Języki Symboliczne
Projekt

Radosław Żerebiec, gl. 08

1 Temat projektu

Projekt to prosty program do wyszukiwania połączeń kolejowych. Zdefiniowane jest 19 przykładowych miast i przykładowe połączenia między nimi. Program pozwala użytkownikowi tworzyć między nimi połączenia, a następnie szukać tras.

2 Funkcjonalność

- Program składa się z dwóch okien:
  - Pierwsze okno zawiera listy rozwijane z nazwami miejscowości, listę połączonych ze sobą miejscowości, przyciski *"Połącz"* i *"Rozłącz"* oraz zatwierdź. Pozwala na ustawianie siatki połączeń
  - Drugie okno zawiera listy rozwijane z nazwami miejscowości, przycisk *"Szukaj"*, listę rozwijaną z wyborem reprezentacji grafu i przycisk *"Edytuj poł."* pozwalający edytować siatkę połączeń. Pozwala na wyszukiwanie połączeń
- Połączenia między miastami są wyszukowanie przy użyciu algorytmu Breadth-First Search
- Połączenia mogą być reprezentowane przy użyciu macierzy sąsiedztwa albo listy sąsiedztwa

2.1 Z czym były problemy

Miałem problem tworzeniem interfejsu przy użyciu biblioteki Tkinter. Problemy sprawiał mi widget `listbox`, który działał inaczej niż się tego spodziewałem, więc ostatecznie trzymam wyświetlane na liście dane w oddzielnej kolekcji którą modyfikuję w tym samym czasie co zawartość `listboxa`. W ten sposób program działa bez problemów.

Drugim problemem z biblioteką było tworzenie pierwszego okna z drugiego okna (mowa tu o przycisku "Edytuj poł."). Wiem, że da się to osiągnąć w pewien sposób przy użyciu klasy `Toplevel`, w moim przypadku sprawiało to problemy, ostatecznie zdecydowałem się to obejść dodając nową wartość w trybie wyliczeniowym `CloseReason` (`EDIT`), którą sprawdzam w funkcji main po zamknięciu `FindRoutesForm`

2.2 Dodane elementy specjalne

Lambda-wyrażenia

W moim projekcie mam tylko jedno lambda-wyrażenie w funkcji `main`. Nie wydaje mi się, żeby były jeszcze jakieś inne miejsca w których byłaby potrzeba ich użyć albo byłyby przydatne. Nie widzę też sensu umieszczać ich jako akcje użycia przycisków, bo w moim przypadku znacznie wydłuża to linie i zmniejsza czytelność kodu.

List comprehensions

List comprehensions używam bardzo często, zarówno w kodzie testowym jak i produkcyjnym. Wystarczy zajrzeć do modułu `graphs` żeby znaleźć sporo przykładów.

Klasy

1. Przypadki dziedziczenia możemy znaleźć w modułach `graphs` (dziedziczenie po klasie `AbstractGraph`) i explorers (dziedziczenie po klasie `GraphExplorer`)
2. Interfejs użytkownika zawiera się w oddzielnym module (`ui`), w dwóch klasach

Wyjątki

W moim projekcie nie ma zbyt wielu miejsc gdzie mogły znaleźć się wyjątki, jedynym miejscem gdzie rzuciłem wyjątek to funkcja `utils.get_node_id_of_city_by_name`, w której rzucam własny, bardzo prosty wyjątek `CityNotFoundError`. Później łapię go w metodach statycznych klasy `Generators` i ponownie rzucam bardziej ogólne wyjątki `AdjacencyListGenerationError` i `AdjacencyMatrixGenerationError`

Własne moduły

Zarówno kod programu jak i testy są podzielone na moduły w zależności od funkcjonalności (np. `ui` - interfejs użytkownika, `graphs` - grafy i co z nimi związane, `data` - dane, itp.).

Wyróżniające elementy

W wielu miejscach w kodzie wykorzystywałem adnotację `@staticmethod`, która czyni metodę statyczną tak jak w innych obiektowych językach programowania.

Klasy `AbstractGraph` oraz `GraphExplorer` wykorzystują meta klasę `ABCmeta`, która powoduje że klasy naśladują zachowanie klas abstrakcyjnych z innych obiektowych języków programowania (rzucą wyjątek jeżeli klasa - potomek nie będzie posiadała własnych implementacji wszystkich metod abstrakcyjnych oznaczanych adnotacją `@abstractmethod`)

3 Klasy, moduły i samodzielne funkcje zawarte w projekcie

3.1 Moduł `data`

Zawiera listę dostępnych miast (`cities`) oraz listę połączeń początkowych (`initial_connections`)

3.2 Moduł `explorers`

Zawiera klasę abstrakcyjną `GraphExplorer` oraz klasę `BfsExplorer` która po niej dziedziczy. Klasa `GraphExplorer` jest abstrakcją klas przeszukiwujących graf w poszukiwaniu ścieżki z punktu `starting_node` do `ending_node`, natomiast klasa `BfsExplorer` jest jej implementacją dla algorytmu BFS

3.3 Moduł `graphs`

Zawiera klasy powiązane z grafami

Klasa `GenerationError`

Reprezentuje wyjątek zgłoszony przy wykorzystywaniu klasy `Generators`

Klasa `AdjacencyListGenerationError`

Dziedziczy po `GenerationError`. Zgłaszany przez metodę `Generators.generate_adjacency_lists` opakowuje wyjątek zgłoszony przez metodę `pair.to_graph_node_ids()`

Klasa `AdjacencyMatrixGenerationError`

Dziedziczy po `GenerationError`. Zgłaszany przez metodę `Generators.generate_adjacency_matrix` opakowuje wyjątek zgłoszony przez metodę `pair.to_graph_node_ids()`

Klasa `Generators`

Zawiera metody generujące macierz sąsiedztwa lub listy sąsiedztwa na podstawie listy par połączonych ze sobą węzłów

Klasa `NodePair`

Reprezentuje parę połączonych ze sobą na grafie węzłów (z dodatkową funkcjonalnością)

Klasa `AbstractGraph`

Klasa abstrakcyjna zawierająca podstawowe metody jakie powinna mieć reprezentacja grafu

Klasa `MatrixDefinedGraph`

Dzedziczy po `AbstractGraph`. Jest implemetacją grafu reprezentowanego za pomocą macierzy sąsiedztwa

Klasa `ListDefinedGraph`

Dzedziczy po `AbstractGraph`. Jest implemetacją grafu reprezentowanego za pomocą listy sąsiedztwa

3.4 Moduł `main`

Zawiera główną funkcję programu `main` która najpierw tworzy okno konfiguracji połączeń, a następnie okno wyszukiwania połączeń.

3.5 Moduł `ui`

Zawiera klasy graficznego interfejsu użytkownika

Enum `CloseReason`

Powód zamknięcia okna

Klasa `SetupRoutesForm`

Okno konfiguracji połączeń. Pozwala nam przy użyciu parametru `connection_list` dodać wcześniej połączone przystanki.

Klasa `FindRoutesForm`

Okno wyszukiwania połączeń. Pozwala wyszukiwać połączenia i wybrać reprezentację grafu której chcemy używać przy wyszukiwaniu. Dodatkowo mamy w niej dostępny przycisk "Edytuj poł." pozwalający edytować połączenia.

3.6 Moduł `utils`

Zawiera funkcje używane w różnych miejscach w progranie niepowiązane konkretnie z żadnym modułem

Klasa `CityNotFoundError`

Wyjątek zgłaszany przez metodę `get_node_id_of_city_by_name`, gdy miasto o podanej nazwie nie zostanie znalezione

Funkcja `flatmap`

Spłaszcza tablicę dwuwymiarową w jednowymiarową (np. [[1, 2], [3, 4]] $->$ [1, 2, 3, 4])

Funkcja `get_node_id_of_city_by_name`

Pobiera id miasta o podanej nazwie

Funkcja `get_city_name_by_node_id`

Pobiera nazwę miasta o podanym id

Funkcja `print_path`

Wyświetla ścieżkę w postaci "Miasto $0 -> $ Miasto $1 -> \ldots -> $ Miasto n"

4 Testy

4.1 Moduł `alltests`

Moduł którego uruchomienie wykona wszystkie testy jednostkowe

4.2 Klasa `explorerstests.BfsExplorerTests`

Test `test_find_path_direct_matrix`

Sprawdza czy `BfsExplorer` znajduje bezpośrednie połączenie w grafie reprezentowanym przez macierz

Test `test_find_path_direct_list`

Sprawdza czy `BfsExplorer` znajduje bezpośrednie połączenie w grafie reprezentowanym przez listy

Test `test_find_path_1stop_matrix`

Sprawdza czy `BfsExplorer` znajduje połączenie z jednym przystankiem pośrednim w grafie reprezentowanym przez macierz

Test `test_find_path_1stop_list`

Sprawdza czy `BfsExplorer` znajduje połączenie z jednym przystankiem pośrednim w grafie reprezentowanym przez listy

Test `test_find_path_3stops_matrix`

Sprawdza czy `BfsExplorer` znajduje połączenie z trzema przystankami pośrednimi w grafie reprezentowanym przez macierz

Test `test_find_path_3stops_list`

Sprawdza czy `BfsExplorer` znajduje połączenie z trzema przystankami pośrednimi w grafie reprezentowanym przez listy

Test `test_should_not_find_path_matrix`

Sprawdza czy `BfsExplorer` nie znajduje połączenia do miasta bez połączeń z innymi miastami w grafie reprezentowanym przez macierz

Test `test_should_not_find_path_list`

Sprawdza czy `BfsExplorer` nie znajduje połączenia do miasta bez połączeń z innymi miastami w grafie reprezentowanym przez listy

Test `test_find_path_cities`

Sprawdza czy `BfsExplorer` znajduje połączenia między miastami (zdefiniowanymi przez ich nazwy) przy użyciu metody `find_path_cities`.

4.3 Klasa `graphtests.GeneratorsTests`

Zawiera testy metod zawartych w klasie `Generators`

Test `test_generate_adjacency_lists`

Sprawdza czy funkcja `generate_adjacency_lists` prawidłowo generuje listy sąsiedztwa

Test `test_generate_adjacency_matrix`

Sprawdza czy funkcja `generate_adjacency_matrix` prawidłowo generuje macierz sąsiedztwa

4.4 Klasa `graphtests.MatrixDefinedGraphTests`

Test `test_incidence_matrix_generation`

Sprawdza czy metoda `get_incidental_matrix` klasy `MatrixDefinedGraph` prawidłowo generuje macierz incydencji

Test `test_get_node_list_should_return_valid_nodes`

Sprawdza czy metoda `get_node_list` klasy `MatrixDefinedGraph` zwraca prawidłową listę węzłów

Test `test_get_incidental_edges_returns_proper_list`

Sprawdza czy metoda `get_incidental_edges` klasa `MatrixDefinedGraph` zwraca prawidłowe listy krawędzi incydentalnych

Test `test_get_nodes_connected_by_edge_returns_proper_sequence`

Sprawdza czy metoda `get_nodes_connected_by_edge` klasy `MatrixDefinedGraph` zwraca prawidłowe listy węzłów

Test `test_get_neighbour_nodes_returns_proper_nodes`

Sprawdza czy metoda `get_neighbour_nodes` klasy `MatrixDefinedGraph` zwraca prawidłową listę sąsiadów dla danego węzła

4.5 Klasa `graphtests.ListDefinedGraphTests`

Test `test_get_node_list_should_return_valid_nodes`

Sprawdza czy metoda `get_node_list` klasy `ListDefinedGraphTests` zwraca prawidłową listę węzłów

Test `test_get_incidental_edges_returns_proper_list`

Sprawdza czy metoda `get_incidental_edges` klasa `ListDefinedGraphTests` zwraca prawidłowe listy krawędzi incydentalnych

Test `test_get_nodes_connected_by_edge_returns_proper_sequence`

Sprawdza czy metoda `get_nodes_connected_by_edge` klasy `ListDefinedGraphTests` zwraca prawidłowe listy węzłów

Test `test_get_neighbour_nodes_returns_proper_nodes`

Sprawdza czy metoda `get_neighbour_nodes` klasy `ListDefinedGraphTests` zwraca prawidłową listę sąsiadów dla danego węzła

4.6 Klasa `utilstests.UtilsTests`

Testuje funkcje z modułu `utils`

Test `test_flatmap`

Sprawdza czy funkcja `flatmap` odpowiednio spłaszcza listę dwuwymiarową

Test `test_found_city`

Sprawdza czy funkcja `get_node_id_of_city_by_name` znajduje poprawne id miasta o podanej nazwie

Test `test_city_not_found`

Sprawdza czy funkcja `get_node_id_of_city_by_name` nie znajduje miasta którego nie ma w danych i czy rzucany jest odpowiedni wyjątek

Test `test_get_city_name_by_node_id`

Sprawdza czy funkcja `get_city_name_by_node_id` zwraca poprawną nazwę miasta o podaym id

Test `test_get_city_name_by_node_id_fail_if_bad_id`

Sprawdza czy funkcja `get_city_name_by_node_id` nie znajduje miasta którego nie ma w danych i czy rzucany jest odpowiedni wyjątek

Test `test_print_path`

Sprawdza czy funkcja `print_path` zwraca poprawny ciąg znaków

Test `test_print_empty_path`

Sprawdza czy funkcja `print_path` zwraca pusty ciąg znaków jeżeli podana ścieżka jest pusta