Refaktorizácia

Vybrali sme si skúškové zadanie z predmetu Programovanie(2)

s názvom Usilovný ježko(http://input.sk/python2018/sk2018l1.html),

ktorého cieľom bolo naprogramovať pomocou backtrackingu trasu ježka po záhrade,

pričom cestou mal pozbierať ovocia a z každého druhu práve jedno.

(Konkrétne refaktorizácie sú od tretej snímky, pričom na druhej je také všeobecnejšie zhrnutie)

Autori: Šimon Babál, Matúš Gál

Refaktorizácia __init__

```python
def __init__(self, meno_suboru):
    def pridaj(v1, v2, o=''):
        try:
            self.g[v1][v2] = o
        except KeyError:
            self.g[v1] = {v2:o}
    with open(meno_suboru, 'r') as file:
        z = [_.split() for _ in
file.read().splitlines()]
    self.g = {}
    self.o = set()
    for _ in z:
        if len(_) == 2:
            pridaj(_[0], _[1])
            pridaj(_[1], _[0])
        elif len(_) == 3:
            self.o.add(_[1])
            pridaj(_[0], _[2], _[1])
            pridaj(_[2], _[0], _[1])
```

```python
def __init__(self, file_name: str) -> None:
    self.garden: Dict[str: Dict[str]] = {}
    self.file_name: str = file_name
    self.fruits: Set[str] = set()
    self.solution: List[str] = []
    self._create_graph(self._read_file())

def _read_file(self) -> List[List[str]]:
    with open(self.file_name, 'r') as file:
        return [row.split() for row in file.read().splitlines()]

def _add_edge(self, vertex1: str, vertex2: str, fruit: str) -> None:
    if self._vertex_exists(vertex1):
        self.garden[vertex1][vertex2] = fruit
    else:
        self.garden[vertex1] = {vertex2: fruit}

def _vertex_exists(self, vertex: str) -> bool:
    return vertex in self.garden

def _create_graph(self, rows: List[List[str]]) -> None:
    for row in rows:
        if self._row_contains_fruit(row):
            vertex1, fruit, vertex2 = row[0], row[1], row[2]
            self.fruits.add(fruit)
        else:
            vertex1, fruit, vertex2 = row[0], '', row[1]
        self._add_edge(vertex1, vertex2, fruit)
        self._add_edge(vertex2, vertex1, fruit)

@staticmethod
def _row_contains_fruit(row: List[str]) -> bool:
    if len(row) == 3:
        return True
    elif len(row) == 2:
        return False
    raise RuntimeError("Row has incorrect number of items")
```

Refaktorizácia 1(__init__)

```
def pridaj(v1, v2, o=''):
    try:
      self.g[v1][v2] = o
    except KeyError:
      self.g[v1] = {v2:o}
```

```
def _add_edge(self, vertex1: str, vertex2: str, fruit: str) -> None:
    if self._vertex_exists(vertex1):
        self.garden[vertex1][vertex2] = fruit
    else:
        self.garden[vertex1] = {vertex2: fruit}

def _vertex_exists(self, vertex: str) -> bool:
    return vertex in self.garden
```

Vysvetlenie: Namiesto toho, aby som skušal, či dostanem vynimku KeyError v pripade, ze dana krizovatka este neexistuje v mape zahradky som dal na to radsej samostatnu funkciu, ktora to overuje, pretoze vynimky by sa mali pouzivat len pri chybovych stavoch, pricom tento stav by nemal byt brany ako chybovy.

Refaktorizácia 2(__init__)

```
for _ in z:
        if len(_) == 2:
            pridaj(_[0], _[1])
            pridaj(_[1], _[0])
        elif len(_) == 3:
            self.o.add(_[1])
            pridaj(_[0], _[2], _[1])
            pridaj(_[2], _[0], _[1])
```

```
def _create_graph(self, rows: List[List[str]]) -> None:
    for row in rows:
        if self._row_contains_fruit(row):
            vertex1, fruit, vertex2 = row[0], row[1], row[2]
            self.fruits.add(fruit)
        else:
            vertex1, fruit, vertex2 = row[0], '', row[1]
        self._add_edge(vertex1, vertex2, fruit)
        self._add_edge(vertex2, vertex1, fruit)

@staticmethod
def _row_contains_fruit(row: List[str]) -> bool:
    if len(row) == 3:
        return True
    elif len(row) == 2:
        return False
    raise RuntimeError("Row has incorrect number of items")
```

Refaktorizácia 3(backtracking)

Pred refaktorizáciou

```
def backtracking(self, v, hrany, z, prejdene):
    if prejdene >= self.o:
        self.riesenie = z.copy()
        return
    else:
        for _ in self.g[v]:
            if not {(v, _), (_, v)} <= hrany and (self.hrana(_, v) == '' or self.hrana(_, v) not in prejdene) and not
self.riesenie:
                self.backtracking(_, hrany | {(v, _), (_, v)}, z + [_], prejdene | {self.hrana(_, v)})
```

Refaktorizácia 3(backtracking)

Po refaktorizácii

```
def backtracking(self, start_vertex: str, visited_edges: Set[Tuple[str]],
                 path: List[str], collected_fruits: Set[str]) -> None:
    if collected_fruits >= self.fruit_types():
        self.solution = path
    else:
        for adjacent_vertex in self.garden[start_vertex]:
            if self._can_continue(start_vertex, adjacent_vertex, visited_edges, collected_fruits):
                new_visited_edges = visited_edges.union({(start_vertex, adjacent_vertex),
                                                                          (adjacent_vertex, start_vertex)})
                new_path = path + [adjacent_vertex]
                new_collected_fruits = collected_fruits.union(self.edge(adjacent_vertex, start_vertex))
                self.backtracking(adjacent_vertex, new_visited_edges, new_path, new_collected_fruits)

def _can_continue(self, start_vertex: str, end_vertex: str,
                  visited_edges: Set[Tuple[str]], collected_fruits: Set[str]) -> bool:
    if (start_vertex, end_vertex) in visited_edges or self.edge(start_vertex, end_vertex) in collected_fruits \
            or self.solution:
        return False
    return True

def start(self, start_vertex: str) -> List[str]:
    self.solution.clear()
    self.backtracking(start_vertex, set(), [start_vertex], set())
    return self.solution
```