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 wszerz poprzez 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 dwudzielności 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 pośrednicwem 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 blibioteki Graphs