algograf

Lab 7: Biblioteka NetworkX, planarność, przepływy oraz SAT-2CNF

Głównym celem laboratorium jest zapoznanie się z bilbioteką NetworkX, wspomagającą wykorzystanie algorytmów grafowych w języku Python. Najpierw wykorzystamy jej możliwości do testowania planarności grafów oraz obliczania maksymalnego przepływu, a następnie zajmiemy się algorytmem znajdującym wartościowania spełniające formuły logiczne postaci 2CNF.

Zadanie 1 (testowanie planarności)

Wykorzystamy bibliotekę NetworkX do testowania, czy dany graf jest planarny. NetworkX ma wbudowany test planarności, więc przede wszystkim musimy wczytać odpowiedni graf, skonstruować go jako obiekt NetworkX i wywołać wbudowaną funkcję.

Elementarne operacje na grafach

```
import networkx as nx            # standardowy sposób importowania biblioteki

# tworzenie grafu
G = nx.Graph()                   # stwórz pusty graf nieskierowany
G.add_node(1)                    # dodaj wierzchołek 1 (wierzchołkami może być cokolwiek h
G.add_nodes_from([2,3])          # dodaj wierzchołki z listy (dowolnego iterowalnego konte
G.remove_node(1)                 # usuń wierchołek 1
G.remove_nodes_from([2,3])       # usuń wierzchołki z listy
G.add_edge(1,2)                  # dodaj krawędź między wierzchołkami 1 i 2
G.add_edges_from([(1,3),(2,3)])  # dodaj krawędzie z listy (iterowalnego kontenera)
G.remove_edge(1,2)               # usuń krawędź
G.remove_edges_from([(1,2),(1,3)])  # usuń krawędzie z listy (iterowalnego kontenera)

# odczytywanie podstawowych informacji o grafie
G.number_of_nodes()              # liczba wierzchołków
G.number_of_edges()              # liczba krawędzi
G.nodes                          # wierzchołki grafu
G.edges                          # krawędzie grafu
G.adj[1]                         # sąsiedzi wierzchołka 1
G[1]                             # sąsiedzi wierzchołka 1
G[1][2]                          # dostęp do krawędzi {1,2} (można jej dodawać atrybuty)
G.has_node(1)                    # czy istnieje wierzchołek 1?
G.has_edge(1,2)                  # czy istnieje krawędź {1,2}?
```

Testowanie planarności

Testowanie planarności sprowadza się do wywołania jednej funkcji:

```
from networkx.algorithms.planarity import check_planarity

check_planarity(G)    # czy graf jest planarny? zwraca parę, której pierwszy element to oc
```

Zadanie 2 (maksymalny przepływ)

W tym zadaniu wykorzystamy bibliotekę NetworkX do rozwiązania zadania z Lab 2, czyli do znalezienia maksymalnego przepływu w grafie.

Grafy skierowane

W Lab 2 grafy, które wyrzystywaliśmy były skierowane i takie należy użyć teraz.

```
import networkx as nx

G = nx.DiGraph()           # stwórz pusty graf skierowany

# operacje dodawania/usuwania/odczytywania wierzchołków i krawędzi są takie
# same jak dla grafów nieskierowanych
```

Znalezienie maksymalnego przepływu

Do znalezienia maksymalnego przepływu służy funkcja:

```
from networkx.algorithms.flow import maximum_flow

maximum_flow( G, s, t )     # znajdź maksymalne przepływ między wierzchołkami s i t grafu
                            # przepustowość krawędzi jest ustawiana jako jej atrybut 'capa
                            # zwraca parę (value,flow) gdzie value to wartość przepływu a
                            # flow to słownik mówiący ile przepływu płynie którą krawędzią

G[1][4]['capacity'] = 7     # ustaw atrybut `capacity` krawędzi (1,4) w grafie G
```

Zadanie 3 (SAT-2CNF)

W tym zadaniu naszym celem jest zaimplementowanie algorytmu sprawdzającego czy dana formuła logiczna w postaci 2CNF (tj. w postacI koniunkcyjnej normalnej z najwyżej dwoma

zmiennymi na klauzulę) jest spełnialna i wypisującego spełniające wartościowanie zmiennych (o ile istnieje).

Problem SAT-2CNF

Formuła w postaci 2CNF składa się z koniunkcji klauzul, a każda klauzula to alternatywa dwóch literałów (czyli zmiennych lub ich negacji). Poniżej mamy przykładowe formuły `F` i `G`:

```
F = (x or y) and (-x or z) and (-y or -z)
G = (x or y) and (-x or y) and (x or -y) and (-x or -y)
```

Formuła `F` jest spełnialna. Wystarczy przyjąć:

```
x = True
y = False
y = True
```

Z kolei formuła `G` jest niespełnialna (co widać, gdyż zawiera wszystkie możliwe kombinacje negacji/braku negacji pary zmiennych `x` i `y`).

Algorytm

Algorytm sprawdzający spełnialność formuły w postaci 2CNF działa następująco.

Krok 1 (budowa grafu implikacji). Budujemy graf skierowany *G*, w którym wierzchołkami są literały (czyli zarówno zmienne jak ich negacje) a krawędzie odpowiadają klauzulom. Konkretnie, jeśli w formule jest klauzula *(x or y)*, to w grafie dodajemy krawędzie skierowane z *-x* do *y* oraz z *-y* do *x* (odpowiadają one implikacjom *(-x => y)* oraz *(-y => x)*, które są równoważne alternatywie *(x or y)*).

Krok 2 (testowanie spełnialności). Obliczamy silnie spójne składowe grafu *G*. Jeśli jakaś *zmienna x* i jej negacja *-x* znajdują się w tej samej silnie spójnej składowej, to formuła jest niespełnialna. W przeciwnym razie formuła jest spełnialna.

Krok 3 (konstrukcja wartościowania spełniającego). Jeśli formuła jest spełnialna, to budujemy graf *H* silnie spójnych składowych grafu *G* (tj. każda silnie spójna składowa grafu *G* jest wierzchołkiem grafu *H*; jeśli *G* zawiera krawędź z jakiegoś wierzchołka *u* w silnie spójnej składowej *U* do wierzchołka *v* w innej silnie spójnej składowej *V*, to we *H* mamy krawędź z *U* do *V*). Wiadomo, że *H* jest dagiem (acyklicznym grafem skierowanym). Sortujemy *H* topologicznie i przeglądamy spójne składowe w uzyskanym porządku i wykonujemy następującą operację:

- zmienne/negacje zmiennych w rozważanej silnie spójnej składowej otrzymują wartość `False` (o ile już nie dostały wcześniej wartości `True`) (wiadomo, że dla każdej silnie spójnej składowej *U* istnieje silna spójna składowa *-U*, która zawiera dokładnie te same literały, ale zanegowane)

W efekcie powstaje wartościowanie zmiennych spełniające wejściową formułę.

Weryfikacja wartościowania

Proszę zaimplementować sprawdzanie, że uzyskane wartościowanie faktycznie spełnia daną formułę.

Testowanie rozwiązania

Proszę przetestować zaimplementowany algorytm na przykładowych danych testowych (patrz niżej). Proszę także zaimplementować własną funkcję sprawdzającą, czy obliczone wartościowanie faktycznie spełnia formułę.

Przydatne fragmenty kodu

Wczytywanie formuły w postaci 2CNF. Aby wczytac formułę można wykorzystać funkcję `loadCNFFormula` z biblioteki `dimacs`.

```
from dimacs import *

(V,F) = loadCNFFormula( nazwa )   # wczytaj formułę z pliku `nazwa`
                                  # zwraca maksymalny numer zmiennej V oraz opis formuły F

print(F)                          # da na przykład [[-1,2],[1,3],[-2,-3]]
                                  # dla formuły (-x_1 or x_2) and (x_1 or x_3) and (-x_2 o
```

Konstrukcja grafu. Odpowiedni graf skierowany należy zbudować korzystając z klas/funkcji opisanych w poprzednich zadaniach.

Silnie spójne składowe. Silnie spójne składowe oblicza funkcja `networkx.algorithms.components.strongly_connected_components`, która zwraca generator zbiorów składających się z wierzchołków w danej silnie spójnej składowej (generator można w naturalny sposób użyć w pętli for, tak jakby to była lista).

```
from networkx.algorithms.components import strongly_connected_components

SCC = strongly_connected_components(G)    # policz silnie spójne składowe grafu G

# wypisz zawartość składowych
t = 0
for S in SCC:
  print("Silnie spojna składowa", t, "zawiera wierzcholki")
  for v in S:
```

```
      print("  ",v)
    t += 1
```

Sortowanie topologiczne. Sortowanie topologiczne realizuje funkcja `networkx.algorithms.dag.topological_sort`:

```
from networkx.algorithms.dag import topological_sort

O = topological_sort(H)          # posortuj topologicznie wierzchołki grafu skierowanego H

# wypisz wierzchołki w kolejności topologicznej (krawędzie tylko "z lewej na prawą")
for v in O:
  print(v)
```

Proponowana kolejność prac

- Uruchom `python3`, zaimportuj bibliotekę NetworkX, stwórz graf, dodaj parę wierzchołków i krawędzi i sprawdź jak biblioteka się zachowuje (np. jaki efekt daje dodanie krawędzi do nieistniejącego wierzchołka?)
- Zaimplementuj testowanie planarności
- Zaimplementuj obliczanie maksymalnego przepływu
- Zastanów się po co były wszystkie poprzednie laboratoria, skoro można wywołać jedną funkcję?
- Zaimplementuj rozwiązanie problemu SAT-2CNF
  - wczytaj formułę
  - zbuduj graf implikacji
  - oblicz silnie spójne składowe
  - sprawdź czy zmienna i literał nie są w tej samej spójnej składowej
  - zaimplementuj odczytywanie wartościowania spełniającego formułę

Pomocne pliki

W ramach laboratorium należy wykorzystać:

- dimacs.py – wczytywanie grafów
- graphs-lab2.zip – grafy testowe w katalogu `flow`
- graphs-lab6.zip – grafy do testowania planarności
- sat.zip – przykładowe instancje SAT-2CNF