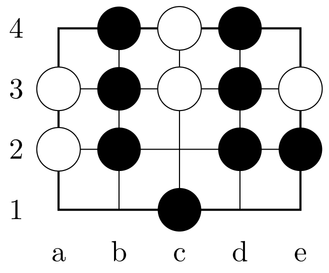
1. Consider the following 4 × 5 Go position, with White stones at `a2`, `a3`, `c3`, `c4`, `e3` and Black stones at `b2`, `b3`, `b4`, `c1`, `d2`, `d3`, `d4`, `e2`.



   (a) Compute the Tromp-Taylor Score for each player.
   Black has 8 stones, and 2 points of territory that reach only Black (d1 and e1), for a total of 10 points. White has 5 stones, and no territory (all empty points reach Black), for a total of 5 points.

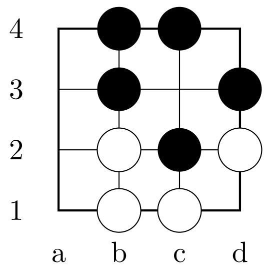
   (b) Can Black make a move that captures at least one White stone? Give coordinates or explain why not.
   Black can capture at least one White stone by playing at either e4 or c2.

   (c) Can White make a move that captures at least one Black stone? Give coordinates or explain why not. No. Every Black group has at least two liberties.

   (d) List all of Black's blocks. Black has three blocks: {`c1`}, {`b2,b3,b4`}, and {`e2,d2,d3,d4`}.

   (e) List all of White's blocks. hite has three blocks: {`a2,a3`}, {`c3,c4`}, and {`e3`}.

2. Consider the following 4 × 4 Go position after 7 moves, with White stones at `b1`, `b2`, `c1`, `d2` and Black stones at `b3`, `b4`, `c2`, `c4`, `d3`.



   (a) List the liberties for the block containing `c4`. a3, a4, c3, d4

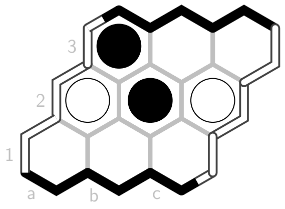
   (b) List the liberties for the block containing `c2`. c3

   (c) Suppose that White places a stone at `c3`. What stones, if any, does it capture? c2

   (d) Suppose that Black replies by playing a stone at `c2`. Is this a legal move? Why or why not? No, this move is forbidden by superko. (Also forbidden by basic ko, any answer referencing ko is acceptable)

   (e) Would it be legal for Black to reply by playing at `d1` instead? Why or why not? No, this move would be suicide. The stone at d2 has two liberties after White's capture of c2, so Black's move on d1 would not capture any stones, and would not have any liberties.

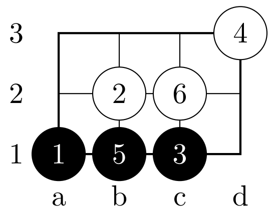
3. Consider the following 3 × 3 Hex board with White stones at `a2`, `c2` and Black stones at `a3`, and `b2`.

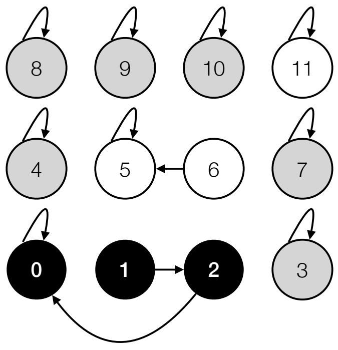


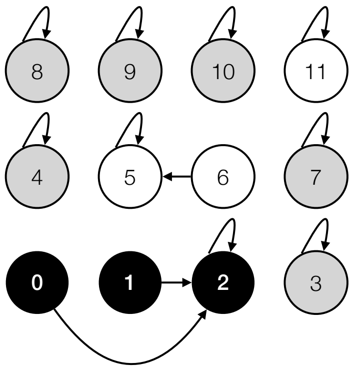
(a) List every move Black could make to win the game. Either of `b1` or `c1` would be an immediate win for Black.

(b) Is there *any* sequence of play from this position in which White wins? If so, list the moves in the form "`B[a3]`, `W[c2]`,...". If not, explain why not. Yes, the following sequence of moves after this position is a win for white: `B[a1]`, `W[b1]`, `B[b3]`, `W[c1]`. More generally, any valid sequence that contains `W[b1]` and `W[c2]` and does *not* contain `B[b1]` or `B[c1]` is acceptable. Only part marks for sequences that satisfy these constraints but start with a move by White.

4. Consider the following position on a 3 × 4 go board, created by the moves `1.B[a1]`, `2.W[b2]`, `3.B[c1]`, `4.W[d3]`, `5.B[b1]`, `6.W[c2]`. Assume that neighbours are iterated over in the order above, right, below, left, and that `union` is implemented without the union-rank optimization.







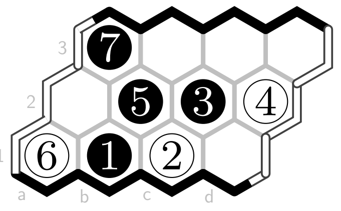
(a) Draw the parent pointers for this position on the parent diagram. See left diagram. I have coloured the nodes according to the colour of the stone that occupies that position, but that is not necessary for full marks.

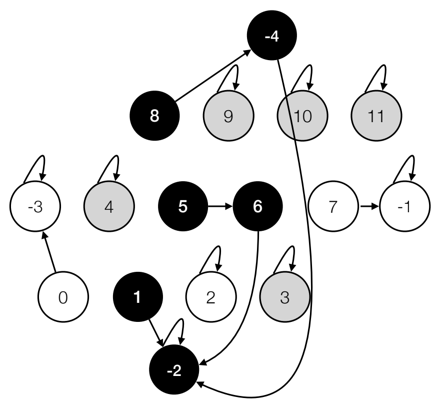
(b) Repeat the question, but assuming that the union-rank optimization is implemented See right diagram. I have coloured the nodes according to the colour of the stone that occupies that position, but that is not necessary for full marks.

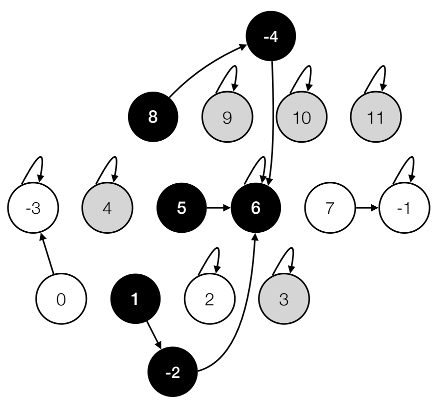
(c) Why does the iteration order matter in (a)? If the order were reversed, then 1 would point to 0, since that was the first neighbouring block encountered; and then 0 would point to 2. So although the block would contain the same stones, it would have different root.

(d) Does the iteration order matter in (b)? Why or why not?
Yes, even in (b) the order matters: with the union rank optimization, iterating in the opposite order would have led to both 1 and 2 pointing to 0, instead of 1 and 0 pointing to 2.

5. Consider the following position on a 3 × 4 Hex board, created by the moves `1.B[b1]`, `2.W[c1]`, `3.B[c2]`, `4.W[d2]`, `5.B[b2]`, `6.W[a1]`, `7.B[a3]`. Assume that neighbours are iterated over in the order above left, above right, right, below right, below left, left, and that `union` is implemented without the union-rank optimization.







   (a) Draw the parent pointers for this position on the parent diagram. See left diagram; I again colour nodes according to stones, but this is optional. I have assumed that borders are treated as "above", "left", "below", or "right" of bordering cells; other answers are acceptable if you assume that borders are iterated over after all neighbouring cells (as they are in `hexgo/stone_board.py`).

   (b) Repeat the question, but assuming that the union-rank optimization is implemented. See right diagram; I again colour nodes according to stones, but this is optional. When `b2` is placed, blocks -2 and 6 have the same rank, so it would be equally acceptable to have an edge from 6 to 2 instead of 2 to 6; in that case, the edge from -4 to 6 would be replaced by an edge from -4 to -2.

6. The following 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) Explain the purpose of this line. These are the coordinate offsets of every possible neighbouring cell; adding each of these tuples elementwise to the row-column coordinates of a cell will give coordinates which, if they are in range, will be the coordinates of a neighbouring cell.

   (b) Give the corresponding line for initializing a go board. `self.nbr_offset = ((-1,0),(0,1),(1,0),(0,`
   (any ordering of these four tuples is acceptable for full marks)

7. Explain the purpose of line 4 below:

```
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:
5           self.nbrs[Pt.rc_point(r,c,self.c)].add(Pt.rc_point(r+y,c+x,self.c))
```

Line 4 checks that the resulting cell coordinates are valid (i.e., don't have negative row or column, and don't have row or column weakly larger than the number of rows or columns on the board).

8. Explain how the `rc_point` function below works. You may be asked to reproduce it.

```
def rc_point(row, col, num_cols):
  return col + row * num_cols
```

For a board with `num_cols` columns, `rc_point` will map each combination of `row` and `col` to a unique integer, which can be used as the "name" of the cell.

9. In `hexgo/stone_board.py`, 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]
```

(2) Merge p and q's blocks in the union-find datastructure. `proot` will be the root of one of the original blocks which now serves as the name of the new, merged block; `qroot` will be the root of the other original block that was added to `proot` to merge the two blocks.

(3) Add all of the stones in the added subtree to the list of `proot`'s stones.

(4) Add all of the liberties of the added subtree to the list of `proot`'s liberties.

(5) Remove all of the stones in the newly added subtree from `proot`'s liberties, since of course a liberty can only be an empty stone. (This handles removing an empty point from the liberties of all neighbouring blocks when a stone of the same colour is placed on it; removing liberties due to a stone of a different colour being placed is handled in `remove_liberties`).

10. Here is a portion of the `tromp_taylor_score` method in `go/go_helper.py`, with two expressions missing.

```
1     while (len(empty_points) > 0):
2       q = empty_points.pop()
3       for j in self.nbr_offsets:
4         x = j + q
5         b_nbr |= (self.brd[x] == BLACK)
6         w_nbr |= (self.brd[x] == WHITE)
7         if self.brd[x] == EMPTY and x not in empty_seen:
8           empty_seen.add(x)
9           empty_points.append(x)
10          territory += 1
11    if   ______expression_(A)______:
12      bt += territory
13    elif ______expression_(B)______:
14      wt += territory
```

(a) What should expression (A) and expression (B) be?

(A) `b_nbr and not w_nbr`

(B) `w_nbr and not b_nbr`

The order of the clauses in each expression of course doesn't matter.

(b) Explain the purpose of lines 5 and 6. Line 5 sets `b_nbr` to `True` when a Black stone is encountered and never sets it back to `False`; after the `for` loop completes it will be `True` iff `q` reaches any Black stone. Similarly, line 6 ensures that `w_nbr` will be `True` iff `q` reaches any White stone.