PUZZLES AND BACKTRACKING

1. Puzzles

1.1. **Knights, Knaves and Spies.** In a Knights and Knaves puzzle you are given two statements from two people and it is your task to find out who is a knight (always speaks truly) and who is a knave (always lies). Two knights or two knaves is usually allowed, unless the puzzle says otherwise. The statements made should be logical in the sense that it should be possible to assign them a truth value of True or False. A self-referential statement like "I am a knave", or similar, is not allowed since it can be neither true or false.

There is a method for solving Knights and Knaves puzzles which always works. We illustrate the method with an example:

A: B is a knave
B: Neither of us are knaves

Let $a = 1$ if A is a knight and $a = 0$ if A is a knave. Similarly for b. Let $A(a, b)$ be the truth value of the statement "B is a knave" given the values of a and b. So for instance $A(1, 0) = 1$ since the statement "B is a knave" is true when $b = 0$. We have the following table.

a	b	$A(a,b)$	$B(a,b)$
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

The only line without a contradiction is line number 3. So the solution in this puzzle is that A is a knight and B is a knave. This method works even if the puzzle is not a valid puzzle (i.e. there is no solution, or more than one possible solution).

We could also have solved the puzzle without using the table by arguing as follows. Note unlike the method above, the argument below is based on the assumption that the puzzle has exactly one solution:

Assume A is a knave. Then the statement "B is a knave" is false, and so B is a knight. Then the statement "Neither of us are knaves" is true and so A must also be a knight. This is a contradiction and so A cannot be a knave. We may conclude that A is a knight. But then "B is a knave" must be true and so B is a knave.

With the addition of a Spy (a person that sometimes lies and sometimes tells the truth) there is usually exactly one in each category. We look at an example.

A: C is a knave.
B: A is a knight.
C: I am the spy.

We set up the table with the six possibilities. I use x to indicate spy.

a	b	c	A	B	C
0	1	x	0	0	1
1	0	x	0	1	1
0	x	1	0	0	0
1	x	0	1	1	0
x	1	0	1	0	0
x	0	1	0	0	0

As x indicates Spy you cannot use it to create a contradiction. However you see by the table that you have contradiction on all lines except line number 4. In other words, A is a knight, B is the spy and C is the knave. As before, this method also shows that this is a valid puzzle (i.e. there is exactly one solution).

An argument based on the assumption that the puzzle is valid:

"I am the spy" can only be said by a knave or a spy, so C is either a knave or a spy. If C is a spy, then "C is a knave" is false and so A is the knave. But then the statement "A is a knight" is also false, and B is also a knave. This is a contradiction since we cannot have two knaves. We can therefore conclude that C is a knave. Then the statement "C is a knave" is true and so A is the knight. Since they are all different B must be the knave.

2. Constraint satisfaction problems

A constrain satisfaction problem consist of a set of variables $x_1, \cdots, x_n$, a set of values $v_1, \cdots, \cdots v_m$ and a set of logical constraints expressing which value assignments are valid. A solution is a specific assignment which satisfies the constraint. A constraint satisfaction problem could have $0, 1$ or more solutions, and depending on the context we might be interested in listing all solutions, finding one solution, or deciding whether a solution exists.

Example 1. *Consider a sudoku game. Variables correspond to unknown entries and values are* $1, \cdots, 9$. *The constraints are that each value occur exactly once in each row, column and* 3×3 *block.*

Example 2. *Examples of "real-world" constraint satisfiaction problems.*

1. *Assignment problems (e.g. who teaches which class)*
2. *Timetabling problems (e.g. where and when should a class be taught)*
3. *Transportation, Scheduling etc.*

Approach 1. Generate and test

Order the domain (i.e. all possible assignments) and loop through until you find one assignment which satisfies the constraint.

Approach 2. Constraint propagation

Use the constraints to reduce the size of the domain ensuring that the smaller domain still include all the solutions to the original problem. If the smaller domain still contain invalid assignments, you can then solve the problem by for instance searching over this smaller domain.

In the sudoku example, you can for instance construct a domain such that assignments on any row, column or box are distinct. This search space is considerable smaller than the one in Approach 1.

Approach 2. Backtracking

Backtracking needs a stack to record states during the search. A stack is a data structure with two methods

```
push(item)   # puts an item on the stack
pop(item)    # removes an item from the stack in the order last in first out.
```

In python you can use lists as stacks, with append() instead of push().

Of course we can also use recursion and the built in call-stack. Recursively backtracking would look something like.

```
def Search(x(1),...,x(n),i):
    If i < n:
        For each possible value for x(i+1)
            If x(1),...,x(i+1) is not rejected by constraints:
                Search(x(1),...,x(n),i+1)
    else:
         Print the solution x(1),...,x(n)

Search(x(1),...,x(n),0)
```

Backtracking has the advantage over Approach 1 that large parts of the domain will never be considered.

Approach 3. Backtracking with constraint propagation

With pseudo-code the algorithm could look something like:

```
Search(x(1),...,x(n),i)
    If i < n:
        Compute a smaller domain D given the values of x(1),...,x(n)
        For each possible value for x(i+1) in D
            If x(1),...,x(n) is not rejected by constraints:
                Search(x(1),...,x(n),i+1)
    else:
         Print the solution x(1),...,x(n)

Search(x(1),...,x(n),0)
```

There is a trade-off between the computation of the domain D and the depth of the search-tree.

We look at an example using approach 3 to solving sudoku, this time without recursion. First lets pick a data structure for our problem. An unfinished puzzle is a list of 81 lists. There is one list for each tile on the sudoku game. The list at that tile is a sublist of $[1, 2, 3, 4, 5, 6, 7, 8, 9]$ indicating the possible values at that tile. At the start each tile is either a singleton [i], indicating a value for that tile or the full list $[1, \cdots, 9]$ of 9 elements, indicating that we know nothing about the values in that tile.

We will need a stack of puzzles, which we implement using a lists. The stack operations in Python are as follows:

```
puzzle = [originalPuzzle]   # stack with one puzzle
puzzle.push(p)              # push a puzzle onto the stack
p = puzzle.pop()            # pop a puzzle from the stack
p = puzzle[-1]              # look at element on top of stack,
                            # but don't pop
```

If at any time there is a singleton at any row, column and box we may remove that singleton from any of the lists at that row, column or box. We do this with the following function which takes an unfinished puzzle as input.

```
def computeDomain(p)
    while p is changing:
        for each singelton [a] in p:
            remove entries equal to a from lists in the same row as [a]
            remove entries equal to a from lists in the same column as [a]
            remove entries equal to a from lists in the same 3x3 box as [a]

    if p changed:
        return True
    else:
        return False
```

The function returns false if no changes were made to the puzzle. When returning one of three things has happened.

1. The puzzle is unchanged: we need to make an assumption on the value in some tile before continuing to solve.
1. The puzzle is a list of singletons: We have a solution
2. The puzzle contains an empty list: We have a contradiction and we know that there is no solution. We now pop from the stack.
3. The puzzle contains no empty lists and a list with more than one element: We need to make an assumption about the contents on one of the tiles and push it onto the stack.

Full pseudo (Python) code is as follows.

```
def computeDomain(p)
```

```
    while p is changing:
        for each singelton [a] in p:
            remove entries equal to a from lists in the same row as [a]
            remove entries equal to a from lists in the same column as [a]
            remove entries equal to a from lists in the same 3x3 box as [a]

    if p changed:
        return True
    else:
        return False

puzzle.append(originalpuzzle)

while not solved:
    changing = True
    while changing:
        changing = computeDomain(puzzle[-1])
        if puzzle[-1] contains an empty list:
            puzzle.pop()  # Backtracking
            changed = True
        elif not puzzle:  # stack is empty
            changing = False

   # At this stage one of three things can happen
   #
   # 1. puzzle is empty, which means there is no solution
   # 2. puzzle[-1] is a list of singelton lists [a] - a solution
   # 3. puzzle[-1] contains a list which is not a singelton
   #
    if not puzzle:  # stack is empty - no solution
        solved = True
        continue

    find an index x of a list in puzzle[-1] of length at least 2
    if no x can be found:  # Solution found
        solved = True
    else:  # We make an assignment and continue searching
        val = puzzle[-1][x].pop()
        puzzle.append(copy.deepcopy(puzzle[-1]))
        puzzle[-1][x] = [val]

puzzle[-1] now contains a solution to the original sudoku puzzle.
```

Exercises.

Exercise 1. (Skip if too hard)

Implement a sudoku solver in Python using Approach 4 above.

Exercise 2.

In the following knights and knave puzzles, determine which is which.

a)

A: "Me or B is a knight"
B: "A is a knave"

b)

A: "I am a knight or B is a knave"
B: "Exactly one of us is a knight"

d)

A: "B is a knight or I am a knight"
B: "A would say that I am a knave"

Exercise 3.

a)

A: "Only a knave would say that B is a knave"
B: "It is false that C is a knave"
C: "B would say that I am a knave"

b)

A: "C is a knight or D is a knight"
B: "A and C are both knights"
C: "I and D are knights"
D: "C is a knave"

Exercise 4.

In the following knights, knaves and spies puzzles, find out if the puzzle is a valid puzzle and give the solution if it is. If the puzzle is invalid then explain why.

a)

A: I am a Knave
B: C is a Knave
C: B is a Knight

b)

A: B is a Knave
B: A is not a Knave
C: A is not a Knave

c)

A: I am a Knight
B: I am a Knave
C: B is not a Knave

d)

A: I am a Knight
B: I am not a Spy
C: I am a Spy

e)

A: I am a Knight
B: I am not a Spy
C: B is a Knight

f)

A: I am a Knight
B: I am not a Spy
C: B is not a Knave

Answers:

Exercise 1.

In the following knights and knave puzzles, determine which is which.

a)

Assuming the puzzle is valid.

Assume A is a knave. Then both A and B are knaves. But then B's statement is true, which is a contradiction. So we conclude that A is a knight. Then B's statement is false, and so B is a knave.

b)

Assuming the puzzle is valid:

Assume that B is a knight. Then the statement "Exactly one of us is a knight" is equivalent to "A is a knave". Then the statement "I am a knight or B is a knave" is false, which means "A is a knave" and "B is a knight".

Assume that B is a knave. Then the statement "Exactly one of us is a knight" is false. In other words, there are either two knights or two knaves. As B is a knave, we must have two knaves. The statement "I am a knight or B is a knave" is then false. That is, "I am a knave and B is a knight" must be true. This contradicts the assumption, and so we may conclue that B is a knight. Then B's statement is true, and so A is a knave.

d)

We make a table

a	b	A	B
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	0

We see from the table that both are knaves.

Exercise 3.

a)

We reformulate the statements.

A: "B is a knight"
B: "C is a knight"
C: "B says C is a knave"
make the table

a	b	c	A	B	C
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	0

and conclude that they are all knaves.

Another argument(assuming the puzzle is valid):

If A is a knight, then B is a knight, and therefore C is a knight. But C's statement is false and this is a contradiction

So A must be a knave. But then B is a knave by A's statement, and C is a knave by B's statement

b)

The puzzle can be solved by making a table (with 16 rows).

We try a different approach (assuming the puzzle is valid):

Assume C is a knight. Then D is a knight by C's statement, but D's statement is then false and this is a contradiction.

We can therefore conclude that C is a knave.

As the statement "C is a knave" is now true, we can conclude that D s a knight.

Assume B is a knight. Then C is a knight by B's statement. This is false and so we have a contradiction.

So we conclude that B is a knave.

The statement A makes is true since we know that D is a knight, and so finally A is the knight.

Exercise 4.

a)

The puzzle is invalid due to the non-logical statement "I am a knave". Relaxing the rules a little and allowing spies to say non-logical statements does not help as the following table shows.

a	b	c	A	B	C
0	1	S	1	0	1
1	0	S	0	0	0
0	S	1	1	0	0
1	S	0	0	1	0
S	0	1	0	0	0
S	1	0	0	1	1

There is a contradiction on every line.

b)

No solution as the following table shows:

a	b	c	A	B	C
0	1	S	0	0	0
1	0	S	1	1	1
0	S	1	0	0	0
1	S	0	0	1	1
S	0	1	1	1	1
S	1	0	0	1	1

Another argument:

If A is a knight, then B is a knave, and so "A is not a knave" is false. But then A is a not a knight. This is a contradiction.

If A is the spy, then "A is not a knave" is true, and so both B and C are knights which is a contradiction.

If A is the knave, then one of B and C is a knight and so the statement "A is not a knave" is true. Again a contradiction.

There is nothing wrong with the three statements made, they are all perfectly valid logical statements. There is however a contradiction with the hidden assumption "There is one of each type". What we have shown is that with these three statements, there cannot be one spy, one knave and one knight.

d)

There are two possibilities, so this is not a valid puzzle.

a	b	c	A	B	C
0	1	S	0	1	1
1	0	S	1	1	1
0	S	1	0	0	0
1	S	0	1	0	0
S	0	1	0	1	0
S	1	0	1	1	0

e)

Two possibilities for the solution. So this is not a valid puzzle.

a	b	c	A	B	C
0	1	S	0	1	1
1	0	S	1	1	0
0	S	1	0	0	0
1	S	0	1	0	0
S	0	1	0	1	0
S	1	0	0	1	1

f)

Two possibilities, so this is not a valid puzzle.