

Klasy grafowe

Biblioteka oferuje cztery klasy odpowiadające podstawowym typom grafów:

- Graph – graf prosty,
- DiGraph – graf skierowany,
- Graph<T> – graf ważony; T oznacza typ wag krawędzi (najczęściej int lub double),
- DiGraph<T> – graf ważony skierowany; T oznacza typ wag krawędzi (najczęściej int lub double).

Każda z powyższych klas zawiera konstruktor z dwoma argumentami:

- vertices – liczba wierzchołków tworzonego grafu,
- representation – reprezentacja grafu (ten argument można pominąć, wtedy tworzony jest graf w domyślnej reprezentacji).

Reprezentację grafu rozumiemy jako obiekt klasy implementującej interfejs IGraphRepresentation. Aktualnie dostępne są dwie reprezentacje:

- MatrixGraphRepresentation – reprezentacja przez macierz sąsiedztwa,
- DictionaryGraphRepresentation – reprezentacja przez listy sąsiedztwa implementowane przez tablice hashujące (wbudowany typ Dictionary).

Reprezentację grafu można pobrać przez atrybut Representation (tylko do odczytu).

Przykłady:

```
Graph G = new Graph(15, new MatrixGraphRepresentation())
DiGraph<double> G2 = new DiGraph<double>(100, new DictionaryGraphRepresentation())
Graph H = new Graph(G.VertexCount, G.Representation)
```

Posługując się powyższymi klasami należy pamiętać o następujących konwencjach:

- Liczba wierzchołków i reprezentacja grafu nie zmienia się w trakcie życia obiektu.
- Wierzchołki grafu numerowane są kolejnymi liczbami całkowitymi, poczynając od 0.

Kilka wskazówek odnośnie operacji na krawędziach:

- Dodawanie i usuwanie krawędzi realizujemy wywołując metody odpowiednio AddEdge i RemoveEdge. Metody zgłaszają wyjątek, gdy numery wierzchołków wychodzą poza zakres oraz zwracają false, gdy operacja nie może zostać wykonana (usuwanie nieistniejącej krawędzi, dodawanie już istniejącej krawędzi).
- Do sprawdzenia istnienia krawędzi i wagi krawędzi służą metody HasEdge oraz GetEdgeWeight. Gdy krawędź nie istnieje w grafie ważonym, próba pobrania jej wagi powoduje zgłoszenie wyjątku.
- Wylistowanie wszystkich wierzchołków, do których prowadzi krawędź z zadanego wierzchołka, można uzyskać posługując się metodą OutNeighbors.
- W grafach ważonych wylistowanie krawędzi wychodzących z zadanego wierzchołka (wraz z ich wagami) otrzymujemy wywołując metodę OutEdges.

Przeszukiwanie grafu

Biblioteka umożliwia przeszukiwanie grafu w głąb i wszcz przez metody odpowiednio DFS i BFS. Metody zwracają obiekt implementujący interfejs IGraphSearcher udostępniający dwie operacje

- SearchFrom – zwraca wyliczenie krawędzi odwiedzanych w trakcie przeszukiwania grafu poczynając od zadanego wierzchołka; w wyliczeniu pojawiają się tylko krawędzie osiągalne z tego wierzchołka.
- SearchAll – zwraca wyliczenie wszystkich krawędzi grafu otrzymane przez serię wywołań SearchFrom z kolejnych jeszcze nieodwiedzonych wierzchołków.

Uwagi:

- Krawędź e jest wyliczana wtedy, kiedy przeszukując graf przechodzimy po niej w kierunku od wierzchołka $e.From$ do $e.To$.
- Grafy nieskierowane są przeszukiwane tak, jakby były grafami skierowanymi zawierającymi krawędzie w obie strony (w szczególności, każda krawędź nieskierowana pojawi się w wyliczeniu dwukrotnie w obu możliwych orientacjach).

Przykład:

```
foreach (Edge e in g.DFS().SearchAll())  
    Console.WriteLine($"Przechodzę przez krawędź {e.From} -> {e.To}");
```

Zadanie: badanie dwudzielnosci i algorytm Kruskala

Uzupełnić w klasie Lab03GraphFunctions następujące metody:

- `DiGraph Lab03Reverse(DiGraph g)` – metoda wyznaczająca odwrotność zadanego grafu skierowanego, gdzie odwrotność grafu to graf skierowany o wszystkich krawędziach przeciwnie skierowanych niż w grafie pierwotnym.
- `bool Lab03IsBipartite(Graph g, out int[] vert)` – metoda sprawdzająca, czy zadany graf jest dwudzielny i, jeśli tak, zwracająca 2-kolorowanie za pośrednictwem parametru wyjściowego `vert`.
- `Graph<int> Lab03Kruskal(Graph<int> g, out int mstw)` – wyznaczanie minimalnego drzewa rozpinającego algorytmem Kruskala.
- `bool Lab03IsUndirectedAcyclic(Graph g)` – sprawdzenie, czy zadany graf jest acykliczny.

Punktacja:

- Etap 1. 0.5 punktu,
- Etap 2. 0.5 punktu,
- Etap 3. 1 punkt,
- Etap 4. 0.5 punkt.

Wskazówki:

- Drugą i czwartą część zadania najłatwiej rozwiązać wykorzystując przeszukiwanie grafu
- W implementacji algorytmu Kruskala przydatne mogą być klasy `UnionFind` oraz `PriorityQueue` z biblioteki `Graphs`

Zadanie: silny indeks chromatyczny

Zadanie składa się z czterech części; części 1-3 są niezależne od siebie, natomiast część 4 korzysta ze wszystkich poprzednich.

Uwagi do wszystkich metod:

1. Grafy wynikowe muszą być reprezentowane w taki sam sposób jak grafy będące parametrami
2. Grafów będących parametrami nie wolno zmieniać

Część I: Funkcja zwracająca kwadrat danego grafu

Kwadratem grafu nazywamy graf o takim samym zbiorze wierzchołków jak graf pierwotny, w którym wierzchołki połączone są krawędzią jeśli w grafie pierwotnym były połączone krawędzią bądź ścieżką złożoną z 2 krawędzi (ale pętli, czyli krawędzi o początku i końcu w tym samym wierzchołku, nie dodajemy!).

Część II: Funkcja zwracająca Graf krawędziowy danego grafu

Wierzchołki grafu krawędziowego odpowiadają krawędziom grafu pierwotnego, wierzchołki grafu krawędziowego połączone są krawędzią jeśli w grafie pierwotnym z krawędzi odpowiadającej pierwszemu z nich można przejść na krawędź odpowiadającą drugiemu z nich przez wspólny wierzchołek.

Tablicę `names` tworzymy i wypełniamy według następującej zasady: każdemu wierzchołkowi grafu krawędziowego odpowiada element tablicy `names` (o indeksie równym numerowi wierzchołka) zawierający informację z jakiej krawędzi grafu pierwotnego wierzchołek ten powstał, np. dla wierzchołka powstałego z krawędzi $\langle 0,1 \rangle$ do tablicy zapisujemy krotkę $(0, 1)$ - przyda się w dalszych etapach.

Część III: Funkcja znajdująca poprawne kolorowanie wierzchołków danego grafu nieskierowanego

Kolorowanie wierzchołków jest poprawne, gdy każde dwa sąsiadujące wierzchołki mają różne kolory.

Funkcja ma szukać kolorowania według następującego algorytmu zachłannego: Dla wszystkich wierzchołków v (od 0 do $n - 1$) pokoloruj wierzchołek v kolorem o najmniejszym możliwym numerze (czyli takim, na który nie są pomalowani jego sąsiedzi).

Kolory numerujemy począwszy od 0.

Funkcja zwraca liczbę użytych kolorów (czyli najwyższy numer użytego koloru $+ 1$), a w tablicy `colors` zapamiętuje kolory poszczególnych wierzchołków.

UWAGA: Podany opis wyznacza kolorowanie jednoznacznie, jakiegokolwiek inne kolorowanie, nawet jeśli spełnia formalnie definicję kolorowania poprawnego, na potrzeby tego zadania będzie uznane za błędne.

Część IV: Funkcja znajduje silne kolorowanie krawędzi danego grafu

Silne kolorowanie krawędzi grafu jest poprawne gdy każde dwie krawędzie, które są ze sobą sąsiednie (czyli można przejść z jednej na drugą przez wspólny wierzchołek) albo są połączone inną krawędzią (czyli można przejść z jednej na drugą przez ową inną krawędź), mają różne kolory.

Należy zwrócić nowy graf, który będzie miał strukturę identyczną jak zadany graf, ale w wagach krawędzi zostaną zapisane przydzielone kolory.

Wskazówka: to bardzo proste. Należy wykorzystać wszystkie poprzednie funkcje. Zastanowić się co możemy powiedzieć o kolorowaniu wierzchołków kwadratu grafu krawędziowego? Jak się to ma do silnego kolorowania krawędzi grafu pierwotnego?