A delivery increases the level of the resource by a certain amount. However, a deliveries are not immediate, so the increase in the amount of a resource is not necessarily effective in the next step. Hence, each of the three actions is parameterized by two parameters, `amount` and `delay`, that will be provided in the problem initialization. During the delay period following the request of a delivery, no other delivery can be requested (for the same or a different resource). Thus, the agent can only wait or build during that time. Delivery actions have no effect on prosperity level.
- **WAIT:** This action allows the agent to wait for a delivery to arrive without building or requesting a new delivery. It is **only** allowed while a delivery is pending. This action has no effect on prosperity level.
- **BUILD1, BUILD2:** These are the only actions that affect the town's prosperity level. However, they consume a large amount of resources and an added price. They represent two different types of builds and each has different resource consumption, price and increase in prosperity. These parameters are also given in the problem initialization.

Some additional notes on actions:

- If any of the resources or money reaches zero, no more actions are possible at this state (a corresponding search tree node has no children).
- `WAIT, RequestFood, RequestMaterials` and `RequestEnergy` each result in consuming one unit of each resource.
- Any action results in spending an amount of money equal to the cost of the resources it consumes. For `BUILD1`, `BUILD2` there is an additional sum (provided in the problem initialization).
- The effects and parameters of actions are summarized in Table 1.

Action	Price	Food	Materials	Energy	Prosperity	Amount	Delay
RequestFood	resources	1	1	1	0	given	given
RequestMaterials	resources	1	1	1	0	given	given
RequestEnergy	resources	1	1	1	0	given	given
WAIT	resources	1	1	1	0	0	0
BUILD1	resources+given	given	given	given	given	0	0
BUILD2	resources+given	given	given	given	given	0	0

Table 1: A summary of the effects of actions. Provided values are fixed for all problem instances. *resources* indicates the total cost of the food, materials and energy used. *given* indicates a parameter that will be provided in the problem initialization.

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 the town has reached prosperity level 100. Optimally, you would like to minimise the amount of money spent 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 23):

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).
- `LLAPSearch`, which extends `GenericSearch`, implementing the "Live Long and Prosper" search problem.
- `Node`, which implements a search-tree node (as defined in Lecture 2).

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

- `solve(`*initalState*`,`*strategy*`,` *visualize*`)` uses search to find a sequence of steps to help the town achieve prosperity if such a sequence exists.
  - `initalState` a provided string that defines the parameters of the instance of the problem. It gives the initial values of the resources, their prices and the values of the effects of the actions. It is a string provided in the following format:

    $initialProsperity$;
    $initialFood, initialMaterials, initialEnergy$;
    $unitPriceFood, unitPriceMaterials, unitPriceEnergy$;
    $amountRequestFood, delayRequestFood$;
    $amountRequestMaterials, delayRequestMaterials$;
    $amountRequestEnergy, delayRequestEnergy$;
    $priceBUILD1, foodUseBUILD1,$
    $materialsUseBUILD1, energyUseBUILD1, prosperityBUILD1$;
    $priceBUILD2, foodUseBUILD2,$
    $materialsUseBUILD2, energyUseBUILD2, prosperityBUILD1$;

    where
    - $initialProsperity$ represents the town's initial prosperity level (maximum is 100).
    - $initialFood, initialMaterials, initialEnergy$; indicate the starting levels of food, materials and energy respectively (maximum is 50).
    - $unitPriceFood, unitPriceMaterials, unitPriceEnergy$; the price per unit of the consumption of each of the resources (consumption as a result of any action).
    - $amountRequestFood$ is the amount of food received when a food delivery arrives.
    - $delayRequestFood$ is the number of time-steps until the food delivery arrives and it has a maximum value of 2. Meaning, if it is made at step t, the food level increases at step $t + delayRequestFood + 1$, there will be further elaboration on that in the Examples section. (A time step is the time it takes for the agent to do one action).
    - $amountRequestMaterials, delayRequestMaterials$;
      $amountRequestEnergy, delayRequestEnergy$; are the same as $amountRequestFood, delayRequestFood$; but for delivery requests of the 2 other resources.
    - $priceBUILDi$ the additional money price of $BUILDi$. Please note that $priceBUILDi$ is in addition to the price of the other resources it consumes.
    - $foodUseBUILDi, materialsUseBUILDi, energyUseBUILDi$ The consumption of each resource as a result of $BUILDi$.
    - $prosperityBUILDi$; The increase in prosperity resulting from $BUILDi$.

    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;monetaryCost;nodesExpanded`
where:

- `plan`: the sequence of actions that lead to the goal (if such a sequence exists) separated by commas.
  For example: `RequestFood,WAIT,BUILD1,RequestMaterials,WAIT,WAIT,BUILD2, RequestEnergy,BUILD1`. (These are all of the possible operator names.)
- `monetaryCost`:the money spent during 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 for the tests to pass.

4. Example:

```
String init = "50;"+
              "22,22,22;" +
              "50,60,70;" +
              "30,2;19,1;15,1;" +
              "300,5,7,3,20;" +
              "500,8,6,3,40;";
```

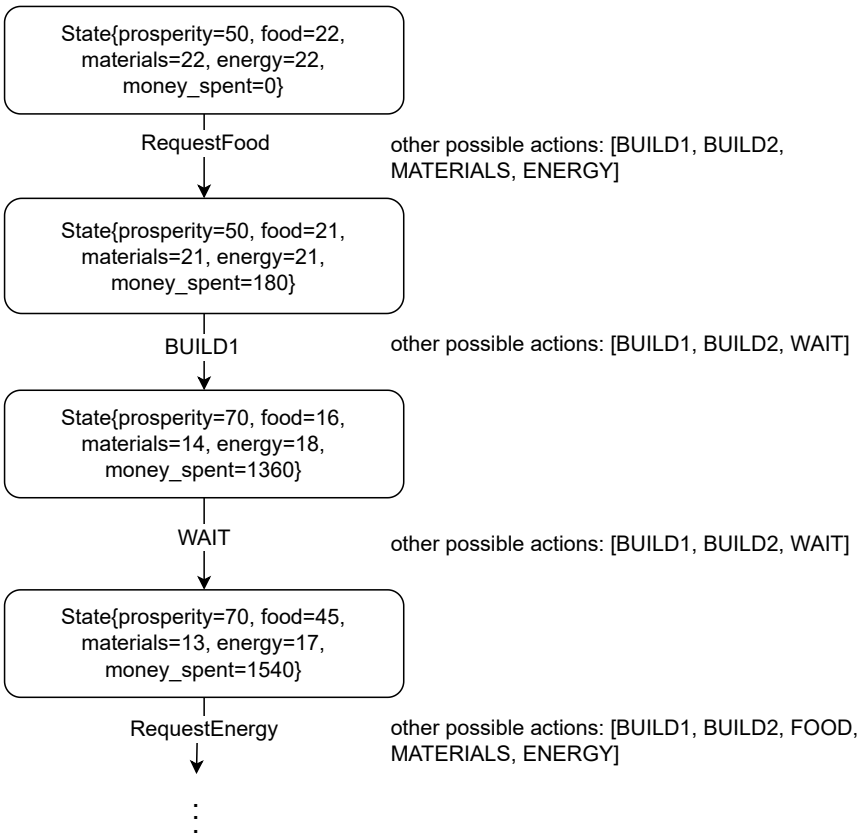

Figure 1: Please note that this example illustrates only one possible sequence of actions and not necessarily one that leads to the goal. The representation of the state also is just for illustration and not necessarily one you should follow.

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 LLAPSearch 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, including the following.

- A brief discussion of the problem.
- A discussion of your implementation of the search-tree node ADT.
- A discussion of your implementation of the search problem ADT.
- A discussion of your implementation of the LLAP problem.
- A description of the main functions you implemented.
- 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. Important Dates

Team Submission Make sure you submit your team member details by October 14th at 23:59 using the following link `https://forms.gle/cUaiQcKnn96E9naT8`. 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.

Source code and Project Report On-line submission by Friday, November 10th at 23:59. The submission details will be announced after the team submission deadline.

Brainstorming Session. In tutorials.