

Practice Problems 3

Problem 1 - Bipartite Graph

Write a function `is_bipartite(graph)` that takes as an argument a **connected** graph, and returns `True` if a graph is bipartite.

Input

The `graph` is a list of length `n`, each element of list is a list of integers. `graph[i]` contains a list all neighbors of vertex `i`. You can assume that the graph is undirected: if `j` is an element of `graph[i]`, then `i` is an element of `graph[j]`. The vertices are numbered from `0` to `n-1` inclusive.

You can assume that the graph given as an argument is connected - all vertices are reachable from vertex `0`.

Hint

As a reminder, a graph is bipartite if its vertices can be colored into two colors, in such a way that every edge connects vertices of different colors.

To check if a graph is bipartite you can use a variant of the DFS algorithm implemented in lecture 12. In addition to array `visited`, pass an array `color` of length `n` (with values `0` and `1`). Before you visit a given vertex, set its color to the opposite of the vertex you are coming from. Check if what it returned is a valid coloring.

Example

```
is_bipartite([[1,3], [0,2, 4], [1,3,5], [0, 2], [1], [2]]) == True
```

Since all vertices with even indices are only connected with odd vertices, and vice-versa.

```
is_bipartite([[1,2], [0,2], [0,1]]) == False
```

The graph here is just a triangle on vertices `0,1,2`. It is not bipartite.

Problem 2 - Banach's fixed point

As we will see, for any function `f` that takes integer in range `0` to `n` (inclusive), and output integers from `0` to `n` (inclusive), at least one of the following two holds:

1. There is an x such that $x = f(x)$

2. There is a pair $x \neq y$ such that $|f(x) - f(y)| \geq |x - y|$.

Your goal is to write a function `banach(n, f)` that takes as an argument an integer `n` and a function `f`. Your function should either return a pair `(x, x)`, such that `f(x) == x`, or a pair `(x, y)` such that `|f(x) - f(y)| >= |x-y|`.

The function `f` can be assumed to output integer from `0` to `n` given an integer from `0` to `n` at the input.

Scoring and algorithm

Full score for a solution running in time $O(n)$, partial score will be given for slower solution.

One way to write algorithm running in time $O(n)$, is the following

1. Set $x := 0, y := f(x)$
2. Repeat in a loop:
 1. If $|f(x) - f(y)| \geq |x - y|$ you can just return the pair (x, y) .
 2. If $f(x) = f(y)$, you can return (y, y) - since $y = f(x)$.
 3. Otherwise, set $x := f(x)$ and $y := f(y)$. (Note after this operation, we still have $y = f(x)$)

Observe that this algorithm will terminate in $O(n)$ steps, since initially $|x - y| \leq n$, and in each iteration of the loop $|x - y|$ is decreasing.

Example

```
N = 100
def fun(x):
    return (x + 7)**2 % N

banach(N, fun) == (49, 36)
```

Since `fun(49) == 36` and `fun(36) == 49`,

```
N= 100

banach(2*N, lambda x : (x - N)//2 + N) == (99, 99)
```

Problem 3 - Valid bracketing

A valid bracketing sequence is a string with characters '(' and ')', s.t. each opening bracket '(' has a matching closing bracket ')'. For example `'(()())'` is a valid bracketing sequence, where `'((())('` is not.

Write a function `brackets(n)` that returns a list of all valid bracketing sequence of length n .

Example

```
brackets(6) == ['((()))', '(()())', '()(())', '()(()()', '()()()' ]  
brackets(5) == []
```

Hint

Implement the recursive function

`brackets_help(n, k)`, which produces all sequences of length n that form a valid bracketing together with k opening brackets at the beginning.

The `brackets` function should just return `brackets_help(n, 0)`.

The `brackets_help` function has the following recursive structure:

1. if $n = k$ the only valid sequence consists of k closing brackets `)`.
2. if $n = 0$, (and $n \neq k$), there are no valid sequences.
3. Otherwise there are two possibilities for the first character.
 1. Starting with '`(`':
 - Use `brackets_help` recursively to get all string of length $n - 1$, closing $k + 1$ brackets; append `'(` at the beginning of each.
 2. Starting with '`)`':
 - If $k > 0$, use `brackets_help` recursively to get all strings of length $n - 1$ closing $k - 1$ brackets; append `')` at the beginning of each.
4. Combine the two lists and return as a result of a function.