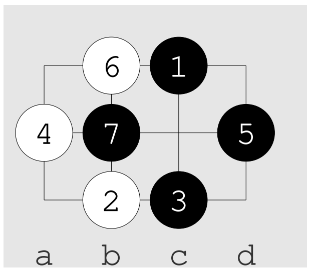
cmput 355 2025 practice quiz questions 1 (answer key)

learning outcomes (LO) For game/puzzle algorithm, you will learn to

1. analyze algorithm runtime and space usage with mathematical reasoning,
2. construct logical proofs to demonstrate algorithm correctness and behaviour,
3. explain the behaviour of algorithms, starting with a precise description of the rules,
4. propose potential improvements of algorithms,
5. implement algorithms using python3.

Unless we say otherwise, for go we assume **logical rules, no self-capture, komi 0**.

1. LO3. For both of the following continuations of the go game, give the first illegal move, and explain why it is illegal (occupied point, liberty violation, superko violation) or answer **all moves legal** if all moves are legal. (a) Assume self-capture is illegal. (b) Assume self-capture is legal.



(a) . . . 8.W[c2] 9.B[b2] 10.W[d1] **B[b2] superko violation**

(b) . . . (same as above) **B[b2] superko violation**

(a) . . . 8.W[a3] 9.B[pass] 10.W[a1] 11.B[c2] **W[a1] self-capture**

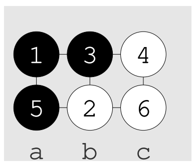
(b) . . . (same as above) **all moves legal**

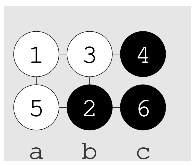
2. LO3. The diagrams show three sequences of go moves. For each sequence,

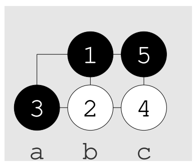
   (a) if some move is illegal, give the first illegal move and explain why it is illegal.

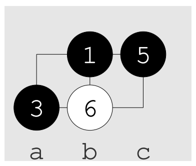
   (b) assume each player passed after the last legal move: give the final score.

   Answer like this: B 3, W 4.









Move 5 (B[a1]) self-capture violation. Score: B 2, W 3

Move 1 (W[a2]) white cannot play first. Score: B 0, W 0

All moves are legal. Score: B 4, W 1

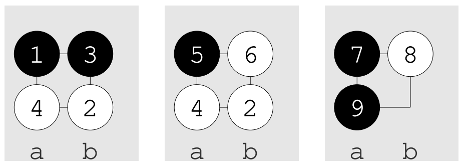
3. LO3. Repeat question 2 if we change the rules to allow self-capture.

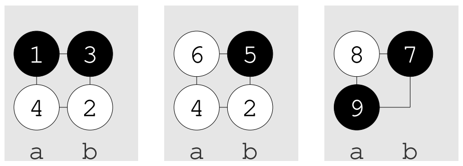
   All moves are legal. Score: B 0, W 6

   Move 1 (W[a2]) white cannot play first. Score: B 0, W 0

   All moves are legal. Score: B 4, W 1

4. LO3. Repeat questions 1 and 2 for the two sequences below.

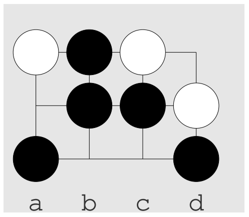




   Sequence 1: Move 7 violates superko. Score: B 0, W 4

   Sequence 2: All moves are legal. Score B 4, W 0

5. LO3. (a) For this go position, give all black blocks and all white blocks. Write each block as a set of points, e.g. {a1}. (b) For each block, give all liberties (again, as a set of points). (c) If this is the final position in the game, what is the final score? Answer like this: Black 32, White 17. (d) Repeat (c) for komi 3.5.



   a) B: {a1}, {b2, b3, c2}, {d1} W: {a3}, {c3}, {d2}

   b) B: {a2, b1}, {a2, b1, c1}, {c1} W: {a2}, {d3}, {d3}

   c) B 7, W 4

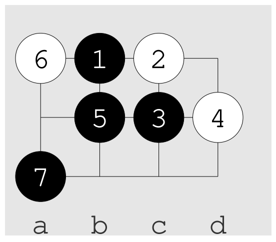
   d) B 7, W 7.5

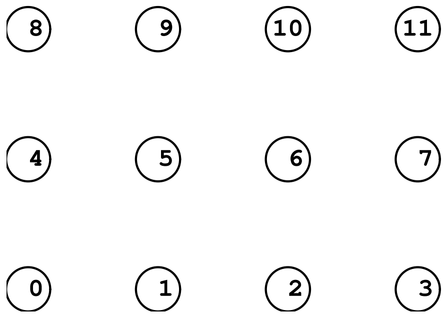
6. LO5. Give the missing code from `union`. Hint: see `gpa/hexgo/hexgo.py`.

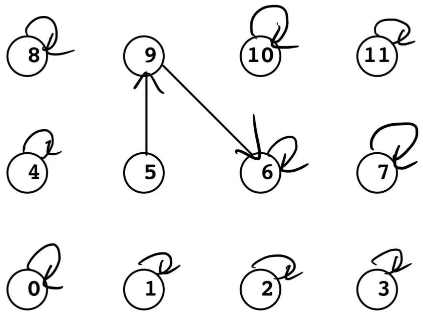
```
def union(parents, x, y):
    rootx = UF.find(parents, x)
    rooty = UF.find(parents, y)
    parents[rooty] = rootx # rootx is root of merged trees
    return rootx, rooty
```

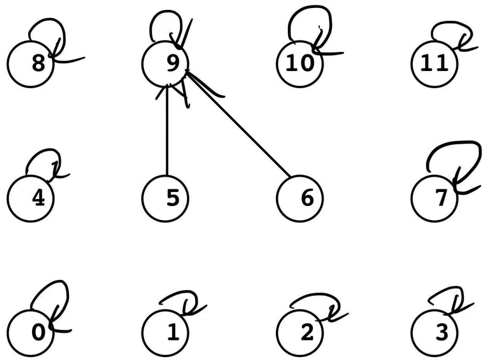
7. LO5. Assume go neighbour offsets are stored in this order: above, right, below, left.

   (a) For this go position, on the pointer diagram, draw the parent pointers. (b) Repeat the question if moves 3 and 5 are exchanged. (c) Why does the order of moves matter? (d) Why does the order of neighbour offsets matter?







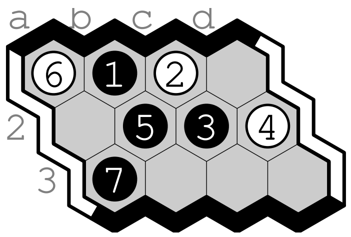


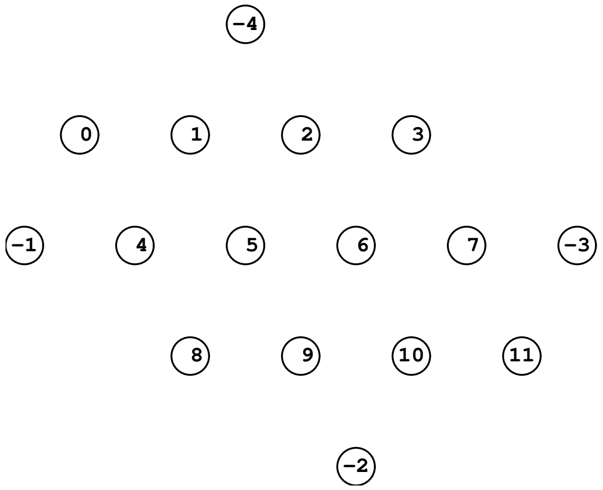
Solution: a) **b)**

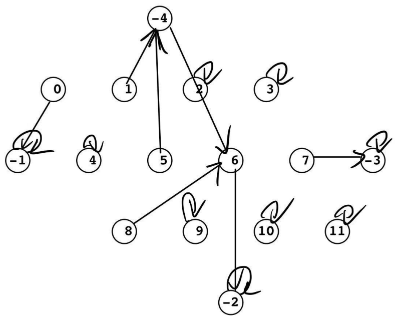
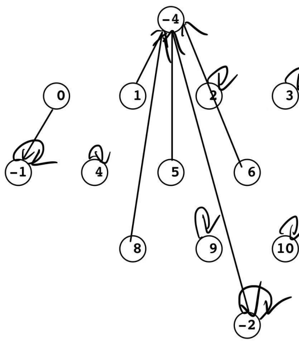
c) The order of the move decided which sets get merged first.

d) The order of neighbor offsets decided which union operation you perform first, which affects the root node.

8. LO5. Repeat the previous question for this hex position. Notice the change in the labeling of rows. Assume hex neighbour offsets are stored in this order: above left, above right, right, below right, below left.

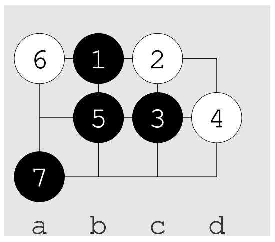


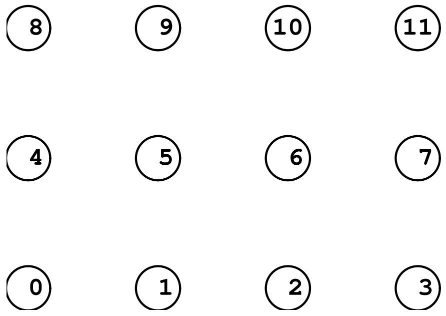


Solution: a) b)

9. LO3, LO5. (a) For github program `hexgo/hex.py`, after move 4, on the pointer diagram at each point that is a root of its block, give the set of liberties of that block. (b) Repeat (a) for 5.





(a) after move 4, root—>liberties:

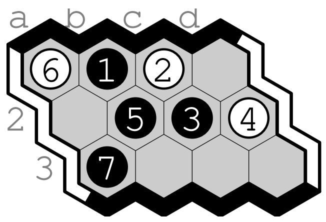
- **9—>{8, 5}**
- **6—>{5, 2}**

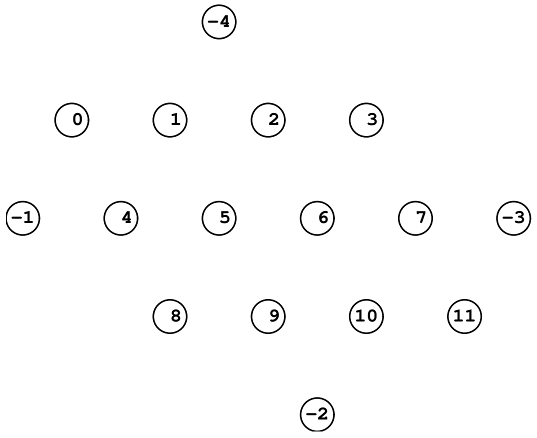
- **10—>{11}**
- **7—>{11, 3}**

(b) after move 5, root—>liberties:

- **6—>{1, 2, 4, 8}**
- **10—>{11}**
- **7—>{11, 3}**

10. LO3, LO5. Repeat the previous question for this hex position.





(a) after move 4, root->liberties:

- **-4->{0, 4, 5}**
- **6->{5, 3, 9, 10}**
- **2->{3, 5}**
- **-3->{3, 10, 11}**

(b) after move 5, root->liberties:

- **6->{0, 4, 8, 9, 10, 3}**
- **2->{3}**
- **7->{3, 10, 11}**

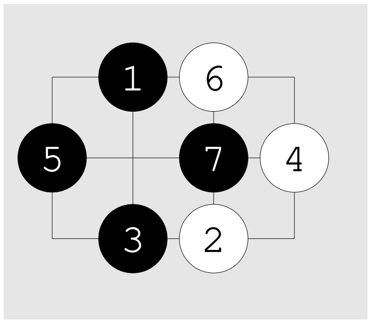
11. LO3. (a) In each of the following go games, give the first illegal move, and explain why it is illegal (occupied point, liberty violation, superko violation). Answer **all moves legal** if all moves are legal. (b) Repeat (a) if we allow self-capture.

. . . 8.W[b2] 9.B[c3] 10.W[d3]

(a) 9.B[c3] (occupied point)

(b) 9.B[c3] (occupied point)

. . . 8.W[b2] 9.B[a3] 10.W[a1]



(a) all moves legal

(b) all moves legal

. . . 8.W[a1] 9.B[a3] 10.W[a1]

(a) 8.W[a1] (liberty violation)

(b) all moves legal

. . . 8.W[b2] 9.B[d3] 10.W[a1]

(a) 9.B[d3] (liberty violation) (b) all moves legal

. . . 8.W[d1] 9.B[a1] 10.W[d3] 11.B[b2]

(a) 10.W[d3] (liberty violation)

(b) all moves legal

12. LO3, LO5. In the class github repo, in directory `hexgo`, execute `hex.py` . (a) List the hex moves that have been made in the game by `m33demo` (the first move is 1.B[b2]). (b) Which of the four lines of output tells you that black has won the game? Explain carefully.

    a) 1.B[b2] 2.W[a1] 3.B[b3] 4.W[c3] 5.B[b1]

    b) `!!! game over: black wins`

13. LO3, LO5. This line is from the hex board initialization in `hexgo/stone_board.py`:

    ```
    self.nbr_offset = ((-1,0),(-1,1),(0,1),(1,0),(1,-1),(0,-1))
    ```

    (a) Give the corresponding line for go board initialization:

    ```
    self.nbr_offset = ((-1,0),(0,1),(1,0),(0,-1))
    ```

    (b) Carefully explain the purpose of line 4 below:

    It prevents the code from checking for stones in cells that do not exist, due to r or c being on an edge.

    ```
    1  for r in range(self.r):
    2      for c in range(self.c):
    3          for (y,x) in self.nbr_offset:
    ```

```
4               if r+y in r_range and c+x in c_range:
                    ...
```

14. LO3, LO5. In the class github repo, in directory `hexgo`, in file `stone_board.py`: (a) explain the purpose of each line of `merge_blocks()`

```
1   def merge_blocks(self, p, q):
2       proot, qroot = UF.union(self.parents, p, q)
3       self.blocks[proot].update(self.blocks[qroot])
4       self.liberties[proot].update(self.liberties[qroot])
5       self.liberties[proot] -= self.blocks[proot]
```

Line 2 performs the union operation on the cells p and q, returning their representative roots.

Line 3 adds all elements in the q block set to the p block set. This is done to join the blocks.

Line 4 adds all the elements in the q liberties set to the p liberties set. This is done to combine the liberties.

Line 5 removes the liberties that now contain stones.

(b) explain the purpose of each line of `remove_liberties()` .

```
1   def remove_liberties(self, p, q):
2       proot = UF.find(self.parents, p)
3       qroot = UF.find(self.parents, q)
4       self.liberties[proot] -= self.blocks[qroot]
5       self.liberties[qroot] -= self.blocks[proot]
```

Line 2 finds the representative of p.

Line 3 finds the representative of q.

Line 4 removes the liberties of p that contain q stones.

Line 5 removes the liberties of q that contain p stones.

15. LO5. (a) Fill in the missing lines from the function that computes the go score below.

```
def tromp_taylor_score(self): # each player: stones, territory
    bs, bt, ws, wt, empty_seen = 0, 0, 0, 0, set()
    for p in range(self.guarded_n):
        if self.brd[p] == BLACK:
            bs += 1
        elif self.brd[p] == WHITE:
            ws += 1
        elif (self.brd[p] == EMPTY) and (p not in empty_seen):
            b_nbr, w_nbr = False, False
            empty_seen.add(p)
            empty_points = [p]
            territory = 1
            while (len(empty_points) > 0):
                q = empty_points.pop()
                for j in self.nbr_offsets:
                    x = j + q
                    b_nbr |= (self.brd[x] == BLACK)
                    w_nbr |= (self.brd[x] == WHITE)
                    if self.brd[x] == EMPTY and x not in empty_seen:
                        empty_seen.add(x)
                        empty_points.append(x)
                        territory += 1
            if b_nbr and not w_nbr:
                _______________________________________  # Fill in
            elif w_nbr and not b_nbr:
                _______________________________________  # Fill in
    return bs, bt, ws, wt
```

1). bt += territory

2). wt += territory

(b) Explain carefully: what does this line do ? `b_nbr |= (self.brd[x] == BLACK)`

(self.brd[x] == BLACK) determines whether point x is a black stone. By taking an 'or' (| =) of each neighbor x, b_nbr represents whether a point has any black stone neighbors.

16. LO3, LO5. Functions below are from `gpa/go/count_legal.py`. Explain how each works. If asked, you should be able give the expression after `return` if it is left blank.

```
def coord_to_point(r, c, C):
  return c + r*C
def point_to_coord(p, C):
  return divmod(p, C)
def point_to_alphanum(p, C):
  r, c = point_to_coord(p, C)
  return 'abcdefghj'[c] + '123456789'[r]
def change_str(s, where, what):
  return s[:where] + what + s[where+1:]
```

coord_to_point takes a pair of coordinates (r, c) and the number of columns in the board (C), and returns the index of the corresponding point in a 1d array representation of the board by multiplying the row coord (r) by the number of columns (C) and adding the column coord (c).

point_to_coord takes a point corresponding to a 1d array board representation and converts it to a row, column coordinate pair by dividing the point by the number of columns. The quotient is the row coord and the remainder is the column coord.

point_to_alphanum converts a point to the corresponding alphanumeric move notation by indexing row and column notation lists.

change_str inserts 'what' into the string s at index 'where'.

17. L03, L05. (a) If necessary, reorder A,B,C,D in this go pseudocode. (b) Explain how to modify the code if self-capture is allowed.

```
def is_legal(c, psn, clr, hist):
   if not is_empty(c): return (False, 'cell occupied', None)
   new_psn = add_stone(c, psn, clr)
A. in new_psn, for each c-neighbour q w. color oppt(clr):
     if not has_liberty(q, new_psn): remove block(q)
B. if new_psn in history: return(False, 'superko', None)
C. in new_psn, for each c-neighbour p w. color clr:
     merge block(c) and block(p)
   if not has_liberty(c, new_psn): return (False, 'self-capture', None)
   return (True, '', new_psn)
D. if not has_liberty(c, new_psn): return (False, 'no liberty', None)
```

a) A B D C

b)first, remove D, then in the third line of C, if c has no liberty, we remove the block instead of return

18. LO1, LO2. (a) Explain why there are 3 possible 1x1 go positions.

In Go, each intersection must be Black, White, or Empty (3 options). A 1x1 board has exactly one intersection, which can be any of the 3 possibilities.

(b) Explain why only 1 of the positions in (a) is legal.

The only position that is legal is Empty. When considering the rules in go, each stone or block of stones must have at least 1 liberty. If a Black or White stone is placed on a board with a single intersection, there are no possible liberties that can surround the stone. Therefore, only when the board is empty can the position be legal.

(c) Explain why the only legal 1x1 go game is 1.B[pass] 2.W[pass].

The inital game state of a legal 1x1 go game is Empty. Since Black goes first, the only possible position to avoid having no liberties is to pass. The turn will then go to White, which also can only pass on their move. Following 2 sebsequent passes, the game will immediately end.

(d) For the $n \times n$ board, give the number of 3-color (black, white, empty) positions.

For a n x n board, each position can be 3 different states. There are n^2 intersections. Therefore, the number of 3-color positions is 3^{n^2}.

(f) Roughly what fraction of all 3×3 go positions are legal: .005? .05? .5?

Run the script and view the output. For 3x3 board: 12675 legal positions out of 19683. 12675/19683 = 0.64. Approximately 0.5 of all go positions in 3x3 are legal.

Hint: `gpa/go/count_legal.py`.

19. LO4. Explain the significance of each of these moves in the DeepMind challenge match between Lee Sedol and AlphaGo.

    (a) game 1: move 7 **AlphaGo played an unconventional move. Lee Sedol was unsettled by this move as it did not match human conventions on how to play the game. It set the precent on the ability for AlphaGo to innovate creative solutions.** (b) game 3: move 37

    Move 37 played by AlphaGo in response from a strong attack by Lee Sedol. The move was surprising as it made itself vulnerable to attack with many believing it was a mistake that a human would not make. However, this move was actually the start of a long term strategy by AlphaGo to counter attack. Shows the originality of AlphaGo that a human would not be able to compute.

    (c) game 4: move 78

    One of the most famous moves from Lee Sedol, also known as the Hand of God. He played an unconventional and risky move that exploited a position within AlphaGo's territory. AlphaGo was unable to find a way to counteract this move, and Lee Sedol won the match. It shows that while AlphaGo was very powerful, there were still areas that could be exploited and also the perseverance of human creativity.

20. LO5. In file `stone_board.py` in directory `hexgo` in class github repo

    `https://github.com/ryanbhayward/games-puzzles-algorithms` :

    (a) explain carefully the purpose of this line and how it is used:

```
self.nbr_offset = ((-1,0),(0,1),(1,0),(0,-1))
```

The purpose of this line is to define the neighbor's offset of position on a board (such as go). It is to represent all the possible movement directions that can move the relative position of a point on the graph:

$$(-1, 0) \text{ moves up}$$

$$(0, 1) \text{ moves right}$$

$$(1, 0) \text{ moves down}$$

$$(0, -1) \text{ moves left}$$

It is used to calculate the neighbors of a point on the graph by applying the relative offsets.

(b) repeat (a) for this line

```
self.nbr_offset = ((-1,0),(-1,1),(0,1),(1,0),(1,-1),(0,-1))
```

The purpose of this line is to define the neighbor's offset of position on hexagonal grid (such as Hex). It is to represent all the possible movement six directions that can move the relative position of a point on the graph. It is used to calculate the neighbors of a point on the graph by applying the relative offsets.

(c) explain carefully the purpose of this line:

```
if r+y in r_range and c+x in c_range:
```

This line is used to validate that the neighboring coordinates (r + y) and (c + x) are within the range of row and column indices, and not out of bounds of the board. This is to ensure that only valid neighbors are checked.