БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СБОРНИК ЗАДАЧ

ПО ТЕОРИИ АЛГОРИТМОВ.

Структуры данных.
Часть. Специальные структуры данных.
Система непересекающихся множеств.

МИНСК
2020

УДК 510.51(075.8)
ББК 22.12я73-1
С23

Авторы:
**С. А. Соболь, К. Ю. Вильчевский, В. М. Котов,
Е. П. Соболевская**

Рецензенты:
кафедра информатики и методики преподавания информатики
физико-математического факультета Белорусского государственного
педагогического университета им. М. Танка
(заведующий кафедрой, кандидат педагогических наук,
доцент *С. В. Вабищевич*);
профессор кафедры информационных технологий в культуре
Белорусского государственного университета культуры и искусства,
кандидат физико-математических наук, доцент *П. В. Гляков*

ISBN 978-985-518-412-4

Часть 1

СПЕЦИАЛЬНЫЕ СТРУКТУРЫ ДАННЫХ

1.1. СИСТЕМА НЕПЕРЕСЕКАЮЩИХСЯ МНОЖЕСТВ

В некоторых задачах необходимо хранить разбиение какого-то набора уникальных объектов на непересекающиеся *динамические множества* (англ. *dynamic sets*). В каждом множестве выделен один из его элементов, который называют *представителем* (англ. *representative*). Представитель множества определяет данное множество, иногда в литературе вместо слова представитель встречается слово *лидер*.

Ниже приведён пример системы непересекающихся множеств (для каждого множества представитель выделен полужирным шрифтом):

$$\{1, \mathbf{2}, 3, 4, 8\}, \{5, \mathbf{6}\}, \{\mathbf{7}\}. \tag{1.1}$$

Пусть изначально каждый объект находится в собственном одноэлементном множестве. С множествами нужно уметь выполнять следующие операции:

1) $\textsc{FindSet}(x)$ — выдать указатель на представителя множества, которому принадлежит элемент x;

2) $\textsc{Union}(x, y)$ — объединить два непересекающихся множества, которые содержат элементы x и y.

Структуру данных, поддерживающую такой интерфейс, называют *системой непересекающихся множеств* (СНМ) (англ. *disjoint set union* (DSU)).

Предположим, что набор объектов, для которого выполнялось разбиение на непересекающиеся множества, — целые числа от 1 до n. Рассмотрим несколько способов реализации описанного выше интерфейса.

1.1.1. Реализация на массиве

Реализация с использованием структуры данных *массив* предполагает, что элемент массива с индексом i (нумерация с 1) содержит представителя множества, которому принадлежит элемент i.

Для системы непересекающихся множеств (1.1) массив, который используется для работы с множествами, будет иметь следующий вид:

1	2	3	4	5	6	7	8
2	2	2	2	6	6	7	2

При такой реализации время выполнения базовых операций следующее:

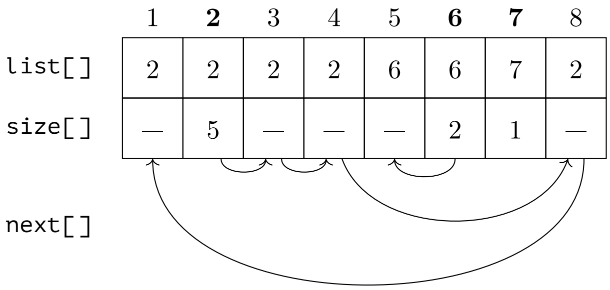
• FindSet(x) выполняется за время O(1) — достаточно одного обращения к элементу массива по индексу;

• Union(x, y) выполняется за время O(n) — необходим проход по всему массиву.

1.1.2. Реализация на массиве и списке

Реализация с использованием структуры данных *список с указателем на представителя* предполагает, что элементы каждого множества связаны в отдельный список. Представителем множества является первый элемент списка. Каждый элемент списка содержит указатель на представителя и на следующий за ним элемент множества. Для каждого списка поддерживают два указателя: на первый и последний элементы списка. Кроме того, для каждого элемента множества необходимо в массиве дополнительно поддерживать указатель на его место в списковой структуре.

Вместо того, чтобы организовывать связные списки на указателях, можно использовать для хранения системы непересекающихся множеств несколько массивов. Система множеств 1.1 может быть представлена следующим образом в виде трёх массивов:



Здесь массив `list` для каждого i содержит представителя множества, к которому принадлежит i. Массив `size` для каждого представителя хранит размер его множества, остальные элементы этого массива не используются (размеры множеств будут учитываться далее в ходе операции объединения). Массив `next` (на рисунке изображён условно в виде стрелок) связывает элементы каждого множества в цепочку, первым элементом которой является представитель.

Такая сложная структура даёт возможность эффективно перечислить элементы любого множества (за время, пропорциональное размеру этого множества). Однако в худшем случае время выполнения базовых операций остаётся прежним:

- FindSet(x) выполняется за время O(1);
- Union(x, y) выполняется за время O(n).

Размер множества выступает его весовой характеристикой. Оказывается, такие веса можно использовать, чтобы сделать выполнение операции объединения более эффективными.

Введём правило «*меньшее к большему*»: при выполнении операции Union(x, y) ссылки на нового представителя изменяются у всех элементов меньшего множества. Данное правило позволяет получить следующую оценку [1]:

Теорема 1.1. *Последовательность из m операций* FindSet(x), Union(x, y) *потребует времени* $O(m + n \log n)$.

Доказательство. Каждый раз, когда какой-то элемент перемещается из одного множества в другое с изменением представителя, размер множества, содержащего этот элемент, увеличивается не менее чем вдвое. Поскольку множество может вырасти лишь до размера n, каждый элемент может подвергнуться перемещению не более чем $\log_2 n$ раз. □

1.1.3. Реализация с помощью корневых деревьев

Другой подход предполагает использование *семейства корневых деревьев*. Каждому множеству поставим в соответствие своё корневое дерево. Каждой вершине дерева соответствует ровно один элемент множества. Корень дерева содержит представителя множества.

Семейство корневых деревьев в памяти компьютера хранится в каноническом виде — как массив `parent`, где `parent[i]` — отец вершины i в корневом дереве. Если вершина i является корнем дерева, то верно равенство `parent[i] == i`, т. е. представитель множества ссылается на самого себя.

На рис. 1.1 система непересекающихся множеств (1.1) представлена в виде семейства корневых деревьев.

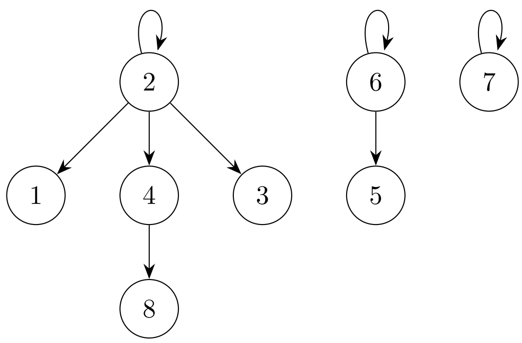


Рис. 1.1. Представление системы непересекающихся множеств в виде семейства корневых деревьев

В памяти компьютера семейство корневых деревьев хранится в следующем виде:

	1	2	3	4	5	6	7	8
`parent[]`	2	2	2	2	6	6	7	4

При простой реализации последовательность из $n-1$ операции Union может привести к тому, что будет построено корневое дерево высоты $n-1$, а время выполнения базовых операций будет следующим:

- FindSet(x) выполняется за время $\mathrm{O}(n)$;
- Union(x, y) выполняется за время $\mathrm{O}(n)$.

Эвристика объединения по размеру или рангу

Время выполнения операций можно улучшить, если у каждого корневого дерева поддерживать весовую характеристику, в качестве которой может выступать или число вершин в дереве (размер), или высота дерева (ранг). Тогда при выполнении операции Union корень дерева с меньшей весовой характеристикой будет ссылаться на корень дерева с большей весовой характеристикой. Можно доказать, что время выполнения базовых операций будет следующим:

- FindSet(x) выполняется за время $O(\log n)$;
- Union(x, y) выполняется за время $O(\log n)$.

Описанное правило (эвристика) называется *объединение по размеру или рангу* (англ. *union by size or rank*).

Рассмотрим случай, когда в качестве весовой характеристики используется размер дерева. В памяти компьютера семейство корневых деревьев с дополнительной весовой характеристикой `size` хранится следующим образом:

	1	2	3	4	5	6	7	8
`size[]`	1	5	1	2	1	2	1	1
`parent[]`	2	2	2	2	6	6	7	4

Иногда не вводят дополнительное поле (`size`), а в массиве `parent` для представителя множества хранят со знаком минус весовую характеристику множества, т. е. знак минус идентифицирует представителя множества [3].

	1	2	3	4	5	6	7	8
`parent[]`	2	–5	2	2	6	–2	–1	4

На рис. 1.2 приведён результат выполнения операции Union(5, 8) для системы непересекающихся множеств, приведённой на рис. 1.1.

В памяти компьютера после выполнения операции Union(5, 8) система непересекающихся множеств будет иметь следующий вид:

	1	2	3	4	5	6	7	8
`size[]`	1	7	1	2	1	2	1	1
`parent[]`	2	2	2	2	6	2	7	4

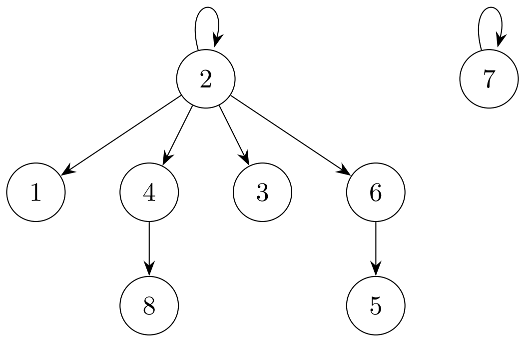


Рис. 1.2. Результат выполнения Union$(5, 8)$ с эвристикой объединения по размеру

Эвристика сжатия пути

Ещё одно усовершенствование, которое позволяет получить практически линейное время работы серии операций FindSet и Union, носит название *сжатие пути* (англ. *path compression*). Предположим, что выполняется операция FindSet(x), которая выполняется простым подъёмом от вершины x к корню дерева, используя массив предков `parent`. Все посещённые при этом подъёме вершины составляют *путь поиска.* Процедура сжатия пути всем вершинам, лежащим на пути поиска, в качестве предка присваивает ссылку на корень данного дерева (сжатие пути не изменяет ранги вершин). Теперь массив `parent` правильнее называть *сжатым массивом предков.*

На рис. 1.3 приведён результат выполнения операции FindSet(8) со сжатием пути для системы непересекающихся множеств, приведённой на рис. 1.2.

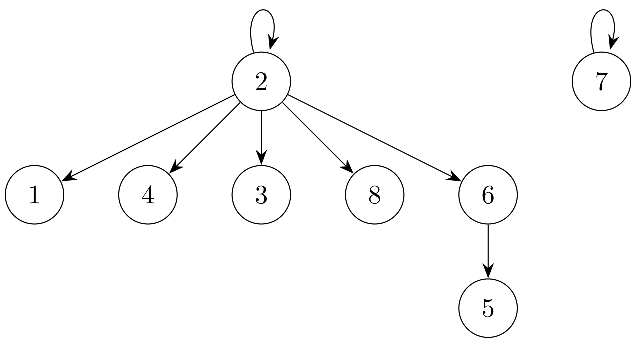


Рис. 1.3. Результат выполнения FindSet(8) с эвристикой сжатия пути

Справедлива следующая теорема (доказательство теоремы можно найти в [1]).

Теорема 1.2. *Последовательность из m операций* FindSet(x) *и*

Union(x, y) *может быть выполнена с использованием эвристик объединения по размеру и сжатия пути за время* $\mathrm{O}(m\alpha(n))$ *в худшем случае.*

В формулировке теоремы функция $\alpha(n)$ — обратная функция для функции Аккермана. Функция $\alpha(n)$ растёт очень медленно, она становится больше числа 4 только для очень больших значений $n \gg 10^{80}$. Поэтому для практических приложений полагают, что $\alpha(n) \leqslant 4$.

Функция Аккермана:

$$A(m, n) = \begin{cases} n + 1, & m = 0; \\ A(m-1, 1), & m > 0, n = 0; \\ A(m-1, A(m, n-1)), & m > 0, n > 0. \end{cases}$$

Обратная функция для функции Аккермана определяется следующим образом:

$$\alpha(n) = \min\{k \geqslant 1 : A(k, k) \geqslant n\}.$$

1.1.4. Псевдокод

Приведём итоговую реализацию системы непересекающихся множеств на псевдокоде, которая хранит множества в виде семейства корневых деревьев и применяет обе описанные эвристики (объединение по размеру и сжатие пути).

```
class DisjointSetUnion:
    def __init__(self, n):
        self.size = [1 for i in range(n)]
        self.parent = [i for i in range(n)]

    def FindSet(self, x):
        if x == self.parent[x]:
            return x
        # Path compression
        self.parent[x] = self.FindSet(self.parent[x])
        return self.parent[x]

    def Union(self, x, y):
        x = self.FindSet(x)
        y = self.FindSet(y)
        if x != y:
            # Union by size
            if self.size[x] < self.size[y]:
                # Swap x and y
                x, y = y, x
```

```
            # Now size[x] >= size[y]
            self.parent[y] = x
            self.size[x] += self.size[y]
```

Заметим, что в результате применения сжатия пути размеры поддеревьев, хранящиеся в массиве `size`, для некоторых вершин становятся неактуальными. Однако это не влияет на корректность алгоритма: при выполнении объединения играют роль только значения размера, которые хранятся в корнях деревьев — у представителей множеств.

Поскольку система непересекающихся множеств является узкоспециализированной структурой данных и используется в прикладном программировании крайне редко, она не включена в стандартные библиотеки популярных языков программирования. Тем не менее реализация DSU есть в известной библиотеке boost для языка C++.

БИБЛИОГРАФИЧЕСКИЕ ССЫЛКИ

1. Алгоритмы: построение и анализ / Т. Кормен [и др.]. — М. : Вильямс, 2005. — 1296 с.
2. *Котов В. М.*, *Мельников О. И.* Информатика. Методы алгоритмизации : учеб. пособие для 10 – 11 кл. общеобразоват. шк. с углубл. изучением информатики. — Минск : Нар. асвета, 2000. — 221 с.
3. *Котов В. М.*, *Соболевская Е. П.*, *Толстиков А. А.* Алгоритмы и структуры данных : учеб. пособие. — Минск : БГУ, 2011. — 267 с. — (Классическое университетское издание).
4. Сборник задач по теории алгоритмов : учеб.-метод. пособие/ В. М. Котов [и др.]. — Минск : БГУ, 2017. — 183 с.
5. Теория алгоритмов : учеб. пособие / П. А. Иржавский [и др.]. — Минск : БГУ, 2013. — 159 с.
6. *Соболь С. А.*, *Котов В. М.*, *Соболевская Е. П.* Опыт использования образовательной платформы Insight Runner на факультете прикладной математики и информатики Белорусского государственного университета // Роль университетского образования и науки в современном обществе : материалы междунар. науч. конф., Минск, 26–27 февр. 2019 г. / Белорус. гос. ун-т ; редкол.: А. Д. Король (пред.) [и др.]. — Минск : БГУ, 2019. — С. 263-267.
7. *Соболь С. А.*, *Котов В. М.*, *Соболевская Е. П.* Методика преподавания дисциплин по теории алгоритмов с использованием образовательной платформы iRunner // Судьбы классического университета: национальный контекст и мировые тренды [Электронный ресурс] : материалы XIII Респ. междисциплинар. науч.-теорет. семинара «Инновационные стратегии в современной социальной философии» и междисциплинар. летней школы молодых ученых «Экология культуры», Минск, 9 апр. 2019 г. / Белорус. гос. ун-т ; сост.: В. В. Анохина, В. С. Сайганова ; редкол.: А. И. Зеленков (отв. ред.) [и др.] — С. 346–355.

СОДЕРЖАНИЕ

Учебное издание

Соболь Сергей Александрович
Вильчевский Константин Юрьевич
Котов Владимир Михайлович и др.

СБОРНИК ЗАДАЧ ПО ТЕОРИИ АЛГОРИТМОВ. СТРУКТУРЫ ДАННЫХ

Учебно-методическое пособие

Редактор *Х. Х. ХХХХХХХ*
Художник обложки *С. А. Соболь*
Технический редактор *Х. Х. ХХХХХХХ*
Компьютерная вёрстка *С. А. Соболя*
Корректор *Х. Х. ХХХХХХ*

Подписано в печать 29.02.2020. Формат 60×84/16. Бумага офсетная.
Печать офсетная. Усл. печ. л. 10,69. Уч.-изд. л. 9,6.
Тираж 150 экз. Заказ

Белорусский государственный университет.
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/270 от 03.04.2014.
Пр. Независимости, 4, 220030, Минск.

Республиканское унитарное предприятие
«Издательский центр Белорусского государственного университета».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 2/63 от 19.03.2014.
Ул. Красноармейская, 6, 220030, Минск.