0. On page 0, in the bubbles, write your ***** CCID ***** (not student id).
   On this page (and all following pages) write your first name, last name and student id.

1. At right, unscramble this code (see `hexgo/stone_board.py`) so that it prints nodes in dfs postorder. Write line numbers only: indent properly. We have written the first line number for you.

```
for nbr in self.nbrs[p]: #(1)
print(p)                 #(2)
self.dfs(nbr, seen)      #(3)
seen[p] = True           #(4)
if not seen[p]:          #(5)
def dfs(self, p, seen):  #(6)
```

(6) ___ ___ ___ ___ ___ ___

___ ___ ___ ___ ___ ___

___ ___ ___ ___ ___ ___

___ ___ ___ ___ ___ ___

___ ___ ___ ___ ___ ___

___ ___ ___ ___ ___ ___

2. For this sliding tile puzzle, give A) number of inversions, B) taxicab score, C) number of nodes in the component of the sliding tile search space graph that includes this position. **Also, below: from the position (the root), draw the next two levels of the search space graph.**

A ____ B ____ C ____

```
_ 1 2 3
4 7 6 5
```

ANSWER ABOVE THIS LINE

ROUGH WORK HERE

3. I ran `stile/15puzzle.py -p 15 14 13 12 10 9 8 11 7 6 4 2 5 1 3` three times, once for each schedule A,B,C below (schedule A places tiles {1,2,3,4} first, schedule B places tile {1} first, etc). For each run, in the solution found, give total moves made and nodes searched.

   Hint: each answer is in {5, 82, 90, 120, 6865, 145722, 1765263, 319625467 }.

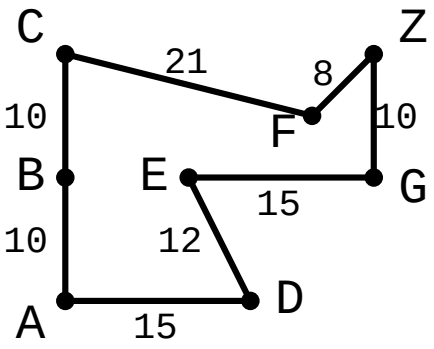
```
                                                                    moves searched

A)[[1,2,3,4], [5,9,13], [6,7,8,10,11,12,14,15]]                      ______  ______

B)[[1], [2], [3,4], [5], [6], [7,8], [9,13], [10,14], [11,12,15]    ______  ______

C)[[1,2], [3,4], [5,6,7,8], [9,10,11,12,13,14,15]]                  ______  ______
```

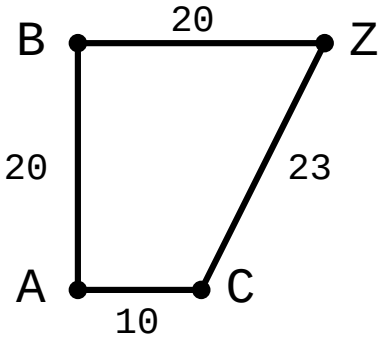
4. Here is a sliding tile puzzle and output from `stile_search_v2.py`. (a) Using this information, give a hardest 2×3 sliding tile puzzle (solvable, but needing the most moves to solve it). (b) Explain how you found your answer to (a). (c) How many moves are needed to solve your puzzle? (d) Explain how you found your answer to (c).

```
1 2 3
5 4 _
360 iterations
level 22 has 0 nodes
last position encountered:
5 4 _
1 2 3
```

```
a) your hardest 2x3 sliding tile position:   ___ ___ ___

                                             ___ ___ ___

b) Explain how you found your answer to a)




c) Moves needed to solve your puzzle  _______




d) Explain how you found your answer to c)
```

3. Here are two roadmaps. Each edge label is a road distance. Below are heuristic estimates of distance remaining to Z, and A* pseudocode from class. Trace the pseudocode on the graph at left, with start A and finish Z. Each time `current` is assigned a node, give the node and its priority. For the graph at right, this is the answer: A 0, C 32, Z 33






```
heuristic
BZ  CZ  DZ  EZ  FZ  GZ  ZZ
26  24  22  18   7  10   0

answer: _____________________________________
```

```
example graph heuristic
BZ  CZ  ZZ
20  22   0

example graph answer: A 0, C 32, Z 33
```

```
fringe = PQ()
fringe.add(start, 0)
parent, cost, done = {}, {}, []
parent[start], cost[start] = None, 0
while not fringe.empty():
  current = fringe.remove() # min priority
  done.add(current)
  if current == target: break
  for next in nbrs(current):
    if next not in done:
      new_cost = cost[current] + wt(current, next)
      if next not in cost or new_cost < cost[next]:
        cost[next] = new_cost
        priority = new_cost + heuristic(next, target)
        fringe.add(next, priority)
        parent[next] = current
```

0. On page 0, in the bubbles, write your ***** CCID ***** (not student id).
   On this page (and all following pages) write your first name, last name and student id.

1. At right, unscramble this code (see `hexgo/stone_board.py`) so that it prints nodes in dfs postorder. Write line numbers only: indent properly. We have written the first line number for you.

```
if not seen[p]:          #(1)
for nbr in self.nbrs[p]: #(2)
print(p)                 #(3)
self.dfs(nbr, seen)      #(4)
seen[p] = True           #(5)
def dfs(self, p, seen):  #(6)
```

(6) ___ ___ ___ ___ ___ ___
___ ___ ___ ___ ___ ___
___ ___ ___ ___ ___ ___
___ ___ ___ ___ ___ ___
___ ___ ___ ___ ___ ___
___ ___ ___ ___ ___ ___

2. For this sliding tile puzzle, give A) number of inversions, B) taxicab score, C) number of nodes in the component of the sliding tile search space graph that includes this position. **Also, below: from the position (the root), draw the next two levels of the search space graph.**

A ____ B ____ C ____

_ 1 2 3
4 7 5 6

ANSWER ABOVE THIS LINE

ROUGH WORK HERE

3. I ran `stile/15puzzle.py -p 15 14 13 12 10 9 8 11 7 6 4 2 5 1 3` three times, once for each schedule A,B,C below (schedule A places tile {1} first, schedule B places tiles {1,2} first, etc). For each run, in the solution found, give total moves made and nodes searched.

   Hint: each answer is in {5, 82, 90, 120, 6865, 145722, 1765263, 319625467 }.

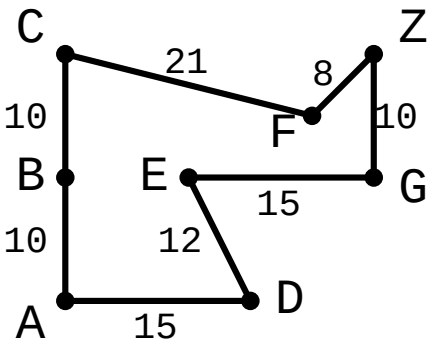
```
                                                                    moves searched

A)[[1], [2], [3,4], [5], [6], [7,8], [9,13], [10,14], [11,12,15] ______  ______

B)[[1,2], [3,4], [5,6,7,8], [9,10,11,12,13,14,15]]               ______  ______

C)[[1,2,3,4], [5,9,13], [6,7,8,10,11,12,14,15]]                  ______  ______
```

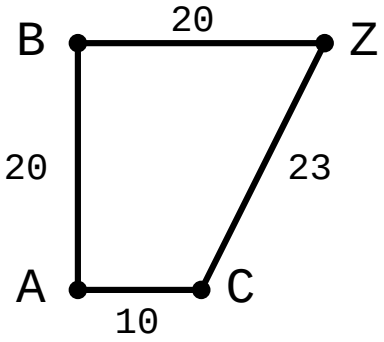
4. Here is a sliding tile puzzle and output from `stile_search_v2.py`. (a) Using this information, give a hardest 2×3 sliding tile puzzle (solvable, but needing the most moves to solve it). (b) Explain how you found your answer to (a). (c) How many moves are needed to solve your puzzle? (d) Explain how you found your answer to (c).

```
2 1 3
4 5 _
360 iterations
level 22 has 0 nodes
last position encountered:
4 5 _
2 1 3
```

```
a) your hardest 2x3 sliding tile position:   ___ ___ ___

                                             ___ ___ ___

b) Explain how you found your answer to a)




c) Moves needed to solve your puzzle  _______




d) Explain how you found your answer to c)
```

3. Here are two roadmaps. Each edge label is a road distance. Below are heuristic estimates of distance remaining to Z, and A* pseudocode from class. Trace the pseudocode on the graph at left, with start A and finish Z. Each time `current` is assigned a node, give the node and its priority. For the graph at right, this is the answer: A 0, C 32, Z 33






```
heuristic
BZ  CZ  DZ  EZ  FZ  GZ  ZZ
26  25  20  17   7  10   0

answer: _____________________________________
```

```
example graph heuristic
BZ  CZ  ZZ
20  22   0

example graph answer: A 0, C 32, Z 33
```

```
fringe = PQ()
fringe.add(start, 0)
parent, cost, done = {}, {}, []
parent[start], cost[start] = None, 0
while not fringe.empty():
  current = fringe.remove() # min priority
  done.add(current)
  if current == target: break
  for next in nbrs(current):
    if next not in done:
      new_cost = cost[current] + wt(current, next)
      if next not in cost or new_cost < cost[next]:
        cost[next] = new_cost
        priority = new_cost + heuristic(next, target)
        fringe.add(next, priority)
        parent[next] = current
```

0. On page 0, in the bubbles, write your ***** CCID ***** (not student id).
   On this page (and all following pages) write your first name, last name and student id.

1. At right, unscramble this code (see `hexgo/stone_board.py`) so that it prints nodes in dfs postorder. Write line numbers only: indent properly. We have written the first line number for you.

```
seen[p] = True            #(1)
if not seen[p]:           #(2)
for nbr in self.nbrs[p]: #(3)
print(p)                  #(4)
self.dfs(nbr, seen)       #(5)
def dfs(self, p, seen):   #(6)
```

(6) ___ ___ ___ ___ ___ ___
___ ___ ___ ___ ___ ___
___ ___ ___ ___ ___ ___
___ ___ ___ ___ ___ ___
___ ___ ___ ___ ___ ___
___ ___ ___ ___ ___ ___

2. For this sliding tile puzzle, give A) number of inversions, B) taxicab score, C) number of nodes in the component of the sliding tile search space graph that includes this position. **Also, below: from the position (the root), draw the next two levels of the search space graph.**

A ____ B ____ C ____

```
_ 1 2 3
4 6 7 5
```

ANSWER ABOVE THIS LINE

ROUGH WORK HERE

3. I ran `stile/15puzzle.py -p 15 14 13 12 10 9 8 11 7 6 4 2 5 1 3` three times, once for each schedule A,B,C below (schedule A places tiles {1,2} first, schedule B places tile {1,2,3,4} first, etc). For each run, in the solution found, give total moves made and nodes searched.

   Hint: each answer is in {5, 82, 90, 120, 6865, 145722, 1765263, 319625467 }.

```
                                                                    moves searched

A)[[1,2], [3,4], [5,6,7,8], [9,10,11,12,13,14,15]]                   ______  ______

B)[[1,2,3,4], [5,9,13], [6,7,8,10,11,12,14,15]]                      ______  ______

C)[[1], [2], [3,4], [5], [6], [7,8], [9,13], [10,14], [11,12,15]     ______  ______
```

4.
```
1 3 2
4 5 _
360 iterations
level 22 has 0 nodes
last position encountered:
4 5 _
1 3 2
```

   Here is a sliding tile puzzle and output from `stile_search_v2.py`. (a) Using this information, give a hardest 2×3 sliding tile puzzle (solvable, but needing the most moves to solve it). (b) Explain how you found your answer to (a). (c) How many moves are needed to solve your puzzle? (d) Explain how you found your answer to (c).

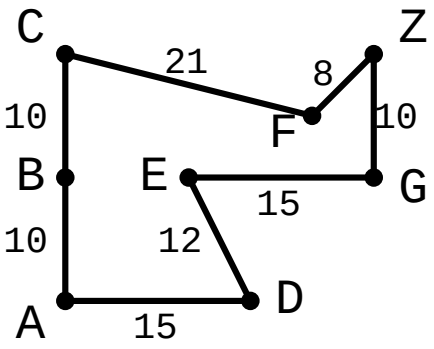
```
a) your hardest 2x3 sliding tile position:   ___ ___ ___

                                             ___ ___ ___
```

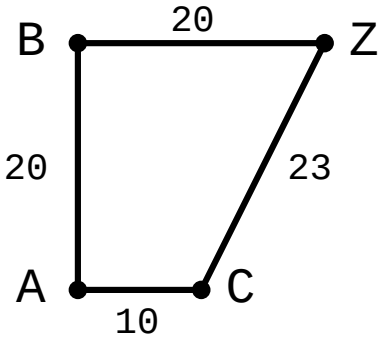
```
b) Explain how you found your answer to a)
```

```
c) Moves needed to solve your puzzle  _______
```

```
d) Explain how you found your answer to c)
```

3. Here are two roadmaps. Each edge label is a road distance. Below are heuristic estimates of distance remaining to Z, and A* pseudocode from class. Trace the pseudocode on the graph at left, with start A and finish Z. Each time `current` is assigned a node, give the node and its priority. For the graph at right, this is the answer: A 0, C 32, Z 33






```
heuristic
BZ  CZ  DZ  EZ  FZ  GZ  ZZ
26  24  22  18   7   2   0

answer: _____________________________________
```

```
example graph heuristic
BZ  CZ  ZZ
20  22   0

example graph answer: A 0, C 32, Z 33
```

```
fringe = PQ()
fringe.add(start, 0)
parent, cost, done = {}, {}, []
parent[start], cost[start] = None, 0
while not fringe.empty():
  current = fringe.remove() # min priority
  done.add(current)
  if current == target: break
  for next in nbrs(current):
    if next not in done:
      new_cost = cost[current] + wt(current, next)
      if next not in cost or new_cost < cost[next]:
        cost[next] = new_cost
        priority = new_cost + heuristic(next, target)
        fringe.add(next, priority)
        parent[next] = current
```