Laboratoria Algorytmów i Struktur Danych

Projekt 3 - opis i wymagania

Sortowanie topologiczne grafy

Prowadzący: **Dominik Piotr Witczak**

Termin oddania projektu: **03.maj**

Termin oddania sprawozdania: **10.maj**

POLITECHNIKA POZNAŃSKA

1 | Tworzenie grafu

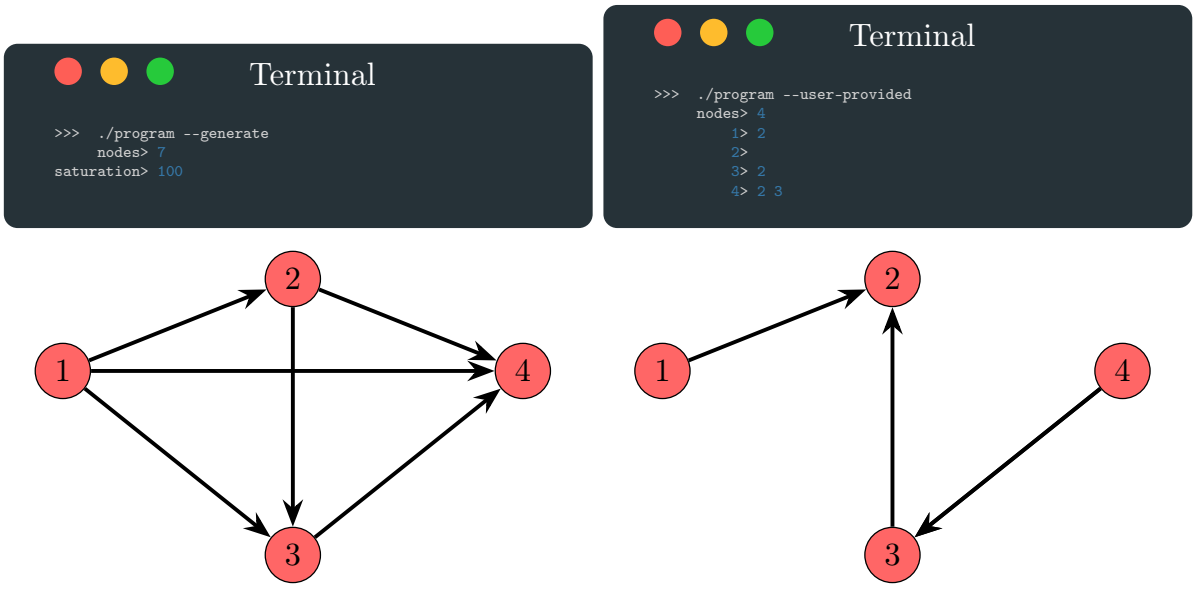
Program ma obsługiwać dwa tryby wprowadzenia grafu:

1. **Generacja grafu** - W przypadku uruchomienia programu z argumentem '–generate'. Wygeneruj spójny skierowany graf acykliczny o *nodes* wierzchołkach oraz nasyceniu *saturation*. O wartości *nodes* i *saturation*, zapytaj użytkownika, wyświetlając odpowiedni znak zachęty.

   Najłatwiej jest utworzyć graf acykliczny skierowany poprzez wypełnienie odpowiednią liczbą jedynek górnego trójkąta macierzy sąsiedztwa.

2. **Ładowanie grafu** - W przypadku uruchomienia programu z argumentem '–user-provided' graf podany zostanie przez użytkownika w formie listy następników. Wpierw zapytaj o liczbę wierzchołków, następnie po kolei pytaj o następników kolejnych wierzchołków.

   Nie musisz sprawdzać czy wprowadzony graf jest acykliczny.



(a) Generowanie grafu (b) Wczytanie grafu

Program powinien obsługiwać wprowadzanie danych za pomocą **heredoc** (skrót od *'here document'*). **heredoc** jest rozwinięciem **herestring**, którego używaliśmy do tej pory i jego składnia prezentuje się następująco:

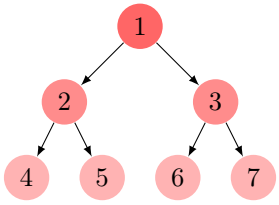
2 | Odporność na błędy

Każdy program powinien być - *w miarę możliwości* - odporny na błędy ludzkie. Nie inaczej jest w tym przypadku. Przy sprawdzaniu waszych programów postaram się wprowadzić różne błędne dane. Jeśli dane będą błędne, program ma wyświetlić odpowiedni komunikat i zakończyć się.

Program nie musi sprawdzać czy wprowadzony graf jest acykliczny.

3 | Operacje na grafach

Waszym zadaniem jest zaimplementować szereg operacji na grafach - na potrzeby wizualizacji użyjmy drzewa - jako przykładowego grafu acyklicznego.

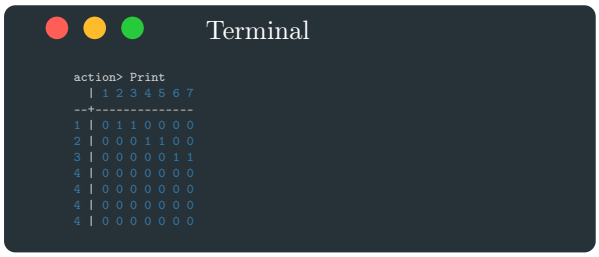


Rysunek 2: Przykładowy graf (drzewo)

1. **Wybranie reprezentacji** grafu - poprzez znak zachęty wyświetlony po uruchomieniu programu (*'matrix'*, *'list'*, *'table'*)

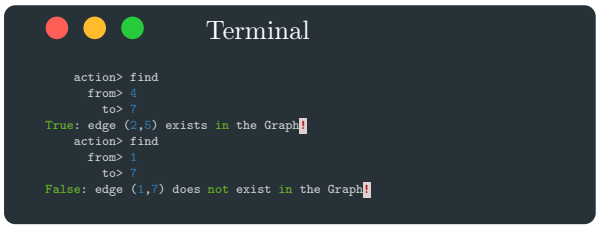
Rysunek 3: Interfejs **wybieranie reprezentacji**

2. **Wypisanie** grafu - w wybranej reprezentacji



Rysunek 4: Interfejs **wypisywania**

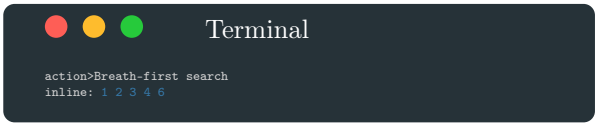
3. **Wyszukanie krawędzi** grafu



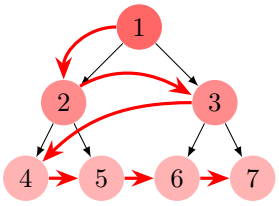
Rysunek 5: Interfejs - przeszukiwanie **wszerz**

4. **Przechodzenie wszerz** grafu

Sposób dostępu do elementów grafu powinien odpowiadać wybranej przez użytkownika metodzie reprezentacji.



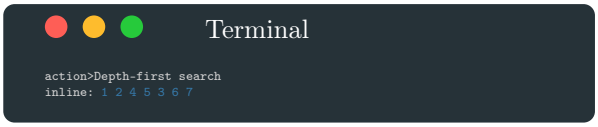
Rysunek 6: Interfejs - przeszukiwanie **wszerz**



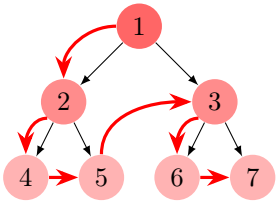
Rysunek 7: Wizualizacja przeszukiwania **wszerz**

5. **Przechodzenie w głąb** grafu

Wskazówka: proponuję przysłać ***funkcję dostępu do sąsiada danego węzła*** *jako argument funkcji - unikniecie zbędnego powielania kodu*

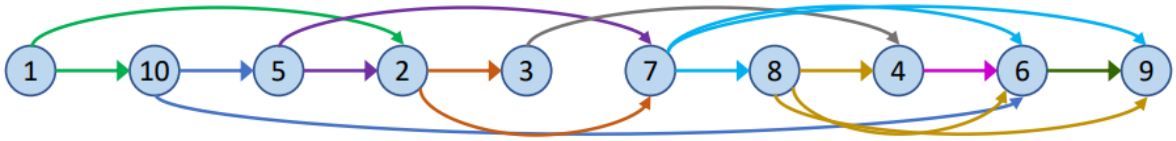


Rysunek 8: Interfejs - przeszukiwanie **w głąb**



Rysunek 9: Wizualizacja przeszukiwania **w głąb**

3.1 | Sortowanie topologiczne grafu



Rysunek 10: Wizualizacja sortowania topologicznego

3.1.1 | Wprowadzenie

Sortowanie topologiczne grafu skierowanego jest to takie liniowe uporządkowanie wierzchołków że

Jeśli w grafie jest łuk z a do b, to a występuje przed b.
inaczej: Jeśli (a,b) to a < b

Dość łatwo zauważyć że taka zależność jest możliwa tylko dla skierowanych grafów acyklicznych (ang. **DAG**) - stosując dowód przez zaprzeczenie:

Dowód. Załóżmy że graf G ma cykle i istnieje jego topologiczne uporządkowanie.
Wybierzmy dowolny cykl *(a,b) (b,c) ... (n, a)*, można zauważyć że:
sortowanie topologiczne cyklu wymaga od nas aby a < b < c ... < n < a.
Ta nierówność jest sprzeczna, zatem założenie że G ma cykle jest błędne.
Zatem jeśli graf ma topologiczne uporządkowanie to jest acykliczny.

□

3.1.2 | Zadanie

Waszym zadaniem jest zaimplementowanie dwóch algorytmów:

Algorytm Kahna (oparty na BFS)

```
function Kahn():
    while S is not empty
        Remove a node n from S
        Add n to L
        for each node m with an edge e
          from n to m do
            Remove edge e from the
              graph
            if m has no other incoming
              edges then
                Insert m into S
    if graph has edges then
        error (graph has at least one cycle)
    else
        return L (a topologically sorted
          order)
```

Algorithm 1: Topological Sorting

Algorytm Tarjana (oparty na DFS)

```
function Tarjan():
    for each node n do
        n ← unmarked
    while
      exists node n without permament mark
        visit(n)
    print L
function visit(n):
    0
if n is permanent then
    return
if n is temporary then
    error (graph has at least one cycle)
n ← temporary
for
  each node m with an edge from n to m
  do
    visit(m)
n ← permanent
append nto L
```

Algorithm 2: Algorytm Tarjana

4 | Wymagania do oceny

Zadanie 1 - Tworzenie		Zadanie 2 -	Zadanie 3 - Operacje			Zadanie 4 - Sortowanie	
Generacja	Wczytanie	Odporność na błędy	Wypisanie	BFS	DFS	Kahna	Tarjana
2	1	2	1	2	2	**5**	**5**

Tabela 1: Szczegółowa Punktacja

W sumie 20 punktów. Dodatkowe punkty, poza skalą: 1 pkt za export grafu do tickzpicture. Do 3 punktów za staranność wykonania projketu (np. czystość kodu, ekspertyza w języku czy używanie gita).

2	3	3.5	4	4.5	5	5.5
do 11	od 12	od 14	od 16	od 18	20	23-24

Tabela 2: Szczegółowa Punktacja

5 | Benchmark implementacji

Ponownie przygotuję dla was generator danych wejściowych, które zapiszę do pliku. Przygotuję też pliki zawierające polecenia które można przekazać do programu. Przygotuję dla was benchmark który zmierzy czas działania i zajętość pamięciową waszych programów do operacji:

(a) wyszukanie krawędzi.

(b) czas działania algorytmu sortowania topologicznego - metodą Kahna,

(c) czas działania algorytmu sortowania topologicznego - metodą Tarjana,

6 | Wymagania do sprawozdania

[3 pkt] Zaprezentuj swój program (3 screeny z wygenerowaniem grafu i wypisaniem reprezentacji dla wszystkich 3 reprezentacji, najlepiej jako sąsiadujące kolumny).

[2 pkt] Pokaż wizualizację utworzonych grafów (tickzpicture, albo inne narzędzie).

[5 pkt] Wykonaj 3 wykresy (jeden wykres dla każdej z operacji: wyszukanie krawędzi, algorytm Kahna, algorytm Tarjana) t=f(n) zależności czasu obliczeń t od liczby n elementów w drzewie. Na każdym wykresie przedstaw 3 krzywe - po jednej krzywej dla reprezentacji.

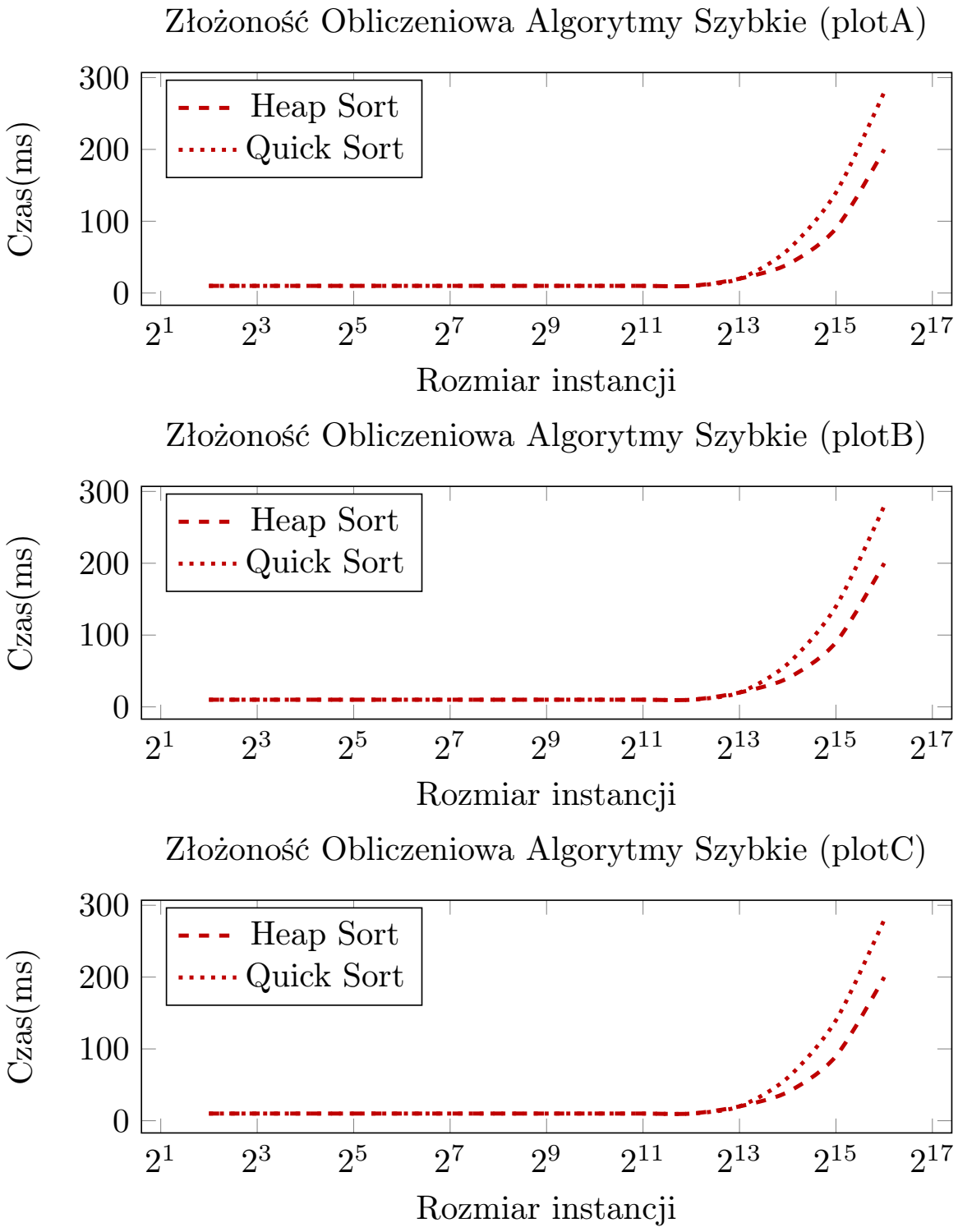
Dodatkowy: [1 pkt] Wykonaj sprawozdanie w LaTeXu :)

Dodatkowy: [1 pkt] Przedstaw wykresy w takim ułożeniu jak na rysunku 11

W sumie 10 punktów.

2	3	3.5	4	4.5	5	5.5
do 5	6	7	8	9	10	12

Tabela 3: Szczegółowa Punktacja



Rysunek 11: Ilustracja podzielona na 3 podwykresy