Started on	Tuesday, 31 October 2023, 9:15 PM
State	Finished
Completed on	Tuesday, 31 October 2023, 9:33 PM
Time taken	18 mins 17 secs

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

Correct

Marked out of 4.00

Given the following graph, trace the frontier of a depth-first search (DFS) without 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, ... %)

```
+a
+b
-b
+bc
-bc
+bca
+bcg
-bcg
```

+a (OK)
+b (OK)
-b (OK)
+bc (OK)
-bc (OK)
+bca (OK)
+bcg (OK)
-bcg (OK)

Passed all tests! ✔

Question 2

Not answered

Marked out of 6.00

Given the following graph, trace the frontier of an A-Star search with multiple-path pruning.

```
ExplicitGraph(
    nodes={'H', 'E', 'D', 'A'},
    estimates={'H':4, 'E':3, 'D':2, 'A':0},
    edge_list=[('H','E',2), ('H','D',5), ('E','H',2), ('E','D',2), ('D','A',3)],
    starting_nodes=['H'],
    goal_nodes={'A'},
    )
```

Notes

- This is a cost-sensitive search. All the trace lines must have a comma followed by a number.
- If pruning is necessary, finish the line with an exclamation mark.
- Priority queues must be stable.

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

```
1
```

Question 3

Correct

Marked out of 5.00

Select the best matching **graph search algorithm** for each of the following descriptions. You can assume that when needed, an appropriate heuristic function will be used. You can ignore details such as cycles, pruning, existence of a solution, etc.

Note: some items in the drop-down list may be used more than once or not used at all.

- It does not use information about the goal but guarantees to find a solution with the lowest number of arcs. — BFS ✔
- It does not use a lot of memory but does not guarantee finding an optimal solution. — DFS ✔
- It uses information about the goal, expands fewer nodes than other algorithms, and guarantees to find a solution with the lowest cost. — A* ✔
- It does not use information about the goal but guarantees to find a solution with the lowest cost to a goal state. — LCFS ✔
- It is guided by the goal but does not guarantee finding an optimal solution. — Best first search ✔
- It does not use a lot of memory but guarantees to find an optimal solution. — Iterative deepening search ✔

Information

Reading knowledge bases

In the following programming questions, your program will need to read in a knowledge base in the form of a string and perform automatic inference. If you wish you can use the following generator function to read the knowledge base.

```
import re

def clauses(knowledge_base):
    """Takes the string of a knowledge base; returns an iterator for pairs
    of (head, body) for propositional definite clauses in the
    knowledge base. Atoms are returned as strings. The head is an atom
    and the body is a (possibly empty) list of atoms.

    -- Kourosh Neshatian - 2 Aug 2021

    """
    ATOM   = r"[a-z][a-zA-Z\d_]*"
    HEAD   = rf"\s*(?P<HEAD>{ATOM})\s*"
    BODY   = rf"\s*(?P<BODY>{ATOM}\s*(,\s*{ATOM}\s*)*)\s*"
    CLAUSE = rf"{HEAD}(:-{BODY})?\."
    KB     = rf"^({CLAUSE})*\s*$"

    assert re.match(KB, knowledge_base)

    for mo in re.finditer(CLAUSE, knowledge_base):
        yield mo.group('HEAD'), re.findall(ATOM, mo.group('BODY') or "")
```

You can use `list(clauses(a_knowledge_base_str))` to see what it returns and whether it is useful to you.

Please note that the function is *not* provided on the server. If you decide to use this function, include it in your code.

▶ [Note: If you are working with older versions of Python 3 at home you can click here to use the following which works for Python 3.5 and below.

Question 4

Correct

Marked out of 7.00

Write a function `forward_deduce` that takes the string of a knowledge base containing propositional definite clauses and returns a (complete) set of atoms (strings) that can be derived (to be true) from the knowledge base.

Note: remember to include the `clauses` generator function (and `import re`) if necessary.

For example:

Test	Result

`from student_answer import forward_deduce`
 	
`kb = """`
`a :- b.`
`b.`
`"""`
 	
`print(", ".join(sorted(forward_deduce(kb))))`
	`a, b`

`from student_answer import forward_deduce`
 	
`kb = """`
`good_programmer :- correct_code.`
`correct_code :- good_programmer.`
`"""`
 	
`print(", ".join(sorted(forward_deduce(kb))))`	

`from student_answer import forward_deduce`
 	
`kb = """`
`a :- b, c.`
`b :- d, e.`
`b :- g, e.`
`c :- e.`
`d.`
`e.`
`f :- a,`
`g.`
`"""`
 	
`print(", ".join(sorted(forward_deduce(kb))))`
	`a, b, c, d, e`

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

```
import re

def clauses(knowledge_base):
    """Takes the string of a knowledge base; returns an iterator for pairs
    of (head, body) for propositional definite clauses in the
    knowledge base. Atoms are returned as strings. The head is an atom
    and the body is a (possibly empty) list of atoms.

    -- Kourosh Neshatian - 2 Aug 2021

    """
    ATOM   = r"[a-z][a-zA-Z\d_]*"
    HEAD   = rf"\s*(?P<HEAD>{ATOM})\s*"
    BODY   = rf"\s*(?P<BODY>{ATOM}\s*(,\s*{ATOM}\s*)*)\s*"
    CLAUSE = rf"{HEAD}(:-{BODY})?\."
    KB     = rf"^({CLAUSE})*\s*$"

    assert re.match(KB, knowledge_base)

    for mo in re.finditer(CLAUSE, knowledge_base):
        yield mo.group('HEAD'), re.findall(ATOM, mo.group('BODY') or "")

def forward_deduce(kb):
    head_bodys = list(clauses(kb))
    ans = set()
    should_stop = False
    while not should_stop:
```

```
        prev_len = len(ans)
        for head, body in head_bodys:
            if head not in ans:
                body_all_true = True
                for i in body:
                    if i not in ans:
                        body_all_true = False
                if body_all_true:
                    ans.add(head)
        if prev_len == len(ans):
            should_stop = True
    return ans
```

	Test	Expected	Got	

✔	`from student_answer import forward_deduce`
	
	`kb = """`
	`a :- b.`
	`b.`
	`"""`
	
	`print(", ".join(sorted(forward_deduce(kb))))`
		`a, b`	`a, b`	✔

✔	`from student_answer import forward_deduce`
	
	`kb = """`
	`good_programmer :- correct_code.`
	`correct_code :- good_programmer.`
	`"""`
	
	`print(", ".join(sorted(forward_deduce(kb))))`
				✔

✔	`from student_answer import forward_deduce`
	
	`kb = """`
	`a :- b, c.`
	`b :- d, e.`
	`b :- g, e.`
	`c :- e.`
	`d.`
	`e.`
	`f :- a,`
	`g.`
	`"""`
	
	`print(", ".join(sorted(forward_deduce(kb))))`
		`a, b, c, d, e`	`a, b, c, d, e`	✔

✔	`from student_answer import forward_deduce`
	
	`kb = ""`
	`print(", ".join(sorted(forward_deduce(kb))))`
				✔

Passed all tests! ✔

Question 5

Correct

Marked out of 5.00

Suppose we represent a directed acyclic graph (DAG) by a collection of facts of the form `edge(tail, head)` which indicates there is a directed edge from `tail` to `head`. Write a predicate `there_is_a_path(?Source, ?Destination)` that succeeds if there is a path from `Source` to `Destination`.

For example:

Test	Result

`edge(a, b).`
`edge(b, c).`
 	
`test_answer :- there_is_a_path(a, c),`
`writeln('OK').`
	`OK`

`edge(a, b).`
`edge(b, c).`
`edge(c, d).`
`edge(e, d).`
 	
`test_answer :- there_is_a_path(a, X),`
`there_is_a_path(e, X),`
`writeln(X).`
	`d`

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

```
there_is_a_path(S, D) :- edge(S, D).
there_is_a_path(S, D) :- edge(S, M), there_is_a_path(M, D).
there_is_a_path(S, S).
```

	Test	Expected	Got	

✔	`edge(a, b).`
	`edge(b, c).`
	
	`test_answer :- there_is_a_path(a, c),`
	`writeln('OK').`
		`OK`	`OK`	✔

✔	`edge(a, b).`
	`edge(b, c).`
	`edge(c, d).`
	`edge(e, d).`
	
	`test_answer :- there_is_a_path(a, X),`
	`there_is_a_path(e, X),`
	`writeln(X).`
		`d`	`d`	✔

	Test	Expected	Got	

✔	`edge(b, c).`
	
	`test_answer :-`
	`  there_is_a_path(a, a),`
	`  writeln('OK').`
		OK	OK	✔

Passed all tests! ✔

Question 6

Correct

Marked out of 5.00

Suppose we represent a path (walk) in a graph as a non-empty list of vertices that the path, in order, passes through. Write a predicate `has_cycle(+Path)` that takes a path (list) and succeeds if the path has a cycle (that is, it goes through a vertex more than once).

For example:

Test	Result

`test_answer :-`
`  has_cycle([a,b,c,d,a,e,f]),`
`  writeln('OK').`
 	
`test_answer :-`
`  writeln('Wrong!').`
	OK

`test_answer :-`
`  \+ has_cycle([a,b,c,d,e,f]),`
`  writeln('OK').`
 	
`test_answer :-`
`  writeln('Wrong!').`
	OK

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

```
not_in(_, []).
not_in(X, [Y | T]) :- X\=Y, not_in(X, T).

no_cycle([H]).
no_cycle([H | Path]) :- not_in(H, Path), no_cycle(Path).

has_cycle(Path) :- \+no_cycle(Path).
```

	Test	Expected	Got	

✔	`test_answer :-`
	`  has_cycle([a,b,c,d,a,e,f]),`
	`  writeln('OK').`
	
	`test_answer :-`
	`  writeln('Wrong!').`
		OK	OK	✔

✔	`test_answer :-`
	`  \+ has_cycle([a,b,c,d,e,f]),`
	`  writeln('OK').`
	
	`test_answer :-`
	`  writeln('Wrong!').`
		OK	OK	✔

Passed all tests! ✔

Question 7

Not answered

Marked out of 5.00

Write a predicate `leaves(+Tree, ?Leaves)` which succeeds when the first argument is a tree and the second argument is the list of the leaf nodes in the tree from left to right.

A tree is a term of the form `tree([child1, child2, ..., childn])`. The list of children is non-empty. Each child is either a tree term (which means a subtree) or an atom (which means a leaf node). For example the term `tree([tree([e]), b, tree([d, f])])` is a tree with the following shape.

```
   .
  /|\
 / | \
.  b  .
|    / \
|   /   \
e  d     f
```

The leaves of this tree from left to right are `[e, b, d, f]`.

Notes

- A non-leaf node (tree term) has one or more children.
- Trees may be arbitrarily deep.
- You can assume that the first argument will always be supplied with a valid tree.
- You can use the built-in predicate `atom(+Term)` which succeeds if Term is an atom. For example `atom(a)` succeeds and `atom(a())` fails.

For example:

Test	Result
`test_answer :- leaves(tree([the_only_leaf]), Leaves), writeln(Leaves).`	`[the_only_leaf]`
`test_answer :- leaves(tree([a, tree([b,c,d])]), L), writeln(L).`	`[a,b,c,d]`

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

```
1
```

Information

An instance of the constraint satisfaction problem can be specified with a set of variables, their domains, and a set of constraints. Here we use the class `CSP` which is available in the csp module. The class has two attributes as the following:

- `var_domains`: this is a dictionary of items of the form *var:domain* where *var* is a variable name (a string) and *domain* is a set of values that can be taken by the variable.
- `constraints`: a set of functions or lambda expressions (anonymous functions) such that each function takes a subset of variables, evaluates a condition, and returns true or false accordingly.

Question 8

Partially correct

Marked out of 5.00

Consider the following instance of the CSP problem and its corresponding constraint network.

```
canterbury_colouring = CSP(
    var_domains={
        'christchurch': {'red', 'green'},
        'selwyn': {'red', 'green'},
        'waimakariri': {'red', 'green'},
        },
    constraints={
        lambda christchurch, selwyn, waimakariri: len({christchurch, selwyn, waimakariri}) == 3
        })
```

Answer the following questions with integers using only numerals. Do not use words or expressions.

1. How many arcs does the network have? 3 ✔
2. Suppose we apply the *General Arc Consistency* algorithm to this network. After the network has been made arc-consistent, what is the size (number of elements) of the domain of the following variables?
   - *christchurch* 2 ✖
   - *selwyn* 2 ✖
   - *waimakariri* 2 ✖
3. Suppose during the execution of the *General Arc Consistency* algorithm, when an arc is being processed, the domain of its variable is changed. What is the maximum number of arcs that will be added to the to-do list again as a result of this change? ✖ [If the number varies depending on which arc has been processed (made consistent), write the largest number here.]

Question 9

Not answered

Marked out of 3.00

After making an instance of the CSP problem arc consistent, we are left with the following domains:

```
var_domains={
  'a': {1, 2},
  'b': {3},
  'c': {4, 5},
}
```

Which one of the following more accurately describes the number of solutions for this problem (considering that we cannot see the constraints):

Select one:

- Exactly 5.
- Any number from 0 to 5 (inclusive).
- Exactly 4.
- Any number from 0 to 4 (inclusive).
- Exactly 2.
- Any number from 1 to 4 (inclusive).
- No solution.
- Any number from 1 to 5 (inclusive).
- Exactly 1

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 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 10

Not answered

Marked out of 5.00

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 == -1,
      lambda b, c: b + c == 1,
      }
   )
```

For example:

Test	Result

`from student_answer import relations, relations_after_elimination`
`from csp import Relation`
 	
`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`
	
	`True`
	`True`

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

Reset answer

```
from csp import Relation

relations = [

      ### COMPLETE ###

      ]

relations_after_elimination = [

    ### COMPLETE ###

    ]

```

Question 11

Not answered

Marked out of 4.00

In the context of optimisation, determine whether the following statements are true or false. [Also see the notes below.]

- The greedy descent algorithms can get stuck in local minima. ✖
- A global search algorithm guarantees finding a globally optimal solution in finite time. ✖
- In the *n*-queens problem, the number of conflicts can be more than *n*. ✖
- A local search algorithm guarantees finding a globally optimal solution in finite time. ✖

Note

- When marking, wrong choices may receive some negative points. This is to discourage random answering.
- You have the option of not making any choices for some parts.
- If you want to leave some parts unanswered and also avoid automatic warning messages about unanswered parts in the question, you can choose "no answer" which will be treated exactly the same as not answering (no selection) when marking.

Question 12

Not answered

Marked out of 5.00

In the context of *Evolutionary Algorithms* answer the following questions. For each given description choose the best matching option.

Note: some items in the drop-down list may be used more than once or not used at all.

It is used for parent selection.	Choose...
It is useful for having non-decreasing maximum fitness over generations.	Choose...
Its role is to introduce new ideas into the search.	Choose...
Its evaluation can be computationally-intensive (e.g. require simulation).	Choose...
It creates new candidates by combining existing ones.	Choose...

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,
        }
    },
}
```

Question 13

Not answered

Marked out of 8.00

Consider the following Belief Network.

```
network = {
    'A': {
        'Parents': [],
        'CPT': {
            (): 0.4
            }},

    'B': {
        'Parents': ['A'],
        'CPT': {
            (True,): 0.1,
            (False,): 0.3,
            }},

    'C': {
        'Parents': ['A'],
        'CPT': {
            (True,): 0.2,
            (False,): 0.5,
            }},
}
```

Use the network to calculate the value of the following probabilities. Write the values with a precision of at least two digits after the decimal point (e.g. 0.12). Also see the notes below.

- *P(A=True)* = ✖
- *P(A=False)* = ✖
- *P(C=False | A=True)* = ✖
- *P(B=True | A=False, C=True)* = ✖
- *P(A=False, B=False, C=False)* = ✖
- *P(B=True | C=True)* = ✖

Notes

- Do not enter fractions, letters, or arithmetic. Only answer with a single number..
- In *P(X|Y)*, *Y* is the condition (evidence).
- Some values can be obtained very easily (little calculation).
- We do not recommend implementing an algorithm to answer this question. When needed, calculate the values manually using a calculator program or the Python shell as a calculator.
- When marking, there will be a tolerance of 0.01 for rounding (or truncation) error. For example, if a value is 0.1298, answers such as 0.12 and 0.13 will be treated as correct.

Question 14

Not answered

Marked out of 7.00

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 1.

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

Notes

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

For example:

Test	Result

`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")`
	`OK`

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

```
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 = 0
learning_rate = 0.5
examples = [
    ([-2, 0], 0),    # index 0 (first example)
    ([-1, 1], 0),
    ([1, 1], 0),
    ([2, 0], 1),
    ([1, -1], 1),
    ([-1, -1], 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

Marked out of 7.00

Write a function `knn_predict(input, examples, distance, combine, k)` that takes an input and predicts the output by combining the output of the *k* nearest neighbours. 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:

- `input`: an input object whose output must be predicted. Do not make any assumption about the type of input other than that it can be consumed by the distance function.
- `examples`: a collection of pairs. In each pair the first element is of type input and the second element is of type output.
- `distance`: a function that takes two objects and returns a non-negative number that is the distance between the two objects according to some metric.
- `combine`: a function that takes a set of outputs and combines them in order to derive a new prediction (output).
- `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).

Notes

- You only need to provide a single function `knn_predict` and the related import statements. You do not need to provide any *distance* or *combine* functions. These functions are defined in the test cases and are passed to your function as arguments.
- The `majority_element` function used in some test cases returns the smallest element when there is a tie. For example `majority_element('--++')` returns `'+'` because it is the most common label (like `-`) and in the character encoding system `'+'` comes before `'-'`.

For example:

Test	Result

`from student_answer import knn_predict`
 	
`examples = [`
`  ([2], '-'),`
`  ([3], '-'),`
`  ([5], '+'),`
`  ([8], '+'),`
`  ([9], '+'),`
`]`
 	
`distance = euclidean_distance`
`combine = majority_element`
 	
`for k in range(1, 6, 2):`
`  print("k =", k)`
`  print("x", "prediction")`
`  for x in range(0,10):`
`  print(x, knn_predict([x], examples, distance, combine, k))`
`  print()`
	`k = 1`
	`x prediction`
	`0 -`
	`1 -`
	`2 -`
	`3 -`
	`4 +`
	`5 +`
	`6 +`
	`7 +`
	`8 +`
	`9 +`
	
	`k = 3`
	`x prediction`
	`0 -`
	`1 -`
	`2 -`
	`3 -`
	`4 -`
	`5 +`
	`6 +`
	`7 +`
	`8 +`
	`9 +`
	
	`k = 5`
	`x prediction`
	`0 +`
	`1 +`
	`2 +`
	`3 +`
	`4 +`
	`5 +`
	`6 +`
	`7 +`
	`8 +`
	`9 +`

Test	Result

`from student_answer import knn_predict`
 	
`# using knn for predicting numeric values`
 	
`examples = [`
`    ([1], 5),`
`    ([2], -1),`
`    ([5], 1),`
`    ([7], 4),`
`    ([9], 8),`
`]`
 	
`def average(values):`
`    return sum(values) / len(values)`
 	
`distance = euclidean_distance`
`combine = average`
 	
`for k in range(1, 6, 2):`
`    print("k =", k)`
`    print("x", "prediction")`
`    for x in range(0,10):`
`        print("{} {:4.2f}".format(x, knn_predict([x], examples, distance, combine, k)))`
`    print()`
	`k = 1`
	`x prediction`
	`0 5.00`
	`1 5.00`
	`2 -1.00`
	`3 -1.00`
	`4 1.00`
	`5 1.00`
	`6 2.50`
	`7 4.00`
	`8 6.00`
	`9 8.00`
	
	`k = 3`
	`x prediction`
	`0 1.67`
	`1 1.67`
	`2 1.67`
	`3 1.67`
	`4 2.25`
	`5 1.33`
	`6 4.33`
	`7 4.33`
	`8 4.33`
	`9 4.33`
	
	`k = 5`
	`x prediction`
	`0 3.40`
	`1 3.40`
	`2 3.40`
	`3 3.40`
	`4 3.40`
	`5 3.40`
	`6 3.40`
	`7 3.40`
	`8 3.40`
	`9 3.40`

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

```
1
```

Question 17

Not answered

Marked out of 6.00

Consider the following explicit game tree.

```
[
    [[1, 2], [3, 4], [5, 6]],
    [1, 2, 3],
    [4, 5]
]
```

Assuming that the player at the root of the tree is Max, 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	Result

`import student_answer`
 	
`check_it_is_a_gametree(student_answer.pruned_tree)`
`check_it_is_a_pruning_events_list(student_answer.pruning_events)`
	OK
	OK

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

Reset answer

```
from math import inf

pruned_tree = # COMPLETE


pruning_events = [
    # (alpha, beta),
    ]
```

Question 18

Not answered

Marked out of 7.00

***Note**: We recommend you work on this question after you have attempted other questions.*

Consider a perfect-information zero-sum turn-based game played between two players called *Even* and *Odd*. The game is played on an array (list) of length *n*. Each position in the array is either filled with an integer or is a special symbol indicating that the position is blank. The players take turn to fill the blank positions with integers (one blank position at a time). In order to fill a blank position with an integer, all the following constraints must be met:

- Players are only allowed to use integers between *0* and *n-1* inclusive;
- *Odd* is only allowed to use odd numbers;
- *Even* is only allowed to use even numbers;
- a blank position can only be filled with a number that has not been used anywhere else in the array; and
- if a blank position is immediately after a number, the blank can only be filled with a greater number.

The game ends when a player cannot fill any blank space. If this is because no blank space is left, then it is a draw. If there are one or more blank spaces but the player (whose turn it is to play) cannot fill any of them while following the rules, then the player loses the game (and the other player wins).

Write a function `play(array, next_to_play)` that takes an array as the current state of the game and a player who has to make the next move and returns an array that is the result of the player's move and the final outcome of the game. The function must assume that both the players play optimally.

Input

- `array`: a (possibly empty) list. Each element of the list is either an integer or `None` which signifies a blank space.
- `next_to_play`: a string which is the name of the player whose turn it is to play. The string will be either `"Odd"` or `"Even"`.

Output

The function must return a pair (tuple of length two) of the form `(the_array_afterwards, final_winner)`

- The first element of the tuple, `the_array_afterwards`, will be the array (list) after the player has made her/his move. If no move is possible, it will be the original input array (list). If multiple moves (placement of numbers in the array) are equally valuable to a player, then among these moves, the player must choose one that uses the left-most blank space (smallest index) and (if there is still a tie) choose the smallest valid number to fill in the blank.
- The second elements of the tuple, `final_winner`, is a string that determines who will eventually win the game (assuming that the two players continue to play optimally). The string has to be either `"Even"` (when *Even* will win), `"Odd"` (when *Odd* will win), or `"no one"` (when there will be a draw).

Notes

1. The initial array is not necessarily the result of a game play; it can contain integers outside the interval of *0* and *n-1* and some integers may be repeated. However, after the game starts, Odd and Even must follow the above rules.
2. The length of the arrays used in the test cases is at most 6.
3. Considering the small size of the input in the test cases, you might be able to answer this without implementing pruning (assuming that you do not have unnecessary expensive operations in your implementation).

For example:

Test	Result

`from student_answer import play`
 	
`array = [None, None] # n = 2, both cells are empty`
`next_to_play = "Odd"`
 	
`# In this case Odd has one number (1). The best place to put it is`
`# the first cell because this blocks Even from playing her only number (0)`
`# in the second position. Therefore Even will lose this game.`
 	
`next_array, final_winner = play(array, next_to_play)`
`print("Odd plays:", next_array)`
`print("Final winner:", final_winner)`
	`Odd plays: [1, None]`
	`Final winner: Odd`

Test	Result

`from student_answer import play`
 	
`array = [None, None] # n = 2, both cells are empty`
`next_to_play = "Even"`
 	
`# In this case Even has one number (0). Both positions are equally`
`# good/bad therefore the earliest position is selected. After Even, it`
`# will be Odd's turn to play. He can place his only number (1) at the blank`
`# position. Next it will be Even's turn to play again but the array is full.`
`# Therefore it will be a draw.`
 	
`next_array, final_winner = play(array, next_to_play)`
`print("Even plays:", next_array)`
`print("Final winner:", final_winner)`
	`Even plays: [0, None]`
	`Final winner: no one`

`from student_answer import play`
 	
`array = [8, None, 6, None, 3, 8] # the initial array (which is not from a game play)`
`next_to_play = "Odd"`
 	
`# In this case Odd has 1 and 5 available to him but he cannot use either of`
`# them (because neither is greater than 8 or 6). He loses immediately.`
 	
`next_array, final_winner = play(array, next_to_play)`
`print("Even plays:", next_array)`
`print("Final winner:", final_winner)`
	`Even plays: [8, None, 6, None, 3, 8]`
	`Final winner: Even`

`from student_answer import play`
 	
`print(play([None, None, None], "Odd"))`
	`([1, None, None], 'Even')`

`from student_answer import play`
 	
`print(play([0,1,2,3], "Even"))`
	`([0, 1, 2, 3], 'no one')`

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

Reset answer

```
# Strings used in this problem:

"Odd"
"Even"
"no one"


```

Question 19

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.