2023 Summer COMP10001 Project 2 Description

Problem 1: Read Files

See grok for hints and two sample files

Your task is to write a function `read_ponies(filename)` that parses a file with the structure described below and returns a tuple of (elevations, path, ponies), where

- `elevations` is a 2-dimensional list with the elevation of each location.
- `path` is a list of adjacent locations from (0, 0) to (M, N), where M and N are the largest possible x and y coordinate, respectively.
- `ponies` is a list of pony tuples (described below).

`elevations` and `path` are the same as in project 1, but ponies in this question are different! Each pony is represented by a tuple of `(id, personality, status)`. In this "pony tuple":

- `id` is an int in range (0, num_ponies)
- `personality` is either the string `"s"` or the string `"a"`. Ponies with a personality of `"s"` are selfish, and ponies with an `"a"` personality are altruistic.
- `status` is a list in the form of `[energy, location]`, where `energy` is a non-negative integer, and `location` is a coordinate (that is, tuple in the form of `(x, y)`).

The list of ponies should be ordered by increasing pony id.

The structure of the file is as follows:

- The text 'Elevations:'
- An arbitrary number of lines, each corresponding to a row of elevations.
- An arbitrary number of blank lines
- The text 'Path:'
- An arbitrary number of lines, each corresponding to a location on a single path.
- An arbitrary number of blank lines
- The text 'Ponies:'
- An arbitrary number of lines, each corresponding to information for a single pony.
- An arbitrary number of blank lines

(Sample file: see Grok)

You may assume that the file is well-formatted .txt file, ie that there are no variations from the format described above. You can also assume that ponies in the file are listed in increasing order of pony id and that every pony has a location of (0, 0). However, there might be an arbitrary number of white spaces at the beginning of the line, end of the line, and after each comma `,` (see file_sample2.txt).

Problem 2: Selfish Ponies Climb

Problem Description with Example Illustration

Your task in this question is to write a function `selfish_climb(elevations, path, ponies)` that simulates the movement of selfish ponies and returns a dictionary in the format described below.

This function should have three arguments - elevations, path, and ponies as described in the previous question. In this question, all ponies in the ponies list have the personality `"s"`.

Your function should return a dictionary where each key is an integer timestep (ie. 0, 1, 2...), and the values are a list of pony tuples at that timestep. The final timestep is defined below. As in the previous question, each ponies list should be sorted by id. That is, ponies with lower ids are at the front.

Selfish Pony Behaviour

The `"s"` (selfish) ponies only care about themselves. Each selfish pony travels individually along the path, completely disregarding other ponies also travelling on the path.

Pony energy follows these rules:

- Ponies use energy when travelling to higher elevations.
- Ponies gain energy when they move to lower elevations.
- Energy usage is the elevation difference between the source location and the destination.
- A pony cannot have negative energy.

For example, when an `"s"` pony with an energy of 52 moves from elevation 13 to 30, its energy is reduced by 17, leaving it with energy = 35. Similarly, when an `"s"` pony with an energy of 54 moves from elevation 30 to 13, its energy is increased by 17, leaving it with an energy of 71. A selfish pony with 10 energy can not move from elevation 20 to 31, because that would lead to negative energy - so it is stuck at its current location.

Simulate Movement

In this question, you are asked to simulate the movement of selfish ponies along the path.

Movement follows these rules:

- At a given time `t`, the pony with `id < t` can move, where t is a valid integer less than or equal to the value of the final timestep. For example, at `t=0`, all ponies with `id < 0` consider a move, and at `t=1`, all ponies with `id < 1` consider a move.
- A pony may or may not move after a move is considered, depending on (1) the energy difference and (2) if the pony has reached the destination.
- At each timestep, ponies move in order of ascending id, ie pony 0 would move before pony 1.

Note that you can assume all ponies have unique ids, and the ids of ponies in the `ponies` list are continuous integers in ascending order. The smallest id is 0, and the largest id is `len(ponies) - 1`.

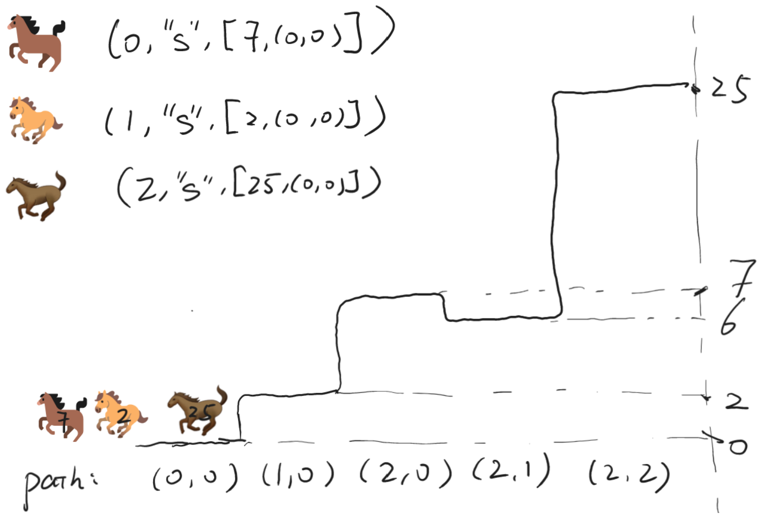
Final Timestep

There must be enough timesteps so that every pony considers at least one move. The final timestep is the first timestep when both of the following conditions are met:

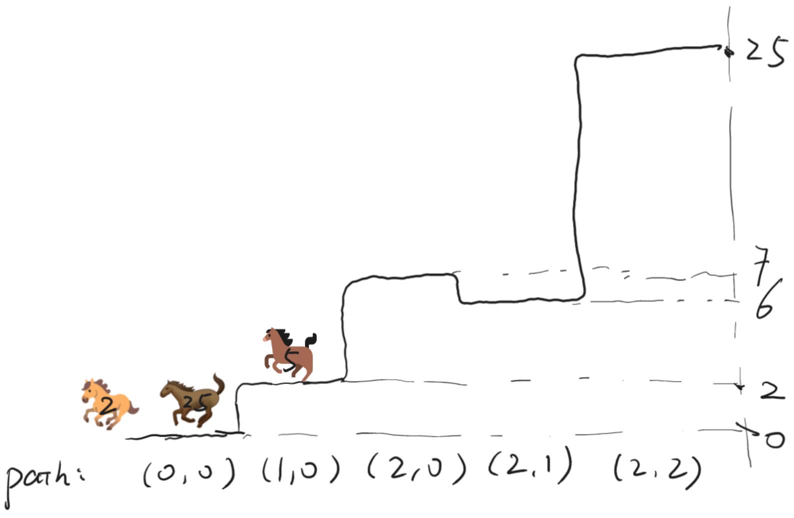
- each pony has considered at least one move
- each pony either has insufficient energy to move or has arrived at the final destination.

For example1 in this question, elevations = [[0, 32, 45], [2, 5, 19], [7, 6, 25]], path = [(0, 0),(1, 0),(2, 0),(2, 1),(2, 2)] and ponies=[(0, s, [7, (0, 0)]), (1, s, [2, (0, 0)]), (2, s, [25, (0, 0)])], the result looks like this:

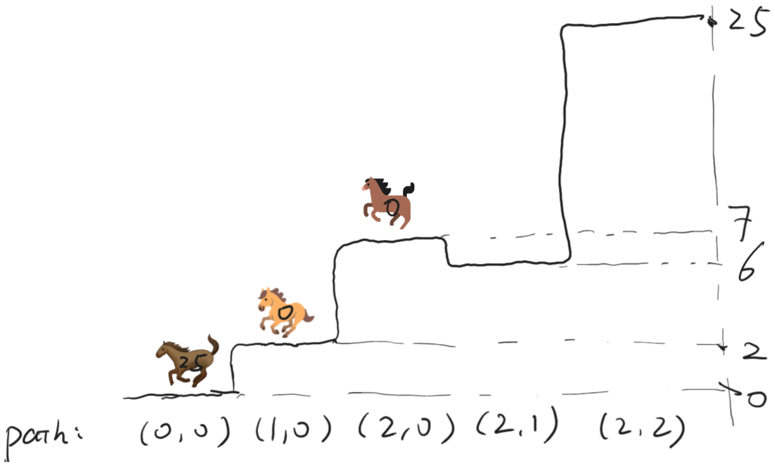
t=0



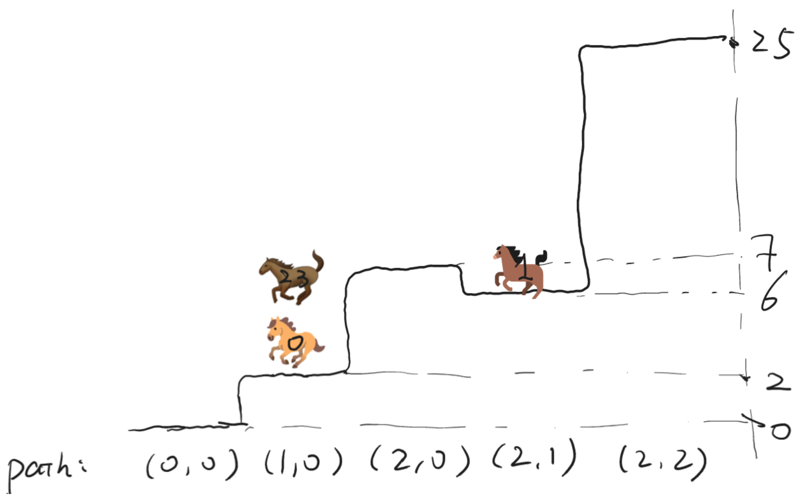
t=1, pony1 considers a move, and move



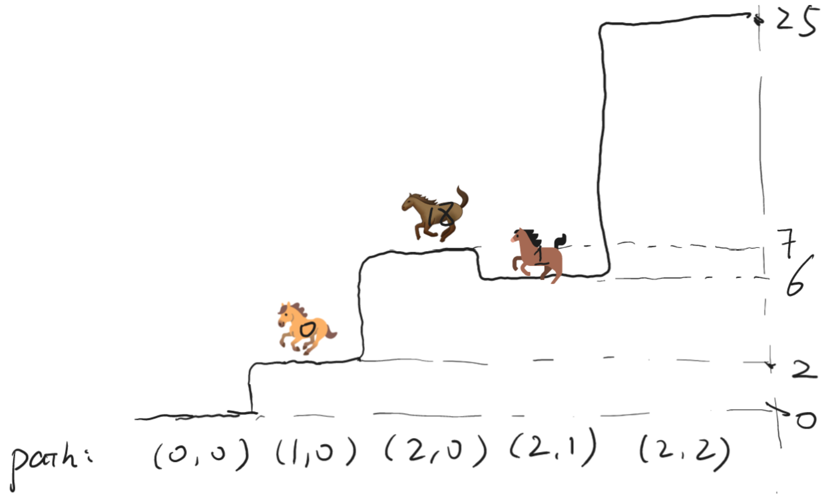
t=2, pony1 pony2 both consider a move, and they all move



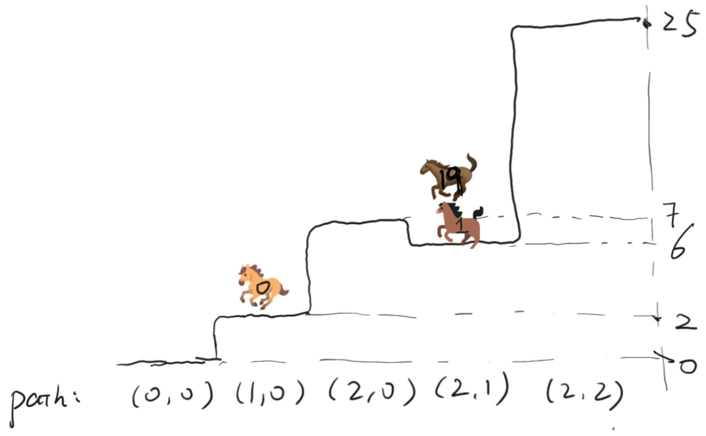
t=3, all pony consider a move, pony0 and pony2 move, but pony1 is stuck



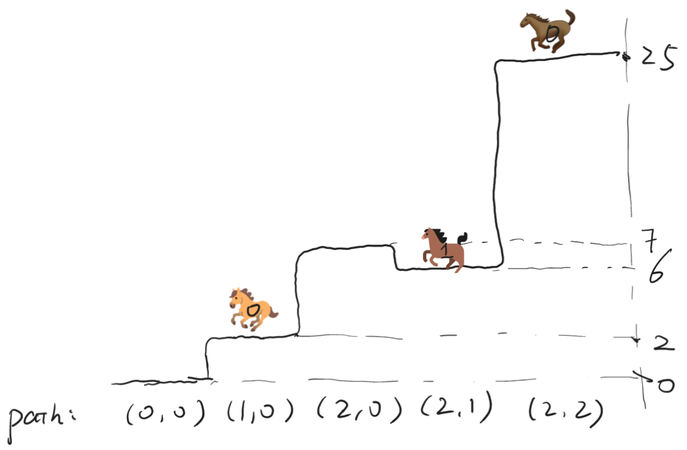
t=4, all pony consider a move, pony2 move, but pony1 and pony2 are stuck



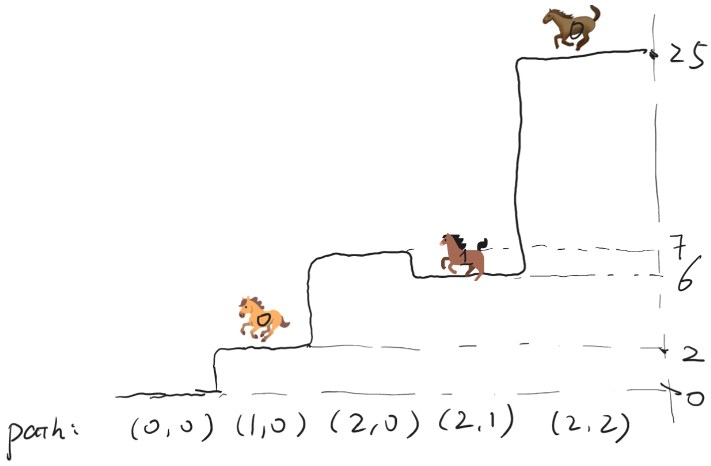
t=5, same as t=4



t=6, same as t=4



t=7, pony2 find itself arrived, pony0 and pony1 are stuck, so the ponies list at t=7 is identical as the ponies list in t=6. Therefore, the final timestep is t=6.



Return:

```
{0: [(0, 's', [7, (0, 0)]), (1, 's', [2, (0, 0)]), (2, 's', [25, (0, 0)])],
1: [(0, 's', [5, (1, 0)]), (1, 's', [2, (0, 0)]), (2, 's', [25, (0, 0)])],
2: [(0, 's', [0, (2, 0)]), (1, 's', [0, (1, 0)]), (2, 's', [25, (0, 0)])],
3: [(0, 's', [1, (2, 1)]), (1, 's', [0, (1, 0)]), (2, 's', [23, (1, 0)])],
4: [(0, 's', [1, (2, 1)]), (1, 's', [0, (1, 0)]), (2, 's', [18, (2, 0)])],
5: [(0, 's', [1, (2, 1)]), (1, 's', [0, (1, 0)]), (2, 's', [19, (2, 1)])],
6: [(0, 's', [1, (2, 1)]), (1, 's', [0, (1, 0)]), (2, 's', [0, (2, 2)])]}
```

Problem 3: Teamwork Climb

Problem Description with Example Illustration

Your task in this question is to write a function `teamwork_climb(elevations, path, ponies)` that simulates the movement of selfish and altruistic ponies and returns a dictionary in the format described below.

This function should have three arguments - `elevations`, `path`, and `ponies` as described in the previous question.

Your function should return a dictionary where each key is an integer timestep (ie. 0, 1, 2...), and the values are a list of pony tuples at that timestep. Consistent with Q2, each ponies list should be sorted by id. That is, ponies with lower ids are at the front. Please also refer to Q2 for the definition of the final timestep.

This task is similar to the task in the previous question, with the difference being that ponies either have the personality `"s"` or `"a"`. Remember that at each time step, ponies try to move in order of ascending id (ie Pony 0, then Pony 1, then Pony 2 ... etc).

Selfish pony behaviour

Refer to Q2

Altruistic pony behaviour

At each time step, an altruistic pony can give its energy to other ponies. Altruistic ponies will try to help other ponies move along the path before moving along the path itself.

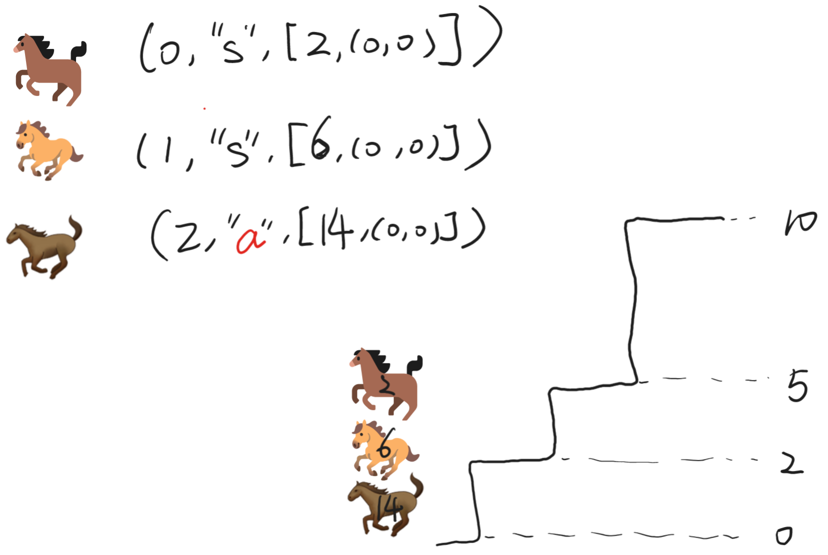
At a timestep `t`, assumes that P1 is an altuistic pony and P2 is a pony of any type, and if all of the following are true in P1's turn:

- P1 and P2 are in the same location
- P2 has a smaller id than P1
- P2 is stuck at the location
- P1 can give enough energy to P2 so that P2 has enough energy to move to the next location
- After giving P2 the energy it requires, P1 still has non-negative energy then the following two events will occur in order:
- In this timestep `t`, P1 gives P2 exactly enough energy to allow P2 to move to the next location
- In this timestep `t`, P1 moves to the next location if it still has sufficient energy to do so

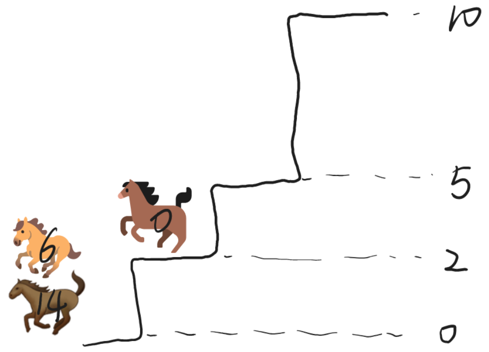
Note that at a time step, an altruistic pony P1 can give its energy to multiple ponies. If there are multiple ponies that satisfy the four criteria above, the altruistic pony gives its energy to the pony with the lower id first. Also, if any of the four criteria for giving energy are not satisfied (that is, no ponies need help or P1 is not able to help), then the altruistic pony P1 behaves just like a selfish pony.

Example 1 - elevations = [[0, 2, 5], [0, 0, 10]], path = [(0, 0), (0, 1), (0, 2), (1, 2)], ponies=[(0, 's', [2, (0, 0)]), (1, 's', [6, (0, 0)]), (2, 'a', [14, (0, 0)])]:

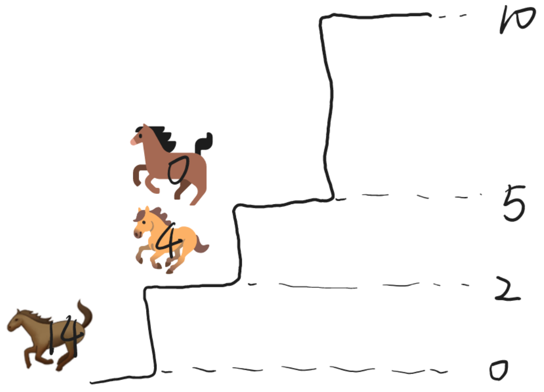
Initially t=0



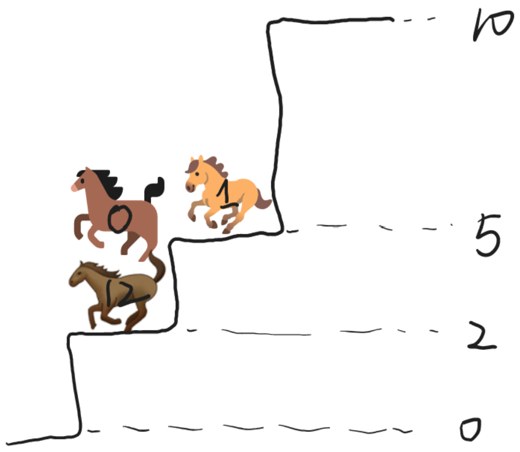
t=1



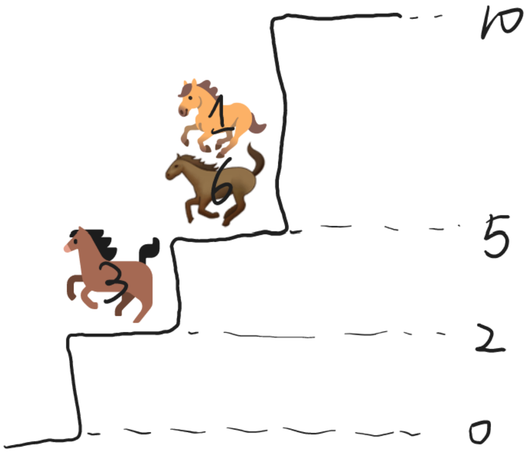
t=2, pony0 is stuck, pony1 moves



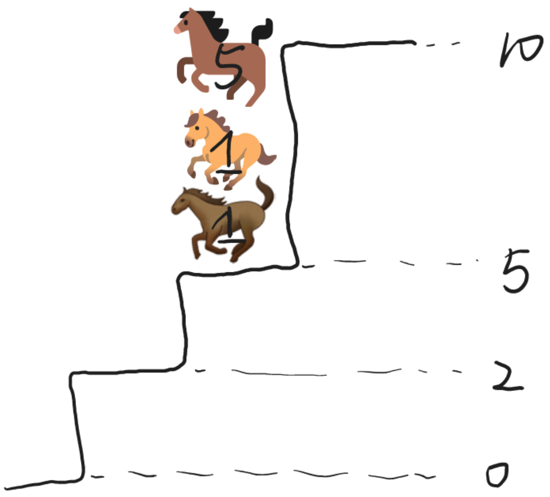
t=3



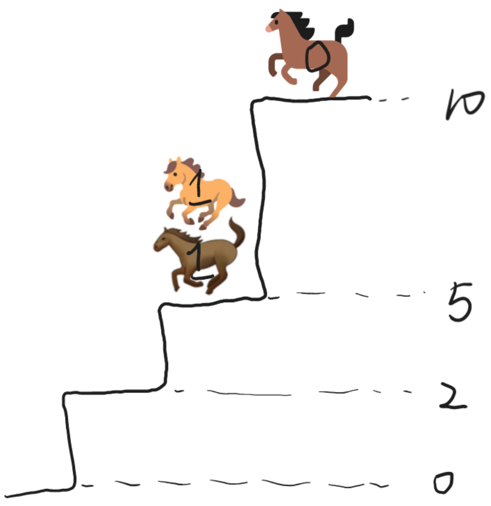
t=4, pony0 is still stuck, then pony1 is also stuck, then pony2 gives its energy to pony 0 and move forward



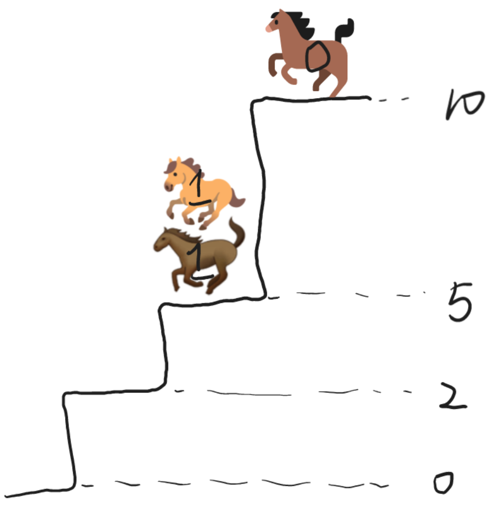
t=5 pony0 moves, pony1 is still stuck, then pony2 gives energy to pony0 and find itself stuck



t=6, pony0 moves, pony1 and pony2 are stuck



t=7, the ponies list does not change. Thus t=6 is the final timestep



Return:
{0: [(0, 's', [2, (0, 0)]), (1, 's', [6, (0, 0)]), (2, 'a', [14, (0, 0)])],
1: [(0, 's', [0, (0, 1)]), (1, 's', [6, (0, 0)]), (2, 'a', [14, (0, 0)])],
2: [(0, 's', [0, (0, 1)]), (1, 's', [4, (0, 1)]), (2, 'a', [14, (0, 0)])],
3: [(0, 's', [0, (0, 1)]), (1, 's', [1, (0, 2)]), (2, 'a', [12, (0, 1)])],
4: [(0, 's', [3, (0, 1)]), (1, 's', [1, (0, 2)]), (2, 'a', [6, (0, 2)])],
5: [(0, 's', [5, (0, 2)]), (1, 's', [1, (0, 2)]), (2, 'a', [1, (0, 2)])],
6: [(0, 's', [0, (1, 2)]), (1, 's', [1, (0, 2)]), (2, 'a', [1, (0, 2)])]}