Lab 4: Tent Packing

Table of Contents

1) Preparation

This lab assumes you have Python 3.5 or later installed on your machine.

The following file contains code and other resources as a starting point for this lab: `lab4.zip`

Most of your changes should be made to `lab.py`, which you will submit at the end of this lab. Importantly, you should not add any imports to the file.

This lab is worth a total of 4 points. Your score for the lab is based on:

- answering the questions on this page (1 point),
- passing the test cases from `test.py` under the time limit (2 points), and
- a brief "checkoff" conversation with a staff member to discuss your code (1 point).

For this lab, you will only receive credit for your tests if they run to completion in under 20 seconds on the server.

Please also review the collaboration policy before continuing.

The questions on this page (including your code submission) are due at 4pm on Friday, Mar 15.

2) Introduction

You decided to go on a luxury beach trip with your `N` friends over Spring Break. You spent all of your internship money on this trip and are looking forward to catching some sun on a secluded island. When you arrive, the water is glorious, and it is just like you had imagined.

Unfortunately, things are not always as they seem. It turns out that instead of sleeping in a bungalow, you will be spending the night in a tent. You must figure out a way to arrange sleeping bags optimally in your tent.

To make matters worse, you realize that several spots under your tent have rocks.

Your assignment is to find a way to pack the tent with sleeping bags such that:

- No one is sleeping on a rock.
- No more than `M` usable (non-rock) portions of the tent are unocuppied.

If no such arrangement exists, you must correctly conclude that no such packing exists.

Unfortunately, many people found themselves in this situation when they attended Fyre, a failed festival that is detailed in Netflix and Hulu documentaries.

2.1) `lab.py` and `test.py`

You must implement your code in this file. You are not expected to read or write any other code provided as part of this lab. Implement the function

```
pack(tent_size, missing_squares, bag_list, max_vacancy)
```

in the file `lab.py`. The four arguments are described below.

If there exists a complete enough tiling of the non-rock squares with no overlap, the function should return a list of people that results in a valid tiling. Each person should be represented with a dictionary (as described in the Section 3) with keys `"anchor"` and `"shape"` and valid corresponding values.

If there is no valid tiling, the function should return `None`.

Your code will be loaded into a tiny web server (`server.py`) which, when running, serves the Tent Packing interfaces from your very own computer acting as a web server (at `http://localhost:8000` -- your computer's own address at port 8000).

Run `./server.py` and visit `http://localhost:8000` in your browser.

2.2) Using the UI

Once your code produces output of a correct type (list of dictionaries with keys `"anchor"` and `"shape"`), it's time to debug your logic!

You can visualize the output to help debug your code. Run `server.py` and open your browser to `localhost:8000`. You will be able to select any of the test inputs from the `./cases` folder and examine them in the browser.

You can visualize the output produced by your code by pressing the `RUN` button. This will display the tiling that your code outputs. If you output `None`, it will color the grid red.

The in-browser UI uses the cases in `resources/cases/`; you are free to add additional ones if you would like to visualize additional cases.

3) Representing Tents and Friends

We will use a 2-dimensional grid to represent the tent. Each rock occupies one square in this grid. A tent configuration is described by variables:

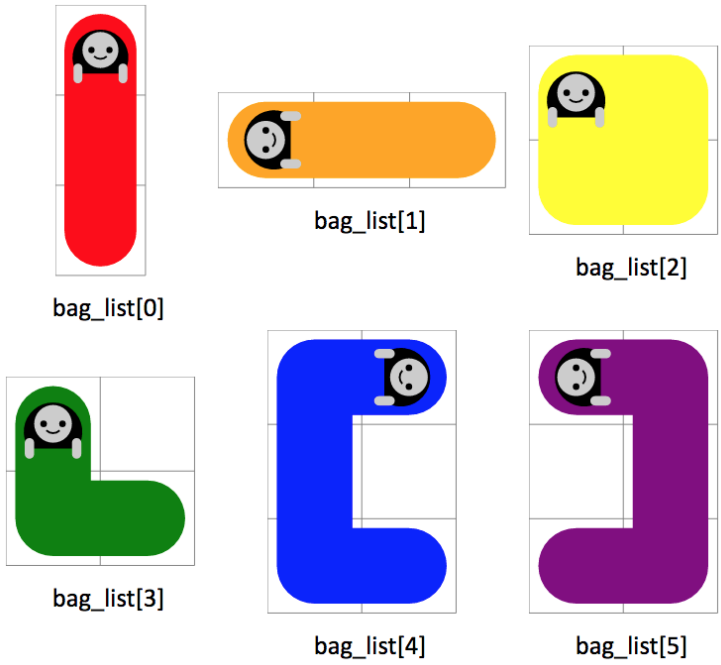
- `tent_size`, which is a Python `tuple` with two integers `(nrows, ncols)` -- the dimensions of the tent in terms of number of rows and number of columns in the grid.
- `missing_squares`, which is a Python `set` (possibly empty) of the grid squares with rocks under them. Each square is represented as a tuple with two integers `(row, col)`, the coordinates of the square. The square `(0,0)` is at the top-left corner, with row numbers increasing down the page and column numbers increasing left-to-right.

For example, a tent configuration would be represented as:

```
tent_size = (3,6)
missing_squares = {(2,1), (0,4), (1,4)}
```

This could denote the following tent configuration.

It turns out your friends are quite flexible and can sleep in any of the following sleeping bag shapes:



You only have to place the bags in the tent in the orientation shown -- don't worry about rotations or reflections.

A sleeping-bag shape is described by a set of tuples that give the row and column of each square occupied by the sleeping bag, relative to the top-left square of the bag which has coordinate `(0,0)`. Note that all sleeping-bag shapes will occupy their top-left squares.

For example, the orange horizontal 1x3 bag is described by the set

```
{(0,0), (0,1), (0,2)}
```

and the blue C-shaped bag by the set

```
{(0,0), (0,1), (1,0), (2,0), (2,1)}.
```

You will be given a list of possible sleeping-bag shapes, i.e., a list of sets, called `bag_list`. We'll be using the shapes shown above, but your code should be general enough to handle any shape given in `bag_list`.

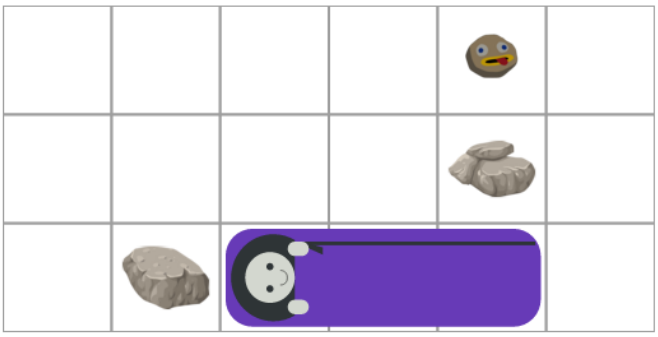
A friend's position and shape are represented by a dictionary with two keys:

- `"anchor"` is a tuple `(row, col)` giving the coordinates of the **top-left square** of the position in the tent occupied by the friend's sleeping bag.
- `"shape"` specifies a particular sleeping-bag shape as an integer index into `bag_list`.

For example, a person would be represented as a dictionary:

```
{"anchor": (2,2), "shape": 1}
```

In the example tent above this would correspond to:



4) Valid Tiling

Let's say our tent has dimensions `nrows` by `ncols`.

We are given a `max_vacancy`, which is an integer (possibly 0) specifying the number of squares allowed to be unoccupied.

A valid tiling is a list of people (with `anchor` and `shape` values) such that, for each square `(r, c)` occupied by a sleeping bag:

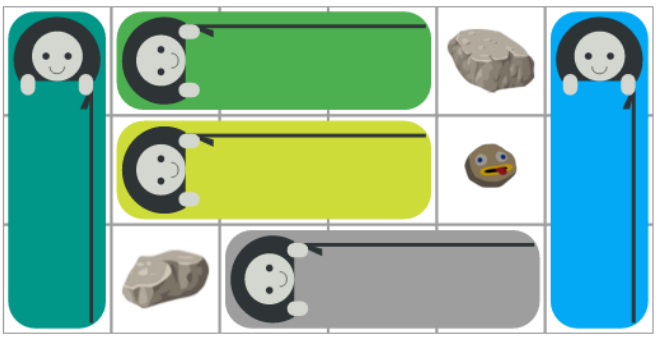
- `0 <= r < nrows` and `0 <= c < ncols` (i.e., each person lies entirely within the tent).
- No rock exists under `(r, c)` (i.e., no person sleeps on a rock).
- No two people have a square in common (i.e., no two people overlap).
- No more than `max_vacancy` non-rock squares are unoccupied by a person.

You can use as many or as few bag shapes from `bag_list` as you like to construct a valid tiling (i.e. you are able to use a bag shape multiple times, and you do not need to use every bag contained in the `bag_list`).

For example, let's say we are given the input from the **Representation** section.

```
tent_size = (3,6)
missing_squares = {(2,1), (0,4), (1,4)}
```

The following is a valid tiling for this tent with three 1x3 bags and two 3x1 bags when `max_vacancy` is 0.

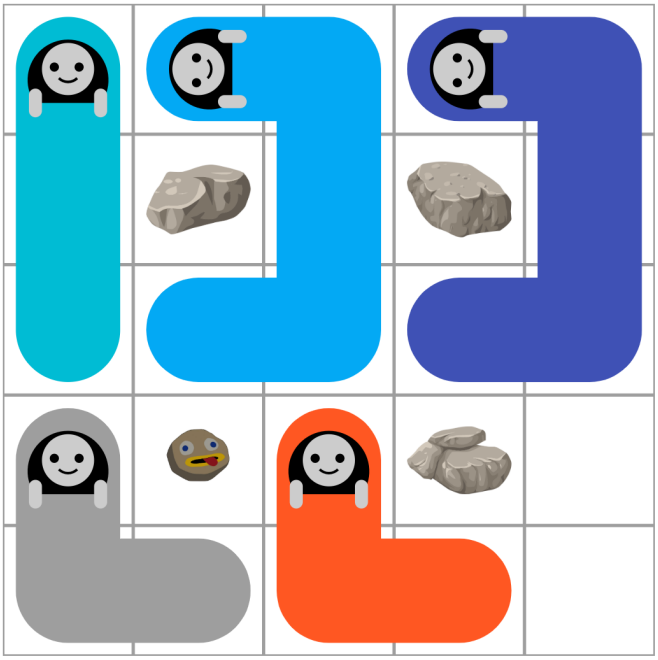


The corresponding list of people (in no particular order) would look like:

```
[
  {"anchor": (0,1), "shape": 1},
  {"anchor": (1,1), "shape": 1},
  {"anchor": (2,2), "shape": 1},
  {"anchor": (0,0), "shape": 0},
  {"anchor": (0,5), "shape": 0},
]
```

4.1) Check Yourself

To familiarize yourself with packing the tent, answer the following questions about valid tiling for a 5x5 tent with 4 rocks.



Is the above a valid tiling for a 5x5 tent with 4 rocks when `max_vacancy` is 3?

Yes

Submit

You have submitted this assignment 1 time.

This question is due on Friday March 15, 2019 at 04:00:00 PM.

Is the above a valid tiling for a 5x5 tent with 4 rocks when `max_vacancy` is 2?

Yes

Submit

You have submitted this assignment 1 time.

This question is due on Friday March 15, 2019 at 04:00:00 PM.

What is the corresponding list of people for the tent tiling above? Enter your answer below as a Python `list` of people, where each person is represented as a dictionary:

[{"anchor": (0,0), "shape": 0},{"anchor": (0,1), "shape": 5},·

Submit

You have submitted this assignment 1 time.

This question is due on Friday March 15, 2019 at 04:00:00 PM.

5) Code Submission

As usual, we provide you with a `test.py` script to help you evaluate the correctness of your code. As always, these tests are not necessarily comprehensive; you should feel free to add additional unit tests to debug your code. The script will call `pack` from `lab.py` with a number of test cases drawn from the `resources/cases/` folder and verify their outputs. The in-browser UI uses the cases in `resources/cases/`; you are free to add additional cases if you would like help visualizing them. We encourage you to use the UI and to write your own test cases to help you diagnose any problems and further verify correctness.

Submit your `lab.py` below:

Download Your Last Submission
Click to View Your Last Submission
Select File No file selected
Submit
You have submitted this assignment 1 time.
This question is due on Friday March 15, 2019 at 04:00:00 PM.

6) Checkoff

Once you are finished with the code, please come to a tutorial, lab session, or office-hours session and add yourself to the queue asking for a checkoff. **You must be ready to discuss your code and test cases in detail before asking for a checkoff.**

You should be prepared to demonstrate your code (which should be well-commented, avoid repetition, and make good use of helper functions). In particular, be prepared to discuss:

- In English, the general recipe you came up with to find a valid tent packing, if one exists.
- How you kept track of the unoccupied tent squares while packing the tent.
- Your code for `pack`, including any helper functions.

6.1) Grade

Grading:

- Concept questions (1 points): 1
- Tests (2 points): 2
- Checkoff (1 points): 1

Total: 4 Points (of 4)