Лабораторна робота № 2

Теоретичні відомості. Каркаси

Каркасом, або *з'єднувальним деревом* графа називають його підграф, який являє собою дерево та містить усі вершини цього графа. Нехай граф $G = (V, E)$ має n вершин і m ребер. Щоб побудувати каркас, можна вилучати ребра, які утворюють прості цикли. Потрібно вилучити $\gamma = m - n + 1$ ребер (оскільки дерево має $n - 1$ ребро [1]). Число γ називається *цикломатичним числом* графа і воно є числовою характеристикою зв'язності графа. Очевидно, що цикломатичне число дерева рівне нулеві.

На практиці каркаси будують не вилученням ребер із простих циклів (це неефективно для комп'ютерної реалізації), а, навпаки, добиранням ребер у каркас. При цьому, потрібно відслідковувати, щоб кожне додане ребро не утворювало циклу у дереві.

Одним із способів побудови каркасу графа є його обхід методом пошуку вглиб та вшир (див. лабораторну роботу „Обхід графів" з курсу „Дискретна математика"). Можна показати, що каркас, побудований обходом графу пошуком ушир з певної вершини графа утворює одночасно найкоротші шляхи з цієї вершини до інших вершин графа (аналог алгоритму Дейкстри (див. лабораторну роботу „Пошук найкоротших шляхів у графі" з курсу „Дискретна математика").

Окремо слід розглянути оптимізаційну задачу побудови каркасу, в якому сума ваг ребер є мінімальною (для зважених графів). Такий каркас називається *мінімальним*. Найбільш відомими алгоритмами розв'язання цієї задачі є *алгоритми Краскала* (J.Kruskal) та *Прима* (R.C.Prim).

Зауваження. Алгоритми Краскала та Прима належать до сімейства „*жадібних*" алгоритмів. Їхня часова складність складає $O(m \log m)$ [3, 7, 8].

Алгоритм Краскала

Розглянемо реалізацію алгоритму Краскала з використанням „*системи множин, що не перетинаються*” (англ. *disjoint set union* (DSU)) [3, 9].

Система DSU характеризується такими особливостями. Нехай ми маємо набір різних елементів. Спочатку кожен елемент заноситься в окрему множину, що складається з нього самого. Додавання нового елемента до системи DSU означає створення нової множини та занесення в неї цього елемента. Окрім того, для системи DSU вводяться операції визначення приналежності елемента до певної множини (операція пошуку) та об’єднання двох різних множин.

Розглянемо найпростіші реалізації згаданих вище операцій. По суті, кожна множина представляється у вигляді кореневого дерева. Для роботи з цим деревом потрібно для кожного елемента `v` ввести змінну `previous[v]`, в якій буде зберігатися посилання на батьківський вузол (елемент) в цьому дереві. При цьому кореневий елемент повинен посилатися на самого себе.

Створення (додавання) нового елемента. Це найпростіша операція. Оскільки, за означенням, кожен елемент знаходиться в окремій множині (дереві) то він і є коренем і повинен вказувати на самого себе. Тобто змінна `previous[v]` ініціалізується самим значенням `v`.

Визначення приналежності елемента до певної множини (дерева). Ця операція зводиться до рекурсивного пошуку кореня дерева, починаючи від заданого елемента. Якщо заданий елемент вже є коренем (тобто `previous[v]==v`), то функція повинна повернути значення цього елемента `v` (або `previous[v]`, оскільки вони співпадають). В іншому випадку повертається результат рекурсивного виклику цієї ж функції пошуку для значення `previous[v]`.

Об’єднання двох множин (дерев). Аргументами цієї функції повинні бути два елементи (вузли), які належать множинам (деревам), які потрібно

об’єднати. Спочатку для кожного елемента викликаємо описану вище функцію пошуку кореня. Якщо значення коренів співпадають, то обоє елементів знаходяться в одній множині (дереві) і далі нічого не потрібно робити. В іншому випадку потрібно вказати, що предок кореня одного є коренем іншого дерева (або навпаки), тим самим об’єднавши множини (дерева). Важливо: об’єднувати дерева потрібно саме через корені, а не через некореневі вузли – в останньому випадку отримується дерево з двома коренями.

Зауваження. Описані вище процедури створення, пошуку та об’єднання для системи DSU разом складають алгоритм об’єднання-пошуку (англ. *union-find algorithm*). Ці процедури є неоптимальними і можуть бути вдосконалені [9]. Систему DSU також називають (англ.) *union–find data structure*.

Алгоритм Краскала (реалізація).

1. Відсортувати ребра графа за зростанням їхніх ваг.
2. Занести кожну вершину в окрему множину (операція створення/додавання).
3. Для кожного ребра (в порядку зростання ваг): якщо вершини, що утворюють кінці цього ребра належать різним множинам (це забезпечує відсутність простих циклів), то долучити це ребро у каркас і об’єднати множини, що містять кінці ребра. В іншому випадку ребро пропускається.

Зауваження. Після розгляду всіх m ребер (з яких $n-1$ ребро повинне бути долученим у каркас) всі множини елементів повинні бути об’єднані в одну множину.

Приклад.

Розглянемо побудову за алгоритмом Краскала мінімального каркасу графа, зображеного на рис. 2.1. Результат побудови наведено на рис. 2.2. Мінімальний каркас зображений на цьому рисунку потовщеними лініями, а ребра, які не входять в каркас – пунктирними. Протокол побудови наведений у таблиці 2.1.

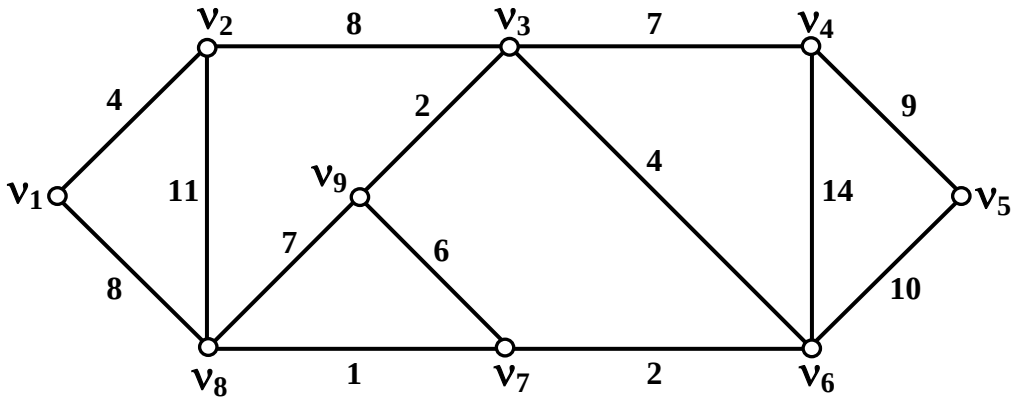

Рис. 2.1

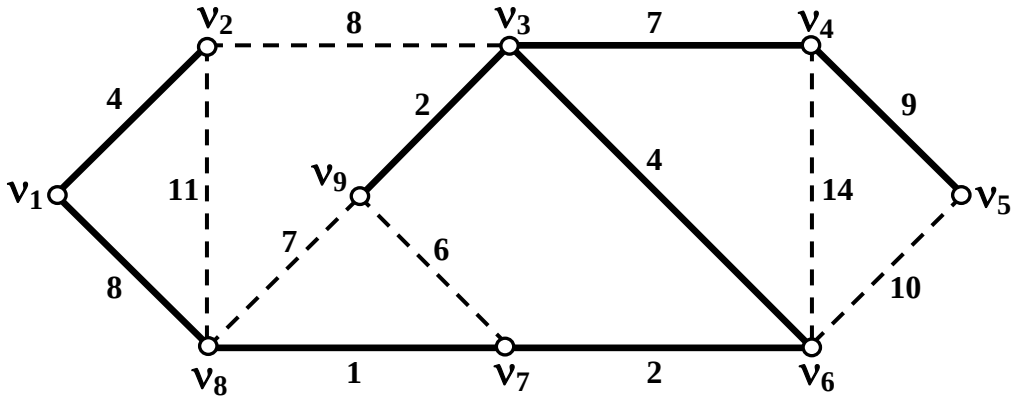

Рис. 2.2

Таблиця 2.1

Ребро	**Вага**	**Розбиття множини вершин**	**Добір ребра в каркас**
		$\{\{v_1\},\{v_2\},\{v_3\},\{v_4\},\{v_5\},\{v_6\},\{v_7\},\{v_8\},\{v_9\},\}$	
$\{v_7,v_8\}$	1	$\{\{v_1\},\{v_2\},\{v_3\},\{v_4\},\{v_5\},\{v_6\},\{v_7,v_8\},\{v_9\},\}$	так

$\{v_3, v_9\}$	2	$\{\{v_1\},\{v_2\},\{v_3, v_9\},\{v_4\},\{v_5\},\{v_6\},\{v_7, v_8\}\}$	так
$\{v_6, v_7\}$	2	$\{\{v_1\},\{v_2\},\{v_3, v_9\},\{v_4\},\{v_5\},\{v_6, v_7, v_8\}\}$	так
$\{v_1, v_2\}$	4	$\{\{v_1, v_2\},\{v_3, v_9\},\{v_4\},\{v_5\},\{v_6, v_7, v_8\}\}$	так
$\{v_3, v_6\}$	4	$\{\{v_1, v_2\},\{v_4\},\{v_5\},\{v_3, v_6, v_7, v_8, v_9\}\}$	так
$\{v_7, v_9\}$	6	$\{\{v_1, v_2\},\{v_4\},\{v_5\},\{v_3, v_6, v_7, v_8, v_9\}\}$	ні
$\{v_3, v_4\}$	7	$\{\{v_1, v_2\},\{v_5\},\{v_3, v_4, v_6, v_7, v_8, v_9\}\}$	так
$\{v_8, v_9\}$	7	$\{\{v_1, v_2\},\{v_5\},\{v_3, v_4, v_6, v_7, v_8, v_9\}\}$	ні
$\{v_1, v_8\}$	8	$\{\{v_5\},\{v_1, v_2, v_3, v_4, v_6, v_7, v_8, v_9\}\}$	так
$\{v_2, v_3\}$	8	$\{\{v_5\},\{v_1, v_2, v_3, v_4, v_6, v_7, v_8, v_9\}\}$	ні
$\{v_4, v_5\}$	9	$\{\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}\}$	так
$\{v_5, v_6\}$	10	$\{\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}\}$	ні
$\{v_2, v_8\}$	11	$\{\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}\}$	ні
$\{v_4, v_6\}$	14	$\{\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}\}$	ні

Хід роботи:

Частина 1. Побудова каркасу мінімальної ваги за алгоритмом Краскала

1. Створити нову бібліотеку `DSU` (файли `DSU.h`, `DSU.cpp`) в якій реалізувати основні функції для роботи з системою DSU: `set_make(v)`, `set_find(v)` та `set_union(v1, v2)`, для операцій створення, пошуку та об'єднання, відповідно.
2. Створити новий проект `Lab_2_1`, до якого підключити бібліотеку `DSU`.

3. В цьому проекті описати структуру для задавання графа списком ребер. Структура повинна містити поля, що задають вершини ребра та його вагу. Задати масив таких структур для опису графа.
4. Реалізувати функцію `sort(…)` для сортування списку ребер графа за зростанням ваг (алгоритм сортування задає викладач) та додати її у проект.
5. Згідно описаної вище методики запрограмувати реалізацію алгоритму Краскала.
6. Отримати від викладача варіант завдання (графу) і виконати для нього побудову мінімального каркасу. Продемонструвати результат викладачеві.

Алгоритм Прима

В алгоритмі Прима будується множина U позначених (відвіданих) вершин, які поступово „наповнюють” мінімальний каркас. Нехай V – множина усіх вершин графа. Спочатку у множину U додаємо одну вершину (будь-яку). На кожному кроці алгоритму шукається ребро найменшої ваги (u,v) таке, що $u \in U$, а $v \in V \setminus U$, після чого вершина v заноситься у множину U. Цей процес продовжується доти, доки $U = V$.

Для практичної реалізації вищеописаної процедури введемо два масиви `sel_e` та `min_e`. Перший масив `sel_e` для кожної вершини графа з множини $V \setminus U$ містить номер вершини у множині U, з якою вона з’єднана ребром мінімальної ваги, а другий масив `min_e` – вагу цього мінімального ребра. Окрім того, введемо логічний масив `visited`, для визначення якій з множин, U чи $V \setminus U$ належить вершина.

На кожній ітерації, після внесення в множину U нової вершини, для кожної вершини з множини $V \setminus U$ масив `sel_e` повинен бути оновлений номером вершини з U, що з’єднана з нею ребром мінімальної ваги, а масив `min_e` – значенням ваги цього ребра. Після цього, серед всіх значень `min_e`

знаходять мінімальне і тоді це ребро стає претендентом для занесення у каркас.

Зауваження. На першій ітерації тільки одна вершина є занесена у каркас, тому значення `sel_e` ініціалізується цією вершиною для решти вершин. Масив `min_e` ініціалізується відповідними ребрами. На кожній з подальших ітерацій у каркас заноситься тільки одна вершина, тому значення `sel_e` стає рівним або номеру цієї вершини, або залишається попереднім (відповідно у масив `min_e` також або вносяться, або не вносяться зміни).

Приклад.

Розглянемо побудову за алгоритмом Прима мінімального каркасу графа, зображеного на рис. 2.1. Початковою вершиною обрано вершину ν_1. Результат побудови наведено на рис. 2.3. Мінімальний каркас зображений на цьому рисунку потовщеними лініями, а ребра, які не входять в каркас – пунктирними. Протокол побудови наведений у таблиці 2.2. Зауважимо, що для даного графа мінімальні каркаси, побудовані різними методами, відрізняються включенням чи не включенням в каркас ребер $\nu_2\nu_3$ та $\nu_1\nu_8$ з вагою 8.

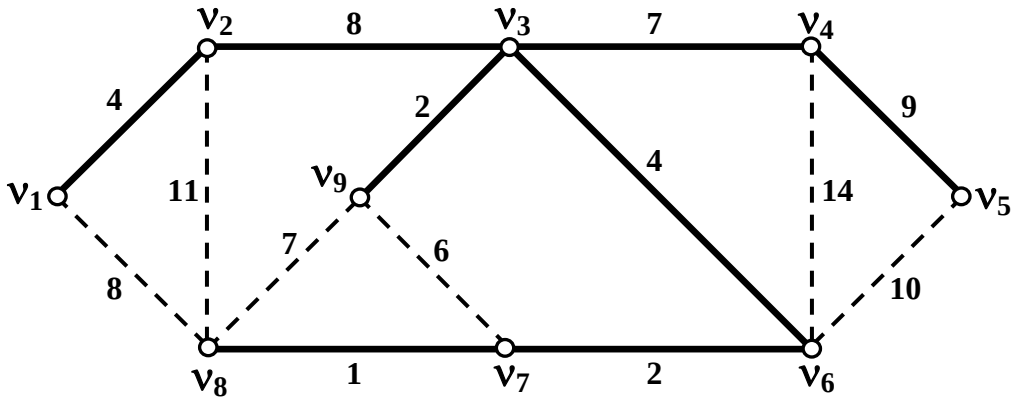

Рис. 2.3

Таблиця 2.2

Множина *U*	**Добір ребра в каркас**	**Вага**
$\{v_1\}$	–	–
$\{v_1, v_2\}$	$\{v_1, v_2\}$	4
$\{v_1, v_2, v_3\}$	$\{v_2, v_3\}$	8
$\{v_1, v_2, v_3, v_9\}$	$\{v_3, v_9\}$	2
$\{v_1, v_2, v_3, v_6, v_9\}$	$\{v_3, v_6\}$	4
$\{v_1, v_2, v_3, v_6, v_7, v_9\}$	$\{v_6, v_7\}$	2
$\{v_1, v_2, v_3, v_6, v_7, v_8, v_9\}$	$\{v_7, v_8\}$	1
$\{v_1, v_2, v_3, v_4, v_6, v_7, v_8, v_9\}$	$\{v_3, v_4\}$	7
$\{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\}$	$\{v_4, v_5\}$	9

Хід роботи:

Частина 2. Побудова каркасу мінімальної ваги за алгоритмом Прима

1. Створити новий проект `Lab_2_2`, в якому реалізувати описаний вище алгоритм Прима:
   1.1. Створити та ініціалізувати масиви `min_e`, `sel_e` та `visited`.
   1.2. Для кожної незанесеної в каркас вершини реалізувати пошук вершини, занесеної в каркас, з якою вона з'єднана ребром мінімальної ваги: оновлення масивів `sel_e` та `min_e`. Серед оновлених значень `min_e` знайти мінімальне значення і відповідне ребро (та вершину) занести у каркас. Продовжувати описану процедуру поки всі вершини графа не будуть занесені у каркас.
2. Отримати від викладача варіант завдання (графу) та початкової вершини і виконати для нього побудову мінімального каркасу методом

Прима. Продемонструвати результат викладачеві.

Перелік посилань

1. Нікольский Ю. В. Дискретна математика / Ю. В. Нікольский, В. В. Пасічник, Ю. М. Щербина. – Львів: „Магнолія–2006”, 2013. – 432 с.
2. Андерсон Д. Дискретная математика и комбинаторика / Д Андерсон. – М.: Изд. дом „Вильямс”, 2004. – 960 с.
3. Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзер, Р. Ривест, К. Штайн. – М.: Изд. дом „Вильямс”, 2011. – 1296 с.
4. Вирт Н. Алгоритмы и структуры данных / Н. Вирт. – М.: Мир, 1989. – 360 с.
5. Ковалюк Т. В. Алгоритмізація та програмування / Т. В. Ковалюк. – Львів: „Магнолія–2006”, 2013. – 400 с.
6. Павловская Т. А. C/C++. Программирование на языке высокого уровня / Т.А. Павловская. – СПб.: Изд-во „Питер”, 2013. – 461 с.
7. Ахо А. Структуры данных и алгоритмы / А. Ахо, Дж. Хопкрофт, Дж. Ульман. – М.: Изд. дом „Вильямс”, 2010. – 400 с.
8. Липский В. Комбинаторика для программистов / В. Липский. – М.: Мир, 1988. – 216 с.
9. Седжвик Р. Алгоритмы на C++ / Р. Седжвик. – М.: Изд. дом „Вильямс”, 2014. – 1056 с.