Tent Packing

In [1]:
```
from instrument import instrument
```

In [2]:
```
# 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 [3]:

```
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 """
        for row in range(tent_size[0]):
            for col in range(tent_size[1]):
                locn = (row,col)
                if locn not in covered_squares:
                    return locn
        return None

    @instrument
    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"""

        # look for first empty square
        locn = first_empty(covered_squares)

        # base case: no empty squares! We return an empty (successful) packing.
        if locn is None:
            return []

        # try placing each type of bag at locn: if it fits, mark its
        # squares as covered and recursively solve resulting problem
        row, col = locn
        for b in range(len(bag_list)):
            # compute set of squares occupied by bag b at locn
            bag_squares = set((r+row, c+col) for r, c in bag_list[b])

            # is bag in-bounds? if not, it doesn't fit here
            if len(bag_squares - all_squares) != 0:
                continue

            # are all of those bag squares free?
            if len(bag_squares & covered_squares) == 0:
                # yes, try packing with bag at this locn
                result = helper(covered_squares | bag_squares)
                if result is not None:
                    # Success! Found packing of subproblem; build solution
                    result.insert(0, {"anchor": locn, "shape": b})
                    return result
                else:
                    # Failure! Need to try another bag
                    continue

        # oops, no valid placement at this locn
        return None

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

In [4]:
```
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))
```

```
[{'shape': 1, 'anchor': (0, 0)}]

call to helper: set()
   call to helper: {(0, 1), (0, 0), (0, 2)}
   helper returns: []
helper returns: [{'shape': 1, 'anchor': (0, 0)}]
```

In [5]:
```
# 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))
```

```
[{'shape': 4, 'anchor': (0, 0)}]

call to helper: {(1, 1)}
   call to helper: {(2, 0), (1, 0), (0, 0), (1, 1)}
   helper returns: None
   call to helper: {(0, 1), (2, 0), (0, 0), (1, 0), (1, 1), (2, 1)}
   helper returns: []
helper returns: [{'shape': 4, 'anchor': (0, 0)}]
```

In [6]:
```
# 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))
```

```
call to helper: {(1, 3), (3, 1), (1, 1)}

[{'shape': 0, 'anchor': (0, 0)}, {'shape': 5, 'anchor': (0, 1)}, {'shape': 5, 'anchor': (0, 3)},
{'shape': 3, 'anchor': (3, 0)}, {'shape': 1, 'anchor': (3, 2)}, {'shape': 1, 'anchor': (4, 2)}]

   call to helper: {(2, 0), (1, 3), (0, 0), (1, 0), (3, 1), (1, 1)}
      call to helper: {(0, 1), (2, 0), (1, 3), (0, 0), (0, 2), (1, 0), (3, 1) ...
         call to helper: {(0, 1), (1, 3), (0, 0), (3, 1), (1, 4), (2, 0), (1, 1) ...
            call to helper: {(0, 1), (1, 2), (3, 2), (1, 3), (0, 0), (2, 2), (3, 1) ...
            helper returns: None
            call to helper: {(0, 1), (1, 2), (1, 3), (0, 0), (2, 2), (3, 1), (1, 4) ...
            helper returns: None
         helper returns: None
      helper returns: None
      call to helper: {(0, 1), (1, 2), (1, 3), (0, 0), (0, 2), (3, 1), (2, 1) ...
         call to helper: {(0, 1), (1, 2), (1, 3), (0, 0), (0, 2), (3, 1), (2, 1) ...
            call to helper: {(1, 3), (0, 0), (3, 0), (2, 2), (2, 1), (1, 4), (1, 1) ...
               call to helper: {(3, 2), (1, 3), (0, 0), (3, 0), (2, 2), (2, 1), (1, 4) ...
                  call to helper: {(3, 2), (1, 3), (0, 0), (3, 0), (2, 2), (2, 1), (1, 4) ...
                  helper returns: []
               helper returns: [{'shape': 1, 'anchor': (4, 2)}]
            helper returns: [{'shape': 1, 'anchor': (3, 2)}, {'shape': 1, 'anchor': (4,  ...
         helper returns: [{'shape': 3, 'anchor': (3, 0)}, {'shape': 1, 'anchor': (3,  ...
      helper returns: [{'shape': 5, 'anchor': (0, 3)}, {'shape': 3, 'anchor': (3,  ...
   helper returns: [{'shape': 5, 'anchor': (0, 1)}, {'shape': 5, 'anchor': (0,  ...
helper returns: [{'shape': 0, 'anchor': (0, 0)}, {'shape': 5, 'anchor': (0,  ...
```

In [7]:
```
# 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))
```

```
None

call to helper: {(1, 3), (3, 1), (3, 3), (1, 1)}
   call to helper: {(2, 0), (1, 3), (0, 0), (3, 3), (1, 0), (3, 1), (1, 1) ...
      call to helper: {(0, 1), (2, 0), (1, 3), (0, 0), (3, 3), (0, 2), (1, 0) ...
         call to helper: {(0, 1), (1, 3), (0, 0), (3, 3), (3, 1), (1, 4), (2, 0) ...
            call to helper: {(0, 1), (1, 2), (3, 2), (1, 3), (0, 0), (3, 3), (2, 2) ...
            helper returns: None
            call to helper: {(0, 1), (1, 2), (1, 3), (0, 0), (3, 3), (2, 2), (3, 1) ...
            helper returns: None
         helper returns: None
      helper returns: None
      call to helper: {(0, 1), (1, 2), (1, 3), (0, 0), (3, 3), (0, 2), (3, 1) ...
         call to helper: {(0, 1), (1, 2), (1, 3), (0, 0), (3, 3), (0, 2), (3, 1) ...
            call to helper: {(1, 3), (0, 0), (3, 0), (2, 2), (2, 1), (1, 4), (1, 1) ...
               call to helper: {(3, 2), (1, 3), (0, 0), (3, 0), (2, 2), (2, 1), (1, 4) ...
               helper returns: None
            helper returns: None
         helper returns: None
      helper returns: None
   helper returns: None
   call to helper: {(0, 1), (1, 3), (0, 0), (3, 3), (0, 2), (3, 1), (1, 1) ...
      call to helper: {(0, 1), (1, 3), (0, 0), (3, 3), (3, 1), (1, 4), (1, 1) ...
         call to helper: {(0, 1), (1, 3), (0, 0), (3, 3), (3, 0), (1, 0), (3, 1) ...
            call to helper: {(0, 1), (1, 2), (3, 2), (1, 3), (0, 0), (3, 3), (3, 0) ...
            helper returns: None
         helper returns: None
         call to helper: {(0, 1), (1, 3), (0, 0), (3, 3), (1, 0), (3, 1), (1, 4) ...
            call to helper: {(0, 1), (1, 2), (3, 2), (1, 3), (0, 0), (3, 3), (1, 0) ...
               call to helper: {(3, 2), (1, 3), (0, 0), (3, 0), (0, 2), (1, 4), (2, 1) ...
               helper returns: None
            helper returns: None
         helper returns: None
      helper returns: None
   helper returns: None
   call to helper: {(0, 1), (1, 3), (0, 0), (3, 3), (3, 1), (2, 1), (2, 0) ...
      call to helper: {(0, 1), (1, 2), (1, 3), (0, 0), (3, 3), (0, 2), (3, 1) ...
         call to helper: {(0, 1), (1, 2), (1, 3), (0, 0), (3, 3), (0, 2), (3, 1) ...
            call to helper: {(1, 3), (0, 0), (3, 0), (2, 2), (2, 1), (1, 4), (1, 1) ...
               call to helper: {(3, 2), (1, 3), (0, 0), (3, 0), (2, 2), (2, 1), (1, 4) ...
               helper returns: None
            helper returns: None
         helper returns: None
      helper returns: None
      call to helper: {(0, 1), (1, 3), (0, 0), (3, 3), (0, 2), (3, 1), (2, 1) ...
         call to helper: {(0, 1), (1, 2), (3, 2), (1, 3), (0, 0), (3, 3), (0, 2) ...
            call to helper: {(0, 1), (1, 2), (3, 2), (1, 3), (0, 0), (3, 3), (0, 2) ...
            helper returns: None
         helper returns: None
         call to helper: {(0, 1), (1, 2), (1, 3), (0, 0), (3, 3), (0, 2), (2, 2) ...
            call to helper: {(0, 1), (1, 2), (1, 3), (0, 0), (3, 3), (0, 2), (2, 2) ...
               call to helper: {(4, 1), (1, 3), (0, 0), (3, 0), (2, 1), (1, 4), (1, 1) ...
                  call to helper: {(4, 1), (1, 3), (0, 0), (3, 2), (3, 0), (2, 1), (1, 4) ...
                  helper returns: None
               helper returns: None
            helper returns: None
         helper returns: None
      helper returns: None
      call to helper: {(0, 1), (1, 2), (1, 3), (0, 0), (3, 3), (0, 2), (3, 1) ...
         call to helper: {(0, 1), (1, 2), (1, 3), (0, 0), (3, 3), (0, 2), (3, 1) ...
            call to helper: {(1, 3), (0, 0), (3, 0), (2, 2), (2, 1), (1, 4), (1, 1) ...
```

```
              call to helper: {(3, 2), (1, 3), (0, 0), (3, 0), (2, 2), (2, 1), (1, 4) ...
              helper returns: None
           helper returns: None
        helper returns: None
     helper returns: None
  helper returns: None
helper returns: None
```

Recursive Backtracking Pattern: do/undo on success/fail

In [8]:

```
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 """
        for row in range(tent_size[0]):
            for col in range(tent_size[1]):
                locn = (row,col)
                if locn not in covered_squares:
                    return locn
        return None

    @instrument
    def helper(result_so_far, covered_squares):
        """ result_so_far: list of placed bags
            covered_squares: set of squares covered by rocks or bags
            output: boolean indicating if packing successfully completed """
        # look for first empty square
        locn = first_empty(covered_squares)

        # base case: no empty squares!
        if locn is None:
            return True #Signal success; results_so_far holds packing

        # try placing each type of bag: if it fits, mark its squares as covered,
        # add it to the results list, and recursively solve resulting problem.
        row, col = locn
        for b in range(len(bag_list)):
            # compute set of squares occupied by bag b at locn
            bag_squares = set((r+row, c+col) for r, c in bag_list[b])

            # is bag in-bounds? if not, it doesn't fit here
            if len(bag_squares - all_squares) != 0:
                continue

            # are all of those bag squares free?
            if len(bag_squares & covered_squares) == 0:
                # yes, try packing with bag at this locn
                bag = {"anchor": locn, "shape": b}
                result_so_far.insert(0, bag)      # mutate result_so_far
                covered_squares |= bag_squares    # mutate covered_squares
                success = helper(result_so_far, covered_squares)
                if success:
                    # SUCCESS -- we're done! result_so_far holds packing
                    return True
                else:
                    # FAILURE! -- need to backtrack. In this case, we need to
                    # UNDO our changes and try other bags (continue for loop)
                    result_so_far.pop(0)
                    covered_squares -= bag_squares

        # oops, no valid placement at this locn
        return False

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

In [9]:
```
# 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))
```

```
[{'shape': 1, 'anchor': (0, 0)}]

call to helper: [], set()
   call to helper: [{'shape': 1, 'anchor': (0, 0)}], {(0, 1), (0, 0), (0,  ...
   helper returns: True
helper returns: True
```

In [10]:
```
# 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))
```

```
[{'shape': 4, 'anchor': (0, 0)}]

call to helper: [], {(1, 1)}
   call to helper: [{'shape': 0, 'anchor': (0, 0)}], {(2, 0), (1, 0), (0,  ...
   helper returns: False
   call to helper: [{'shape': 4, 'anchor': (0, 0)}], {(0, 1), (2, 0), (0,  ...
   helper returns: True
helper returns: True
```

In [11]:
```
# 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))
```

```
[{'shape': 1, 'anchor': (4, 2)}, {'shape': 1, 'anchor': (3, 2)}, {'shape': 3, 'anchor': (3, 0)},
{'shape': 5, 'anchor': (0, 3)}, {'shape': 5, 'anchor': (0, 1)}, {'shape': 0, 'anchor': (0, 0)}]

call to helper: [], {(1, 3), (3, 1), (1, 1)}
   call to helper: [{'shape': 0, 'anchor': (0, 0)}], {(2, 0), (1, 3), (0,  ...
      call to helper: [{'shape': 1, 'anchor': (0, 1)}, {'shape': 0, 'anchor': ...
         call to helper: [{'shape': 0, 'anchor': (0, 4)}, {'shape': 1, 'anchor': ...
            call to helper: [{'shape': 0, 'anchor': (1, 2)}, {'shape': 0, 'anchor': ...
            helper returns: False
            call to helper: [{'shape': 3, 'anchor': (1, 2)}, {'shape': 0, 'anchor': ...
            helper returns: False
         helper returns: False
      helper returns: False
      call to helper: [{'shape': 5, 'anchor': (0, 1)}, {'shape': 0, 'anchor': ...
         call to helper: [{'shape': 5, 'anchor': (0, 3)}, {'shape': 5, 'anchor': ...
            call to helper: [{'shape': 3, 'anchor': (3, 0)}, {'shape': 5, 'anchor': ...
               call to helper: [{'shape': 1, 'anchor': (3, 2)}, {'shape': 3, 'anchor': ...
                  call to helper: [{'shape': 1, 'anchor': (4, 2)}, {'shape': 1, 'anchor': ...
                  helper returns: True
               helper returns: True
            helper returns: True
         helper returns: True
      helper returns: True
   helper returns: True
helper returns: True
```

In [12]:
```
# 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))
```

```
None

call to helper: [], {(1, 3), (3, 1), (3, 3), (1, 1)}
   call to helper: [{'shape': 0, 'anchor': (0, 0)}], {(2, 0), (1, 3), (0,  ...
      call to helper: [{'shape': 1, 'anchor': (0, 1)}, {'shape': 0, 'anchor': ...
         call to helper: [{'shape': 0, 'anchor': (0, 4)}, {'shape': 1, 'anchor': ...
            call to helper: [{'shape': 0, 'anchor': (1, 2)}, {'shape': 0, 'anchor': ...
            helper returns: False
            call to helper: [{'shape': 3, 'anchor': (1, 2)}, {'shape': 0, 'anchor': ...
            helper returns: False
         helper returns: False
      helper returns: False
      call to helper: [{'shape': 5, 'anchor': (0, 1)}, {'shape': 0, 'anchor': ...
         call to helper: [{'shape': 5, 'anchor': (0, 3)}, {'shape': 5, 'anchor': ...
            call to helper: [{'shape': 3, 'anchor': (3, 0)}, {'shape': 5, 'anchor': ...
               call to helper: [{'shape': 3, 'anchor': (3, 2)}, {'shape': 3, 'anchor': ...
               helper returns: False
            helper returns: False
         helper returns: False
      helper returns: False
   helper returns: False
   call to helper: [{'shape': 1, 'anchor': (0, 0)}], {(0, 1), (1, 3), (0,  ...
      call to helper: [{'shape': 5, 'anchor': (0, 3)}, {'shape': 1, 'anchor': ...
         call to helper: [{'shape': 0, 'anchor': (1, 0)}, {'shape': 5, 'anchor': ...
            call to helper: [{'shape': 0, 'anchor': (1, 2)}, {'shape': 0, 'anchor': ...
            helper returns: False
         helper returns: False
         call to helper: [{'shape': 3, 'anchor': (1, 0)}, {'shape': 5, 'anchor': ...
            call to helper: [{'shape': 0, 'anchor': (1, 2)}, {'shape': 3, 'anchor': ...
               call to helper: [{'shape': 3, 'anchor': (3, 0)}, {'shape': 0, 'anchor': ...
               helper returns: False
            helper returns: False
         helper returns: False
      helper returns: False
   helper returns: False
   call to helper: [{'shape': 4, 'anchor': (0, 0)}], {(0, 1), (1, 3), (0,  ...
      call to helper: [{'shape': 0, 'anchor': (0, 2)}, {'shape': 4, 'anchor': ...
         call to helper: [{'shape': 5, 'anchor': (0, 3)}, {'shape': 0, 'anchor': ...
            call to helper: [{'shape': 3, 'anchor': (3, 0)}, {'shape': 5, 'anchor': ...
               call to helper: [{'shape': 3, 'anchor': (3, 2)}, {'shape': 3, 'anchor': ...
               helper returns: False
            helper returns: False
         helper returns: False
      helper returns: False
      call to helper: [{'shape': 1, 'anchor': (0, 2)}, {'shape': 4, 'anchor': ...
         call to helper: [{'shape': 0, 'anchor': (1, 2)}, {'shape': 1, 'anchor': ...
            call to helper: [{'shape': 0, 'anchor': (1, 4)}, {'shape': 0, 'anchor': ...
            helper returns: False
         helper returns: False
         call to helper: [{'shape': 3, 'anchor': (1, 2)}, {'shape': 1, 'anchor': ...
            call to helper: [{'shape': 0, 'anchor': (1, 4)}, {'shape': 3, 'anchor': ...
               call to helper: [{'shape': 3, 'anchor': (3, 0)}, {'shape': 0, 'anchor': ...
                  call to helper: [{'shape': 3, 'anchor': (3, 2)}, {'shape': 3, 'anchor': ...
                  helper returns: False
               helper returns: False
            helper returns: False
         helper returns: False
      helper returns: False
      call to helper: [{'shape': 4, 'anchor': (0, 2)}, {'shape': 4, 'anchor': ...
         call to helper: [{'shape': 0, 'anchor': (0, 4)}, {'shape': 4, 'anchor': ...
            call to helper: [{'shape': 3, 'anchor': (3, 0)}, {'shape': 0, 'anchor': ...
```

```
            call to helper: [{'shape': 3, 'anchor': (3, 2)}, {'shape': 3, 'anchor': ...
            helper returns: False
         helper returns: False
      helper returns: False
   helper returns: False
helper returns: False
```

What if we want *all* packings?

In [13]:

```
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 """
        for row in range(tent_size[0]):
            for col in range(tent_size[1]):
                locn = (row,col)
                if locn not in covered_squares:
                    return locn
        return None

    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
        """
        # look for first empty square
        locn = first_empty(covered_squares)

        # base case: no empty squares! A packing [] is valid; return a list of that
        if locn is None:
            return [[]]

        ## CHANGED: now build list of all succeeding packings
        packings = None

        # try placing each type of bag: if it fits, mark its squares
        # as covered and recursively solve resulting problem.
        row, col = locn
        for b in range(len(bag_list)):
            # compute set of squares occupied by bag b at locn
            bag_squares = set((r+row, c+col) for r, c in bag_list[b])

            # is bag in-bounds? if not, it doesn't fit here
            if len(bag_squares - all_squares) != 0:
                continue

            # are all of those bag squares free?
            if len(bag_squares & covered_squares) == 0:
                # yes, try packing with bag at this locn
                result = helper(covered_squares | bag_squares)
                if result:
                    ## CHANGED to record ALL PACKINGS. Don't return; instead
                    ## add to list of packings and continue
                    for r in result:
                        if packings is None:
                            packings = []
                        packings.append([{"anchor": locn, "shape": b}] + r)
            # CHANGED: keep looking for more (continue for loop)
        ## CHANGED: Exhausted bag options. Return packings (might be None)
        return packings

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

In [14]:
```
# 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)
```

```
NUMBER PACKINGS: 2
[[{'shape': 0, 'anchor': (0, 0)}, {'shape': 0, 'anchor': (0, 1)}, {'shape': 0, 'anchor': (0, 2)}],
[{'shape': 1, 'anchor': (0, 0)}, {'shape': 1, 'anchor': (1, 0)}, {'shape': 1, 'anchor': (2, 0)}]]
```

In [15]:
```
# 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)
```

```
NUMBER PACKINGS: 5
```

In [16]:
```
# 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)
```

```
NUMBER PACKINGS: 63
```