

<b>Started on</b>	Thursday, 7 November 2024, 8:13 AM
<b>State</b>	Finished
<b>Completed on</b>	Thursday, 7 November 2024, 11:13 AM
<b>Time taken</b>	3 hours

**Information**

## Frontier trace format

This information box appears in any quiz that includes questions on tracing the frontier of a graph search problem.

### Trace format

- Starting with an empty frontier we record all the calls to the frontier: to add or to get a path. We dedicate one line per call.
- When we ask the frontier to add a path, we start the line with a + followed by the path that is being added.
- When we ask for a path from the frontier, we start the line with a – followed by the path that is being removed.
- When using a priority queue, the path is followed by a comma and then the key (e.g. cost, heuristic, f-value, ...).
- The lines of the trace should match the following regular expression (case and space insensitive):  
`^[+-] [a-z]+(, \d+)?!?$`
- The symbol ! is used at the end of a line to indicate a pruning operation. It must be used when the path that is being added to, or is removed from the frontier is to a node that is already expanded. Note that if a path that is removed from the frontier is pruned, it is not returned to the search algorithm; the frontier has to continue removing paths until it removes one that can be returned.

### Precheck

You can check the format of your answer by clicking the "Precheck" button which checks the format of each line against a regular expression. Precheck only looks at the syntax of your answer and the frontier that must be used. It does not use the information in the graph object (so for example, + a would pass the format check for a BFS or DFS question even if the graph does not have a node called a).

If your answer fails the precheck, it will fail the check. If it passes the precheck, it may pass the main check, it may not. You will not lose or gain any marks by using "Precheck".

### Notes

- All state-space graphs are directed.
- In explicit graphs, the order of arcs is determined by

the order of elements in `edge_list`.

- Frontier traces are not case sensitive - you don't have to type capital letters.
- There can be any number of white space characters between characters in a trace line and in between lines. When evaluating your answer, white spaces will be removed even if they are between characters.
- You can have comments in a trace. Comments start with a hash sign, `#`. They can take an entire line or just appear at the end of a line. Any character string appearing after `#` in a line will be ignored by the grader. You can use comments to keep track of things such as the *expanded* set (when pruning).
- In questions where the search strategy is cost-sensitive (e.g. LCFS) you must include the cost information on all lines. This means that in every trace line, the path is followed by a comma and a number.
- In questions where a priority queue is needed, the queue must be stable.

### Question 1

Not answered

Mark 0.00 out of 6.00

Given the following graph, trace the frontier of a depth-first search (DFS). Do not perform pruning.

```
ExplicitGraph(
  nodes={'A', 'B', 'C', 'G'},
  edge_list=[('A', 'B'), ('A', 'G'), ('B', 'C'), ('C', 'A'), ('C', 'G')],
  starting_nodes=['A', 'B'],
  goal_nodes={'G'},)
```

**Answer:** (penalty regime: 20, 40, ... %)

1 ||

**Question 2**

Not answered

Mark 0.00 out of 7.00

Write a class `LCFSFrontier` such that an instance of this class along with a graph object and `generic_search` perform lowest-cost-first search (LCFS) without pruning.

**Notes**

- You can download [search.py](#) for local testing if needed.
- Priority queues must be stable.
- You can complete the code preloaded in the answer box if you wish.
- Recall that a path is a sequence of Arc objects. Each arc object has a tail, a head, an action, and a cost.
- Remember to include all the required import statements.
- 

**For example:**

Test	Result
<pre>from search import * from student_answer import LCFSFrontier  graph = ExplicitGraph(     nodes = {'S', 'A', 'B', 'G'},     edge_list=[('S','A',3),                 ('S','B',1), ('B','A',1), ('A','B',1),                 ('A','G',5)],     starting_nodes = ['S'],     goal_nodes = {'G'})  solution = next(generic_search(graph,                                LCFSFrontier())) print_actions(solution)</pre>	<pre>Actions: S-&gt;B, B-&gt;A, A-&gt;G. Total cost: 7</pre>

**Answer:** (penalty regime: 0, 10, 20, ... %)

Reset answer

```

1 from search import *
2 import heapq
3
4
5 class LCFSFrontier(Frontier):
6     """Implements a frontier appropriate
7
8     def __init__(self):
9         """The constructor takes no arguments
10         container to an empty stack."""
11         self.container = []
12         # add more code if necessary
13
14
15     def add(self, path):
16         raise NotImplementedError # FIX
17
18     def __iter__(self):
19         """The object returns itself because
20         method and does not need any additional
21         return self
22
23     def __next__(self):
24         if len(self.container) > 0:
25             raise NotImplementedError #
26         else:
27             raise StopIteration # don't
28
```

29  
30  
31**Question 3**

Not answered

Marked out of  
6.00

Determine whether the following statements are true or false. In all the statements "the algorithm" refers to A\* graph search without pruning. In all cases assume that graphs are finite and the cost and heuristic values are non-negative and finite.

Note: Incorrect answers will result in some negative marks. You may choose to leave some items blank, or select "No Answer," which will be treated the same as leaving it blank. However, selecting "No Answer" will prevent an "incomplete" warning from appearing at the end of the exam.

- If  $h_1(\text{node})$  and  $h_2(\text{node})$  are two valid heuristic functions for the algorithm, then the heuristic function  $h(\text{node})$  defined as  $\max(h_1(\text{node}), h_2(\text{node}))$  is also a valid heuristic function for the algorithm.  ✖
- If  $h_1(\text{node})$  and  $h_2(\text{node})$  are two valid heuristic functions for the algorithm, then the heuristic function  $h(\text{node})$  defined as  $\min(h_1(\text{node}), h_2(\text{node}))$  is also a valid heuristic function for the algorithm.  ✖
- If a heuristic function underestimates the cost to a goal node, the algorithm may find a non-optimal solution.  ✖
- If a heuristic function underestimates the cost to a goal node, the algorithm may not find a solution when there is one.  ✖
- If a heuristic function overestimates the cost to a goal node, the algorithm may find a non-optimal solution.  ✖
- If a heuristic function overestimates the cost to a goal node, the algorithm may not find a solution when there is one.  ✖

One tiny technicality with "positive" is that it can be a "halving" series. Something like "greater than 1" would fix this problem but may make the question confusing.

**Question 4**

Not answered

Marked out of  
6.00

A top-down proof procedure can be seen as a recursive procedure that takes a query, selects a relevant clause from the knowledge base, reduces the query to a new query and calls itself. If a reduction fails, other selections (if available) will be tried.

Assume that we apply the top-down proof procedure to the query  $c$ ,  $a$  and the following knowledge base. For each clause determine if it will be selected by the procedure. Do not assume any particular order of clauses. In cases where selecting a clause depends on the order of clauses, or success or failure of previous selections, choose "may or may not". All the statements (including the term "never") are for the given query.

 $a :- h.$ 

✗

 $b :- f.$ 

✗

 $a :- g.$ 

✗

 $c :- d, f.$ 

✗

 $d :- k.$ 

✗

 $e :- f.$ 

✗

 $c.$ 

✗

 $g.$ 

✗

**Notes**

- Assume that you know that the knowledge base contains the above clauses but you don't know in what order these clauses appear.

### Question 5

Not answered

Mark 0.00 out of 6.00

Write a predicate `all_distinct(+List)` that is true if all the elements in `List` are distinct (i.e. no element appears more than once).

**For example:**

Test	Result
<pre>test_answer :-     all_distinct([1,2,3,4]),     writeln('OK').</pre>	OK
<pre>test_answer :-     all_distinct([a,b,c,b]),     writeln("Wrong!").</pre>	

**Answer:** (penalty regime: 0, 10, ... %)

Question 6

Not answered

Mark 0.00 out of 6.00

Write a predicate `inorder(+Tree, ?Traversal)` that determines the *inorder* traversal of a given binary tree. Each tree/subtree is either a leaf or of the form `tree(root, left_subtree, right_subtree)`. An inorder traversal records the left branch, then the current node, then the right branch.

For example:

Test	Result
<code>test_answer :- inorder(tree(1, leaf(2), leaf(3)), T), writeln(T).</code>	<code>[2,1,3]</code>
<code>test_answer :- inorder(tree(a, tree(b, leaf(c), leaf(d)), leaf(e)), T), writeln(T).</code>	<code>[c,b,d,a,e]</code>

Answer: (penalty regime: 0, 20, ... %)

1

Information

Some of the following questions need the [csp module](#).  
Save the file as `csp.py`. Read the module code and documentation (doc strings).  
  
The `csp` module is available on the quiz server, that is, you can import the module in your code; just remember to include the import statement in your answer.

Variable and constraint representation

We use Python dictionaries to represent assignments. The keys represent variable names (of type string) and the values are the values of the variables. For example `{ 'a': 3 }` is an assignment in which variable *a* has taken the value 3.  
  
A constraint can be represented as a lambda expression, which is an anonymous function that evaluates a condition based on specific variables. For example, the constraint that two variables *x* and *y* must sum to 10 can be written as `lambda x, y: x + y == 10`. This lambda expression returns `True` if the sum of *x* and *y* equals 10, and `False` otherwise.

Some functions in the csp module

The function `satisfies` from the `csp` module tests whether an assignment satisfies a constraint. For example `satisfies({'a':1, 'b':2}, lambda a: a > 0)` return `True`.  
  
The function `scope` in the `csp` module returns the set of the names of formal parameters of a function. For example, `scope(lambda a, b: a + b)` returns `{ 'a', 'b' }`.

Question 7

Not answered  
Mark 0.00 out of 6.00

Make the following CSP *arc consistent* by modifying the code (if necessary) and pasting the result in the answer box.

```
from csp import CSP

csp_instance = CSP(
    var_domains = {var:{1,2,3,4} for var in 'abcd'},
    constraints = {
        lambda a, b: a >= b,
        lambda a, b: b >= a,
        lambda a, b, c: c > a + b,
        lambda d: d <= d,
    }
)
```

Note: remember to include the import statement in your answer.

For example:

Test	Re
	[ 'a', 'b', 'c', 'd', 4

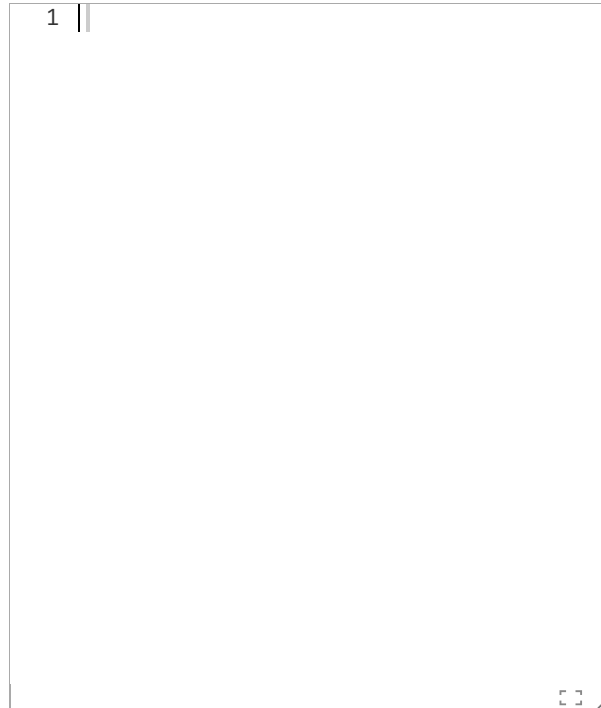


```
from csp import CSP
from student_answer import csp_instance

assert type(csp_instance) is CSP
print(sorted(csp_instance.var_domains.keys()))
print(len(csp_instance.constraints))
```

**Answer:** (penalty regime: 25, 50, ... %)

1 ||



A common misunderstanding seems to be as the following:

"Shouldn't only 1 be valid for the domains of a and b (as I first put) rather than both 1 and 2? since a and b have to be equal and c has to be greater than the sum of them, if a and b are both 2 theres no valid value for c since it would be minimum 5. the valid domains for C being 3 and 4 support this too."

The problem in this argument is that it assumes that making consistent is a global process; it is not. Arcs are considered locally. A global version of this would be exponential.

**Information**

Another way of representing a CSP is by a collection of *relations*. There will be one relation per constraint in the problem. A relation is basically a table that holds zero or more rows. The columns of the table are the variables that are in the scope of the constraint. See the documentation of the `Relation` class in the `csp` module.

For example, the following instance of CSP

```
CSP(
    var_domains = {var:{0,1,2} for var in 'ab'},
    constraints = {
        lambda a, b: a > b,
        lambda b: b > 0,
    })
```

can be represented in relational form as the following:

```
[
    Relation(header=['a', 'b'],
             tuples={(2, 0),
                    (1, 0),
                    (2, 1)}),

    Relation(header=['b'],
             tuples={(2,),
                    (1,)}),
]
```

**Notes**

- Since the variable names and the effect of their domains appear in the relations, there is no need to specify them separately.
- Here we are using a list for the collection of relations instead of a set to avoid the need for creating hashable objects (required for Python sets). From an algorithmic perspective, the order of relation objects in the collection does not matter, therefore, a set would have been a more natural choice.
- Single (one-element) tuples in Python require an extra comma at the end, for example, `(2,)`.
- You can write a short function that takes a CSP object and returns a collection of Relations equivalent to the object.

Question 8

Not answered

Mark 0.00 out of 6.00

Convert the following instance of CSP to an equivalent list of relations called `relations`. In each relation, the variables must appear in alphabetic order. The order of relations in the outer list is not important. Then use the *variable elimination* algorithm to eliminate variable ***b*** and produce a new list of relations called `relations_after_elimination`.

```
csp = CSP(
    var_domains = {var:{-1,0,1} for var in 'abc'},
    constraints = {
        lambda a, b: a * b > 0,
        lambda b, c: b + c > 0,
    }
)
```

For example:

Test	Result
from student_answer import relations, relations_after_elimination from csp import Relation	True True
print(type(relations) is list) print(all(type(r) is Relation for r in relations)) print() print(type(relations_after_elimination) is list) print(all(type(r) is Relation for r in relations_after_elimination))	True True

Answer: (penalty regime: 25, 50, ... %)

Reset answer

```
1 from csp import Relation
2
3 relations = [
4     ### COMPLETE ###
5 ]
6
7 relations_after_elimination = [
8     ### COMPLETE ###
9 ]
10
11
12
13
14
```

**Question 9**

Not answered

Marked out of 4.00

A CSP instance can be solved as an optimisation problem by using an appropriate heuristic (cost) function. When using the greedy descent algorithm to solve these problems, the algorithm may get stuck in local optima. Which of the following is commonly used to alleviate this issue?

Select one or more:

- ☐ Taking some random steps during the search;
- ☐ Restarting the search from a new random state;
- ☐ Using hill climbing instead of greedy descent and then inverting the solution;
- ☐ Using a heuristic function that underestimates the cost;
- ☐ Using a heuristic function that overestimates the cost;

Your answer is incorrect.

**Question 10**

Not answered

Mark 0.00 out of 7.00

Suppose we are using Genetic Algorithms to evolve antennas. For any given antenna (individual) we can measure how much error is made when it is used for communication. The aim is to evolve an individual that can provide low-error communication.

Write a function `select(population, error, max_error, r)` that takes a list of individuals, an error function, the maximum error value possible, and a floating-point random number  $r$  in the interval  $[0, 1)$ , and selects and returns an individual from the population using the roulette wheel selection mechanism. The fitness of each individual must be inferred from its error and the maximum error possible. The lower the error the better.

**Further notes and requirements**

- The error function (which is provided as the second argument) takes an individual and returns a non-negative number which is its error.
- The maximum error (the third argument) is the highest communication error that is possible for any antenna. The population does not necessarily contain an antenna that produces this much error.
- The fitness of an antenna that causes the maximum error is zero. The fitness of an antenna that does not make any error must be as large as the value of `max_error`. This means that you have to write (a very simple) expression to convert the value of error and `max_error` to a fitness value.
- When constructing the roulette wheel, do not change the order of individuals in the population.

**For example:**

Test	Result

<pre>from student_answer import select  population = ['a', 'b']  def error(x):     return {'a': 14,             'b': 12}[x]  max_error = 15  for r in [0, 0.1, 0.24, 0.26, 0.5, 0.9]:     print(select(population, error,                  max_error, r))  # since the fitness of 'a' is 1 and the # fitness of 'b' is 3, # for r's below 0.25 we get 'a', for r's # above it we get 'b'.</pre>	<pre>a a a b b b</pre>
---	------------------------

**Answer:** (penalty regime: 0, 15, ... %)

1 ||



**Question 11**

Not answered

Marked out of  
4.00

Complete the following statements.

Compared with using a single joint distribution table:

- a belief net typically requires  × free parameters;
- a belief net typically requires  × training data to estimate the probability values;
- a belief net has  × (random) variables;
- running inference by enumeration on a belief net takes  × time (assuming that the time required to compute `joint_prob(network, assignment)` is negligible).

**Information**

## Representation of belief networks in Python

A belief (or Bayesian) network is represented by a dictionary. The keys are the names of variables. The values are dictionaries themselves. The second level dictionaries have two keys: 'Parents' whose value is a list of the names of the variables that are the parents of the current variable, and 'CPT' whose value is a dictionary again. The keys of the third level dictionaries are tuples of Booleans which correspond to possible assignments of values to the parents of the current node (in the order they are listed) and the values are real numbers representing the probability of the current node being true given the specified assignment to the parents.

### Notes

- Variable names are case sensitive.
- If a node does not have any parents, the value of 'Parents' must be an empty list and the only key of the third level dictionary is the empty tuple.
- For simplicity, we assume that all the variables are Boolean.

### Example

The following is the representation of the *alarm network* presented in the lecture notes.

```
network = {
  'Burglary': {
    'Parents': [],
    'CPT': {
      (): 0.001,
    }
  },

  'Earthquake': {
    'Parents': [],
    'CPT': {
      (): 0.002,
    }
  },

  'Alarm': {
    'Parents': ['Burglary', 'Earthquake'],
    'CPT': {
      (True, True): 0.95,
      (True, False): 0.94,
      (False, True): 0.29,
      (False, False): 0.001,
    }
  },

  'John': {
    'Parents': ['Alarm'],
    'CPT': {
      (True,): 0.9,
      (False,): 0.05,
    }
  },

  'Mary': {
    'Parents': ['Alarm'],
    'CPT': {
      (True,): 0.7,
      (False,): 0.01,
    }
  },
}
```

Mark 0.00 out of 6.00



**Question 13**Partially  
correctMark 0.87 out  
of 6.00

Suppose we want to predict the value of variable Y based on the values of variables X1, and X2. Assuming that we want to use a Naive Bayes model for this purpose, create a belief net for the model called network. The probabilities must be learnt by using the dataset given below. Use Laplacian smoothing with a pseudocount of 2.

X1	X2	Y
T	F	F
F	F	F
F	T	F
F	F	T
F	F	T
F	F	T
F	T	T

**Notes**

- Node names are case sensitive.
- Since we are using Python syntax, you can use fraction expressions if you wish.

**Answer:** (penalty regime: 20, 40, ... %)

```

1 network = {
2     'x1': {
3         'Parents': ['Y'],
4         'CPT': {
5             (True,): (0+2)/(4+4),
6             (False,): (1+2)/(3+4),
7         }
8     },
9
10    'x2': {
11        'Parents': ['Y'],
12        'CPT': {
13            (True,): (1+2)/(4+4),
14            (False,): (1+2)/(3+4)
15        }
16    },
17
18    'Y': {
19        'Parents': [],
20        'CPT': {
21            (): (4+2)/(7+4),
22        }
23    },
24 }
```

	Test	Expected	Go
✓		OK	OK

	<pre> from student_answer import network from numbers import Number  # Checking the overall type- correctness of the network # without checking anything question-specific  assert type(network) is dict for node_name, node_info in network.items():     assert type(node_name) is str     assert type(node_info) is dict     assert set(node_info.keys()) == {'Parents', 'CPT'}     assert type(node_info['Parents']) is list     assert all(type(s) is str for s in node_info['Parents'])     for assignment, prob in node_info['CPT'].items():         assert type(assignment) is tuple         assert isinstance(prob, Number)  print("OK") </pre>		
✖	<pre> from student_answer import network  print(sorted(network.keys())) </pre>	<pre> ['X1', 'X2', 'Y'] </pre>	<pre> ['\ 'x: 'x: </pre>

Testing was aborted due to error.

Show differences

Partially correct

Marks for this submission: 1.09/6.00. Accounting for previous tries, this gives **0.87/6.00**.

#### Question 14

Not answered

Mark 0.00 out of 7.00

We have a collection of observations where each observation is a temperature reading at a given time. Each observation is a pair (tuple) of the form (*time*, *temperature*). The observations do not uniformly cover the time axis - for some times there is no observation and for some times there are more than one observation. You have to write a function that given a time, it estimate the temperature by looking at neighbouring observations.

Write a function `estimate(time, observations, k)` that takes a time value and a collection of observations and produces a temperature estimate by averaging the  $k$  nearest neighbours of the given time. If after selecting  $k$  nearest neighbours, the distance to the farthest selected neighbour and the distance to the nearest unselected neighbour are the same, more neighbours must be

selected until these two distances become different or all the examples are selected. The description of the parameters of the function are as the following:

- **time:** a number (positive or negative; integer or floating point). The origin of time and its unit do not matter to us.
- **observations:** a collection of pairs. In each pair the first element is a time value (a number) and the second element is a temperature which is also a number. The collection might be a list, a set, or other Python collections. Do not make any assumption about the uniqueness of observations. Multiple (possibly different) temperature readings may be available for the same time.
- **k:** a positive integer which is the number of nearest neighbours to be selected. If there is a tie more neighbours will be selected (see the description above). If k is larger than the number of observations, then the algorithm must use as many neighbours as possible.

**For example:**

Test	Result
<pre>from student_answer import estimate  observations = [     (-1, 1),     (0, 0),     (-1, 1),     (5, 6),     (2, 0),     (2, 3), ]  for time in [-1, 1, 3, 3.5, 6]:     print(estimate(time, observations, 2))</pre>	<pre>1.0 1.0 1.5 3.0 3.0</pre>

**Answer:** (penalty regime: 0, 15, ... %)

1 ||

**Question 15**

Not answered

Marked out of  
6.00

Consider a binary classification problem (i.e. there are two classes in the domain) where each object is represented by 2 numeric values (2 features). We are using a single perceptron as a classifier for this domain and want to learn its parameters. The weight update rule is  $w \leftarrow w + \eta x(t - y)$ . We use the following configuration.

```
weights = [1, 1]
bias = 1
learning_rate = 0.5
examples = [
    ([1, -1], 0), # index 0 (first example)
    ([1, 4], 0),
    ([-1, -1], 1),
    ([3, 4], 1),
]
```

Answer the following with numeric values. Do not use fractions.

- After seeing the example at index 0, the value of the weight vector is [  × ,  × ] and the value of bias is  × .
- After seeing the example at index 1, the value of the weight vector is [  × ,  × ] and the value of bias is  × .
- After seeing the example at index 2, the value of the weight vector is [  × ,  × ] and the value of bias is  × .

**Question 16**

Not answered

Mark 0.00 out of 6.00

Consider the following explicit game tree.

```
[0, [-1, 1], [1, -1, 1]]
```

Assuming that the player at the root of the tree is Min, prune the tree (if necessary). Provide two variables: `pruned_tree` which is the pruned game tree and `pruning_events` which is a list of pairs of alpha and beta, one for each time a pruning event was triggered.

For example:

**Test**

```
import student_answer

check_it_is_a_gametree(student_answer.pruned_tree)
check_it_is_a_pruning_events_list(student_answer.p
```

**Answer:** (penalty regime: 25, 50 %)

Reset answer

```
1 from math import inf
2
3 pruned_tree = # COMPLETE
4
5
6 pruning_events = [
7     # (alpha, beta),
8 ]
```

**Question 17**

Not answered

Marked out of  
5.00

In the context of game-playing agents, game trees, and the related algorithms, answer this question.

Select one or more:

- ☐ Pruning game trees is a way of reducing the search space.
- ☐ One way to reduce the maximum depth of search is to use heuristic evaluation instead of actual utility.
- ☐ After alpha-beta pruning, assuming that some pruning takes place, the (utility) value of the tree improves.
- ☐ Reordering of subtrees in a game tree can change the amount of pruning.
- ☐ Reordering of subtrees in a game tree can change the (utility) value of the tree (for either Max or Min).
- ☐ The time complexity of alpha-beta pruning is at most as large as the complexity of computing the utility value of the root of the tree.

Your answer is incorrect.

**Question 18**

Not answered

Not graded

This is not an actual exam question and carries no marks. You can ignore any warning that indicates this "question" is incomplete; you can safely leave this empty. However, if you have any comments that the examiners must be aware of, use the text area below.