

Algorytmy optymalizacji dyskretnej 2026 / 27

LABORATORIUM 1

Podstawowe algorytmy grafowe

Termin realizacji: **ostatnie zajęcia przed 28.10.2026 r.**

Waga listy: $w_1 = 0,05$

Warunek zaliczenia listy: realizacja zadań 0, 1, 2 i 3

Zadanie 0. Zapoznaj się z artykułem [Joh02] na temat eksperymentalnego badania algorytmów (dostępnym m.in. na stronie <http://dimacs.rutgers.edu/archive/Challenges/TSP/papers/expguide.pdf>). Podczas testowania algorytmów i przeprowadzania eksperymentalnej analizy ich własności stosuj się do omówionych tam dobrych praktyk i staraj się unikać wspomnianych pułapek.

W kontekście zagadnień, które będą omawiane na kursie (oraz przyszłych zadań na laboratorium), warto przeczytać także rozdział 18 (*Computational Testing of Algorithms*) z podręcznika [AMO93].

W zadaniach 1–4 wybierz odpowiednią reprezentację grafów, pozwalającą na uzyskanie możliwie efektywnych implementacji. Przetestuj swoje implementacje dla przykładów wskazanych w zadaniach, obejmujących m.in. grafy z paczki `aod_testy1.zip` dostępnej na stronie kursu (pliki z definicją grafów o liczbie wierzchołków rzędu 10^k dla $1 \leq k \leq 6$).

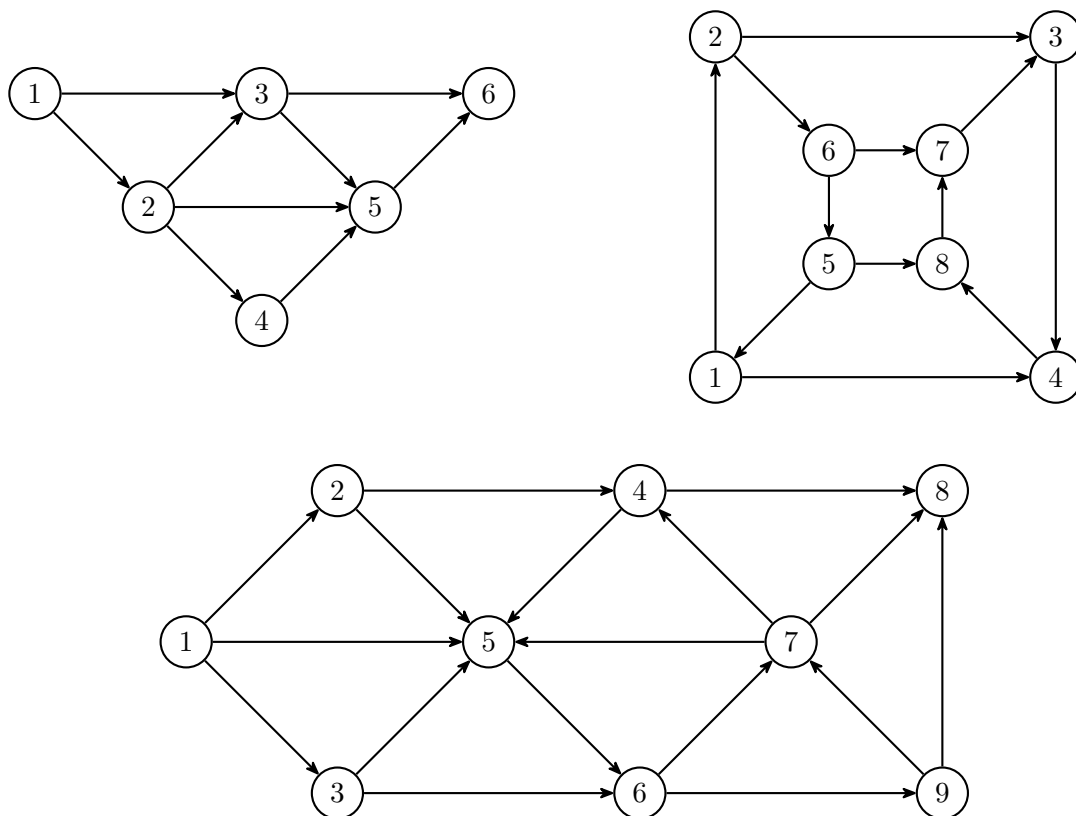
W trakcie oddawania listy należy m.in. zaprezentować otrzymane rezultaty wcześniej przeprowadzonych testów, obejmujące zwrócone wyniki (zgodnie ze specyfikacją zadania) oraz czas działania programów dla wskazanych danych testowych.

Definicja grafu $G = (V, E)$ powinna być przekazywana na standardowym wejściu / wczytywana z pliku w następujący sposób (w osobnych liniach kolejno podane są):

- jednoliterowa flaga mówiąca, czy graf jest skierowany (D), czy nieskierowany (U),
- liczba wierzchołków $n = |V|$ (przyjmujemy, że wierzchołki są etykietowane kolejnymi liczbami naturalnymi ze zbioru $\{1, \dots, n\}$),
- liczba krawędzi $m = |E|$,
- kolejno m definicji krawędzi postaci $u \ v$.

Zadanie 1. [1,5 pkt] Zaimplementuj algorytmy przeszukiwania grafów wgłąb i wszerz (DFS i BFS; patrz np. rozdział 3.4 w [AMO93], rozdziały 3.2, 3.3 i 4.2 w [DPV06] lub rozdziały 20.2 i 20.3 w [CLRS22]). Program na wyjściu powinien wypisywać kolejność, w której wierzchołki były odwiedzane oraz – po podaniu odpowiedniego parametru wywołania – zwracać drzewo przeszukiwania (DFS/BFS tree). Program powinien obsługiwać przypadki grafów skierowanych i nieskierowanych.

Dane testowe: grafy z rysunku 1 (w wersji skierowanej i nieskierowanej); własny przykład grafu skierowanego i nieskierowanego z $10 \leq n \leq 100$ wierzchołkami (można poszukać czegoś np. w [DPV06] lub wygenerować przy użyciu dostępnych narzędzi); ewentualne inne testy prowadzącego laboratorium.



Rysunek 1: Grafy testowe do zadania 1.

Zadanie 2. [1 pkt] Zaimplementuj algorytm sortowania topologicznego dla grafów skierowanych z wykrywaniem istnienia skierowanego cyklu (patrz np. rozdział 3.4 w [AMO93], rozdział 3.3.2 w [DPV06] lub rozdział 20.4 w [CLRS22]). Program na wyjściu powinien odpowiadać, czy graf zawiera skierowany cykl. Jeżeli graf jest acykliczny, a liczba wierzchołków $n \leq 200$, to program powinien również wypisywać listę wierzchołków w porządku topologicznym.

Dane testowe: własny przykład grafu acyklicznego i ze skierowanym cyklem z $10 \leq n \leq 100$ wierzchołkami; wszystkie grafy z folderu 2 z paczki `aod_testy1.zip` (12 plików); ewentualne inne testy prowadzącego laboratorium.

Zadanie 3. [1 pkt] Zaimplementuj algorytm, który dla podanego na wejściu grafu skierowanego $G = (V, E)$ zwróci jego rozkład na silnie spójne składowe (patrz np. rozdział 3.4 w [DPV06] lub 20.5 w [CLRS22]). Algorytm powinien działać w czasie $O(|V| + |E|)$. Program na wyjściu powinien wypisywać liczbę silnie spójnych składowych oraz liczbę wierzchołków w każdej z nich. Jeśli $n \leq 200$, to program powinien również wypisywać listę wierzchołków w każdej ze składowych.

Dane testowe: własny przykład grafu silnie spójnego oraz spójnego z > 1 silnie spójną składową z $10 \leq n \leq 100$ wierzchołkami; wszystkie grafy z folderu 3 z paczki `aod_testy1.zip` (6 plików); ewentualne inne testy prowadzącego laboratorium.

Zadanie 4. [1,5 pkt] Zaimplementuj efektywny algorytm, który dla podanego na wejściu grafu $G = (V, E)$ (skierowanego lub nieskierowanego, niekoniecznie spójnego) zwraca informację, czy G jest grafem dwudzielnym. Jeśli tak, to dla $n \leq 200$ program powinien również wypisywać rozbięcie V na dwa podzbiory V_0 i V_1 takie, że jeśli $(u, v) \in E$ (odpowiednio, $\{u, v\} \in E$ dla grafów nieskierowanych), to $u \in V_i$ i $v \in V_{1-i}$, $i \in \{0, 1\}$. Jaką złożoność ma zaimplementowany algorytm?

Dane testowe: własne przykłady grafów dwudzielnego oraz niedwudzielnego (skierowanego i nieskierowanego) z $10 \leq n \leq 100$ wierzchołkami; wszystkie grafy z folderu 4 z paczki `aod_testy1.zip` (24 pliki); ewentualne inne testy prowadzącego laboratorium.

Literatura

- [AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., USA, 1993.
- [CLRS22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 4th edition, 2022.
- [DPV06] Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill, Inc., USA, 1st edition, 2006.
- [Joh02] David S. Johnson. A Theoretician’s Guide to the Experimental Analysis of Algorithms. In D.S. Johnson M.H. Goldwasser and C.C. McGeoch, editors, *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, pages 215–250. American Mathematical Society, January 2002.