Całkowanie symboliczne - zapis grafów w formie macierzy sąsiedztwa i ich graficzna reprezentacja.

Łukasz Pluszyński

Wstęp - celem projektu jest stworzenie zestawu funkcji rozszerzających projekt Maksymiliana Siemaba o zapis grafów w formie macierzy sąsiedztwa oraz ich graficzną reprezentację. Program został przygotowany w środowisku Python Anaconda 3.8 i wykorzystuje następujące biblioteki zewnętrzne: numpy, maplotlib, networkx i ast.

Projekt jest dostępny na grupowym githubie: https://github.com/Void-Caller/Projekt_LAdI

Funkcje:

integrate_to_graph(function, variables):

Całkuje podany wielomian przez podane zmienne. Zwraca scałkowane równanie jako graf w postaci tablicowej. Wykorzystuje algorytm całkowania stworzony przez Maksymiliana i przyjmuje funkcje oraz zmienne w ten sam sposób co jego funkcje.

equation - wielomian w postaci ciągu znaków
variables - lista zmiennych w formie tablicy

integrate_from_File(input="input.txt"):

Odczytuje podane równanie w postaci wielomianu, a następnie przetwarza je i zwraca graf w formie tablicowej. Stosuje algorytmy z biblioteki Maksymiliana plus funkcję integrate_to_graph.

input - plik wejściowy (domyślnie input.txt)

count_vertices(graph):

Zwraca liczbę wierzchołków grafu. Przyjmuje graf w postaci tablicowej i przechodzi po jego wierzchołkach zliczając je po drodze.

graph - graf, którego wierzchołki zliczamy

map_graph(matrix, graph, row, col):

Rekurencyjne uzupełnianie macierzy sąsiedztwa grafu nad przekątną. Algorytm bada drugi i, jeśli takowy istnieje, trzeci element tablicy. Jeśli element jest tablicą, to oznacza, że jest wierzchołkiem posiadającym liście i trzeba dla niego wywołać tę samą funkcję ze zmienionymi parametrami.

Algorytm wykorzystuje to, że każdy wierzchołek grafu zapisany jako tablica posiada co najmniej jednego potomka i nie więcej niż dwóch, żeby zmniejszyć ilość wykonywanych testów na istnienie potomków.

Zmienne row i col są wykorzystywane do przekazywania informacji o aktualnie sprawdzanym wierzchołku do kolejnych wywołań funkcji. Przy pierwszym wywołaniu row i col równe są zeru, co oznacza rozpoczęcie od korzenia grafu.

Poprawnie wykonana funkcja zwraca zero.

matrix - macierz do uzupełnienia
graph - graf w postaci tablicy tablic
row - wiersz
col - kolumna

map_graph_labels(labels, graph, row, col):

Rekurencyjne przypisywanie oznaczeń do wierzchołków grafu na rysunku. Oparty na identycznym algorytmie co funkcja map_graph. Główną różnicą jest zastąpienie procesu uzupełniania macierzy sąsiedztwa nad przekątną procesem budowania słownika przyporządkowującego symbole równania ich indeksom na wektorze przypisanym do macierzy sąsiedztwa.

Słownik posiada strukturę indeks: znak. Indeksy to typ int, znaki typ char.

labels - słownik etykiet
graph - graf w postaci tablicy tablic
row - wiersz
col - kolumna

map_full(graph):

Funkcja tworząca macierz sąsiedztwa podanego grafu na podstawie funkcji map_graph i operacji na macierzach biblioteki numpy. Zwraca macierz sąsiedztwa.

Algorytm:

1. Zlicz wierzchołki grafu funkcją graph_count, jako zmienną graph_count.
2. Za pomocą bibliotek numpy utwórz wypełniony zerami array dwuwymiarowy graph_count na graph_count, który będzie naszą macierzą.
3. Wykonaj map_graph(graph_matrix, graph, 0, 0) na wprowadzonym grafie i utworzonej macierzy.
4. Stwórz tymczasową macierz będącą transponowaną kopią utworzonej w podpunkcie drugim macierzy.
5. Zsumuj macierze dzięki wbudowanej obsłudze operatorów numpy.

graph - graf w postaci tablicy tablic

print_graph(graph, output='test.pdf'):

Funkcja rysująca graf i zapisująca go w pliku. Oparta na założeniach poprzednich funkcjach.

Algorytm:

1. Utwórz pusty słownik labels, który będzie zapisywał symbole wierzchołków grafów i ich kolejność przy w macierzy sąsiedztwa.
2. Utwórz macierz sąsiedztwa za pomocą funkcji map_full.
3. Zapełnij słownik labels funkcją map_graph_labels.
4. Otwórz dokument funkcją PdfPages
5. Przygotuj graf do rysunku za pomocą networkx.from_numpy_matrix wywołanej na macierzy sąsiedztwa.
6. Utwórz słownik pozycji wierzchołków na rysunku funkcją networkx.spring_layout dla przygotowanego grafu.
7. Utwórz tablicę kolorów node_colors. Pierwszy element oznaczający korzeń grafu jest ma kolor 'r' - czerwony, podczas, gdy reszta otrzymuje kolor blado niebieski '#1f77b4'
8. Narysuj graf z podanymi parametrami funkcją networkx.draw
9. Zapisz graf do pliku.
10. Wyczyść pamięć podręczną plotera funkcją maplotlib.clf
11. Zamknij plik zapisu.

graph - rysowany graf w postaci tablicowej
output - string z nazwą pliku zapisu

save_matrix(matrix, labels, output='matrix.txt'):

Zapisuje macierz sąsiedztwa w pliku tekstowym w formie czytelnej dla ludzi. Pierwszą linią kodu jest słownik symboli zapisany w formie string'a. Pozostałe linie odpowiadają kolejnym kolumnom macierzy sąsiedztwa sformatowanym jako: np.array2string(matrix)[1:-1], gdzie splicing [1:-1] usuwa trudne w parsowaniu nawiasy zaczynające i kończące nadrzędną tablicę macierzy.

matrix - macierz do zapisu
labels - etykiety wierzchołków
output - nazwa pliku zapisu

load_matrix(input='matrix.txt'):

Wczytuje macierz sąsiedztwa z pliku i zwraca ją wraz z etykietami w formie krotki.
Algorytm:

1. Przygotowanie pustej tablicy adj_matrix do zapisu macierzy sąsiedztwa.
2. Przygotowanie pustego string'a label do zapisuje linii tekstu ze słownikiem oznaczeń.
3. Otwarcie pliku.
4. Wczytanie pierwszej linii, słownika oznaczeń, i zapisanie jej do labels.
5. Czytanie reszty pliku linia po linii. Każda linia jest w pierwszej kolejności pozbawiana znaków, które utrudniałyby jej czytanie jak: '[', ']', ' ' i '\n'.
6. Następnie sformatowana linia jest zamieniana tablicę znaków matrix.
7. Na podstawie tablicy matrix tworzona jest tablica liczb mat. Stosowany jest mechanizm list comperhension.
8. Tablica mat jest dodawana do adj_matrix.
9. Po przeczytaniu ostatniej linii pliku program zamyka go i zwraca dane w formie krotki zawierającej odpowiednio: adj_matrix przekonwertowany na typ numpy.array i słownik utworzony przez funkcję ast.literal_eval wykonaną na label.

input - plik zapisu