Tent Packing

In [ ]:
```
from instrument import instrument
```

In [ ]:
```
# Pack a tent with different sleeping bag shapes leaving no empty squares
#
# INPUTS:
#   tent_size = (rows, cols) for tent grid
#   missing_squares = set of (r, c) tuples giving location of rocks
#   bag_list = list of sets, each decribing a sleeping bag shape
#      Each set contains (r, c) tuples enumerating contiguous grid
#      squares occupied by the bag, coords are relative to the upper-
#      left corner of the bag.  You can assume every bag occupies
#      at least the grid (0,0).
#
# Example bag_list entries:
#      vertical 3x1 bag: { (0,0), (1,0), (2,0) }
#      horizontal 1x3 bag: { (0,0), (0,1), (0,2) }
#      square bag: { (0,0), (0,1), (1,0), (1,1) }
#      L-shaped bag: { (0,0), (1,0), (1,1) }
#      C-shaped bag: { (0,0), (0,1), (1,0), (2,0), (2,1) }
#      reverse-C-shaped bag: { (0,0), (0,1), (1,1), (2,0), (2,1) }
#
# OUTPUT:
#   None if no packing can be found; otherwise a list giving the
#   placement and type for each placed bag expressed as a dictionary
#   with keys
#     "anchor": (r, c) for upper-left corner of bag
#     "shape": index of bag on bag list
```

Recursive Backtracking Pattern: build on result of sub-problem

In [ ]:
```
def pack(tent_size, missing_squares, bag_list):
    all_squares = set((r, c) for r in range(tent_size[0])
                              for c in range(tent_size[1]))

    def first_empty(covered_squares):
        """ returns (r, c) for first empty square, else None if no empty squares """
        return 'todo'

    def helper(covered_squares):
        """ input: set of covered squares (covered by rocks or bags)
            output: None if no packing can be found, else a list of placed bags"""
        return 'todo'

    # get things started
    return helper(missing_squares)
```

In [ ]:
```python
bag_list = [
  { (0,0), (1,0), (2,0) },  # vertical 3x1 bag
  { (0,0), (0,1), (0,2) },  # horizontal 1x3 bag
  { (0,0), (0,1), (1,0), (1,1) }, # square bag
  { (0,0), (1,0), (1,1) },  # L-shaped bag
  { (0,0), (0,1), (1,0), (2,0), (2,1) },  # C-shaped bag
  { (0,0), (0,1), (1,1), (2,0), (2,1) },  # reverse C-shaped bag
]

# horizontal bag in 1x3 tent, no rocks => fits, no backtracking (case 1)
tent_size = (1,3)
rocks = set()
print(pack(tent_size, rocks, bag_list))
```

In [ ]:
```python
# C-shaped bag in 3x2 tent, one rock => fits, one backtrack (case 6)
tent_size = (3,2)
rocks = {(1,1)}
print(pack(tent_size, rocks, bag_list))
```

In [ ]:
```python
# 5x5 tent with three rocks => fits, backtracking (case 13)
tent_size = (5,5)
rocks = {(1,1),(1,3),(3,1)}
print(pack(tent_size, rocks, bag_list))
```

In [ ]:
```python
# 5x5 tent with 4 rocks => fails; lots of backtracking to try every possibility (case 12)
tent_size = (5,5)
rocks = {(1,1),(1,3),(3,1),(3,3)}
print(pack(tent_size, rocks, bag_list))
```

Recursive Backtracking Pattern: do/undo on success/fail

In [ ]:
```python
def pack(tent_size, missing_squares, bag_list):
    all_squares = set((r, c) for r in range(tent_size[0])
                             for c in range(tent_size[1]))
    def first_empty(covered_squares):
        """ returns (r, c) for first empty square, else None if no empty squares """
        return 'todo'

    def helper(result_so_far, covered_squares):
        """ result_so_far: list of placed bags
            covered_squares: set of squares covered by rocks or bags (will be mutated)
            output: boolean indicating if packing successfully completed """
        return 'todo'

    # get things started
    result = []
    covered_squares = set(missing_squares)
    success = helper(result, covered_squares)
    return result if success else None
```

In [ ]:
```python
# horizontal bag in 1x3 tent, no rocks => fits, no backtracking (case 1)
tent_size = (1,3)
rocks = set()
print(pack(tent_size, rocks, bag_list))
```

In [ ]:
```python
# C-shaped bag in 3x2 tent, one rock => fits, one backtrack (case 6)
tent_size = (3,2)
rocks = {(1,1)}
print(pack(tent_size, rocks, bag_list))
```

```
In [ ]: # 5x5 tent with three rocks => fits, backtracking (case 13)
        tent_size = (5,5)
        rocks = {(1,1),(1,3),(3,1)}
        print(pack(tent_size, rocks, bag_list))
```

```
In [ ]: # 5x5 tent with 4 rocks => fails; lots of backtracking to try every possibility (case 12)
        tent_size = (5,5)
        rocks = {(1,1),(1,3),(3,1),(3,3)}
        print(pack(tent_size, rocks, bag_list))
```

What if we want *all* packings?

```
In [ ]: def all_packings(tent_size, missing_squares, bag_list):
            all_squares = set((r, c) for r in range(tent_size[0])
                                     for c in range(tent_size[1]))

            def first_empty(covered_squares):
                """ returns (r, c) for first empty square, else None if no empty squares """
                return 'todo'

            def helper(covered_squares):
                """ input: set of covered squares (covered by rocks or bags)
                    output: None if no packing can be found, else a list of packings,
                    each packing being a list of placed bags
                """
                return 'todo'

            # get things started
            return helper(missing_squares)
```

```
In [ ]: # Succeeds; more than one packing possible
        tent_size = (3,3)
        rocks = set()
        res = all_packings(tent_size, rocks, bag_list)
        print("NUMBER PACKINGS:", len(res) if res is not None else 0)
        print(res)
```

```
In [ ]: # More packings... (case 5)
        tent_size = (4,4)
        rocks = set()
        res = all_packings(tent_size, rocks, bag_list)
        print("NUMBER PACKINGS:", len(res) if res is not None else 0)
```

```
In [ ]: # 9x7 tent with scattered rocks -- Lots of possibilities (case 15)
        tent_size = (9,7)
        rocks = {(0,2), (2,2), (2,4), (3,4), (7,4), (5,4), (5,5), (8,6), (7,1)}
        res = all_packings(tent_size, rocks, bag_list)
        print("NUMBER PACKINGS:", len(res) if res is not None else 0)
```