Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: Объектно-ориентированное программирование

Тема: Структура данных с непересекающимся набором (Union-Find)

Студентка гр. 3331506/90401 Чеботарева В. Ю.

Преподаватель Ананьевский М. С.

Санкт-Петербург
2022 г.

Оглавление

Введение

Система непересекающихся множест**в** (англ. *disjoint-set*, или *union–find data structure*) — структура данных, которая позволяет администрировать множество элементов, разбитое на непересекающиеся подмножества. При этом каждому подмножеству назначается его представитель — элемент этого подмножества.

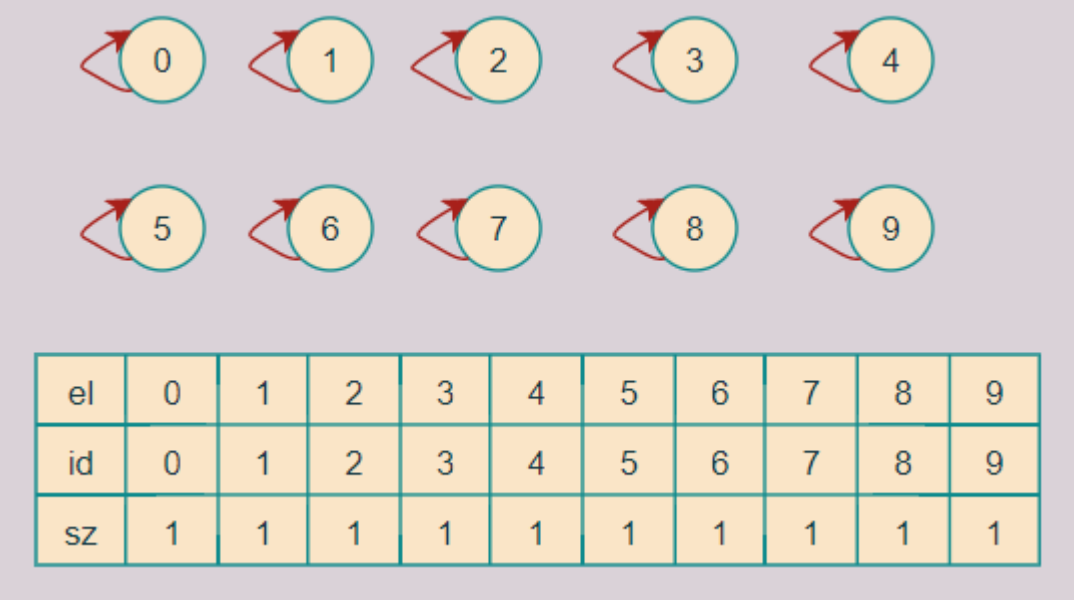
Не имеет значения, что из себя представляют объекты. Для простоты можно сопоставить N объектам числа от 0 до $N - 1$.

Не нужно определять, каким образом объекты связаны между собой, достаточно будет определить, принадлежат ли сопоставляемые им элементы одному множеству.

Когда элементы непересекающихся множеств образуют связь, их множества объединяются.

Описание алгоритма

Допустим, у нас есть 10 элементов (рисунок 1), которые изначально являются множествами, замкнутыми на себя (т.е. они - корни).

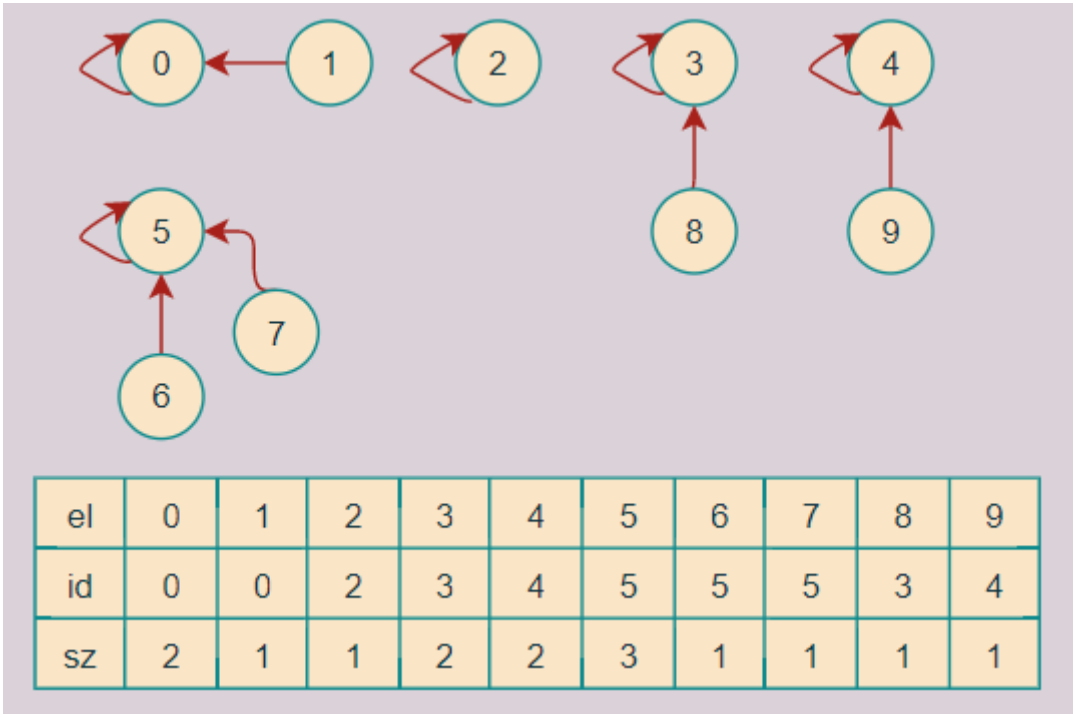


el	0	1	2	3	4	5	6	7	8	9
id	0	1	2	3	4	5	6	7	8	9
sz	1	1	1	1	1	1	1	1	1	1

Рисунок 1 – 10 одноэлементных множеств

Когда мы хотим объединить элементы в множество, сначала ищется корень множества каждого элемента, после чего сравнивается размер множеств, и корнем корня меньшего множества становится корень большего. При таком подходе высота самого высокого дерева будет не более $\log(N)$.

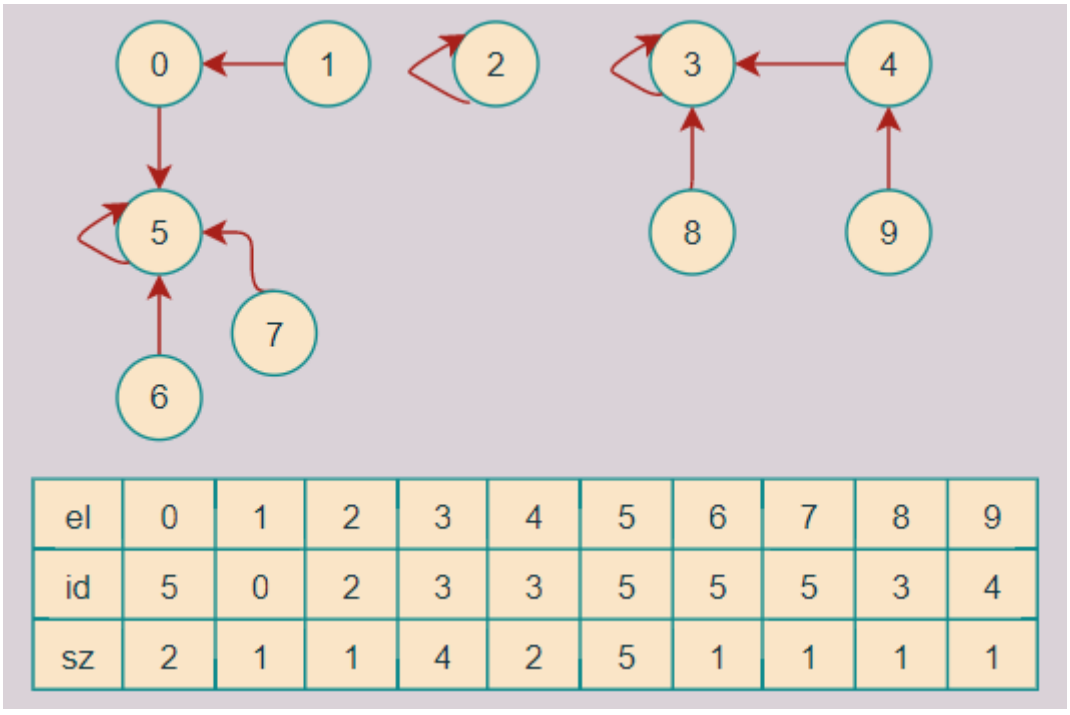
На рисунке 2 представлен результат объединения элементов 0 и 1, 8 и 3, 4 и 9, 6 и 5, 7 и 5. В таблице видно, как поменялось id элементов при объединении.



el	0	1	2	3	4	5	6	7	8	9
id	0	0	2	3	4	5	5	5	3	4
sz	2	1	1	2	2	3	1	1	1	1

Рисунок 2 – Результат объединения элементов

На рисунке 3 показан результат объединения множеств с корнями 0 и 5, 3 и 4.



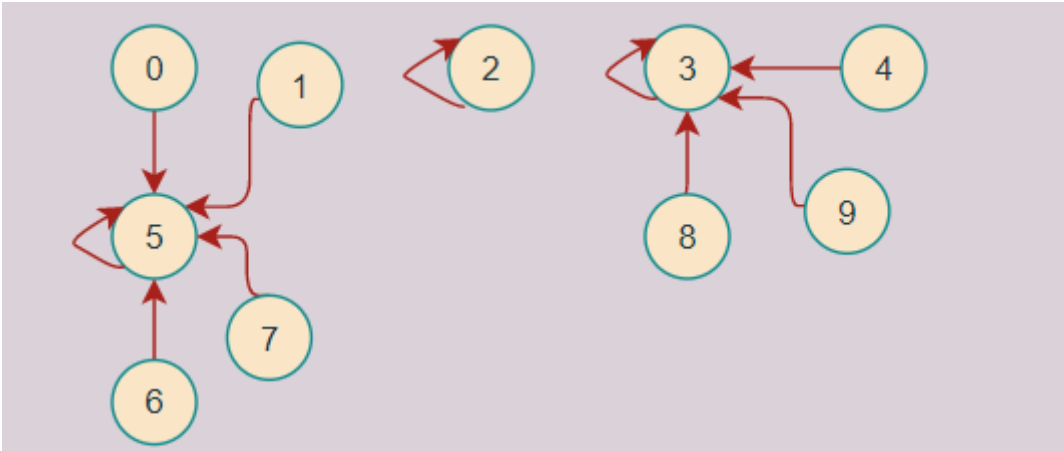
el	0	1	2	3	4	5	6	7	8	9
id	5	0	2	3	3	5	5	5	3	4
sz	2	1	1	4	2	5	1	1	1	1

Рисунок 3 – Результат объединения множеств

При этом в функции find, которая находит корни множеств, происходит сжатие путей, что позволяет ускорить алгоритм при последующем

обращении к элементам множества. «Сжатие путей» означает присоединение элемента непосредственно к корню, без посредника.

На рисунке 4 видно, что элемент 1 теперь “прикрепляется” непосредственно к корню 5, а 9 – к корню 4.



el	0	1	2	3	4	5	6	7	8	9
id	5	5	2	3	3	5	5	5	3	3
sz	2	1	1	4	2	5	1	1	1	1

Рисунок 4 – Пример сжатия путей

Оценка сложности

Сложность операции *merge* после применения сжатия путей ограничивается функцией итерированного логарифма $O(\log^*(N))$. Эта функция возрастает неограниченно, но очень медленно. Это означает, что для объема данных, встречающегося на практике, операция выполняется за константное время.

n	$\log_2^* n$
$(-\infty, 1]$	0
(1, 2]	1
(2, 4]	2
(4, 16]	3
(16, 65536]	4
$(65536, 2^{65536}\ (\sim 10^{19660})]$	5

Вывод

В ходе выполнения курсового проекта был рассмотрен и реализован алгоритм реализации структуры данных с непересекающимися множествами.

Алгоритм имеет временную сложность, описываемую функцией итерированного логарифма, что означает при объемах данных, встречающихся на практике алгоритм выполняется за константное время.

Список литературы

1. Union-Find / Kevin Wayne, Pearson-Addison Wesley
2. Disjoint-set data structure (2019) in Wikipedia: The Free Encyclopedia, Wikimedia Foundation Inc.

Приложение 1

```
#ifndef UNTITLEDDDDDDDDDDDD_LIBRARY_H
#define UNTITLEDDDDDDDDDDDD_LIBRARY_H
using namespace std;

class UF {
private:
    int *id;
    int amount = 0;
    int *size;
public:
    UF(int N);

    ~UF();

    int find(int p);

    void merge(int x, int y);

    bool connected(int x, int y);

    int count();
};

#endif //UNTITLEDDDDDDDDDDDD_LIBRARY_H
```

Приложение 2

```
#include "library.h"
//создание пустой структуры данных из N изолированных множеств
UF::UF(int N) {
    amount = N;
    id = new int[N];
    size = new int[N];
    for (int i = 0; i < N; i++) {
        id[i] = i;
        size[i] = 1;
    }
}
//удаление
UF::~UF() {
    delete[] id;
    delete[] size;
}

// Возвращение корня объекта p
int UF::find(int p) {
    int root = p;
    while (root != id[root])
        root = id[root];
    // "path compression"
    //делаем так, чтобы каждый элемент напрямую присоединялся к корню
    while (p != root) {
        int next = id[p];
        id[p] = root;
        p = next;
    }
    return root;
}

// Объединение множеств
void UF::merge(int x, int y) {
    //находим корни множеств
    int i = find(x);
    int j = find(y);

    //если x и y и так в одном множестве ничего не надо делать
    if (i == j) return;

    // присоединяем меньшее множество к большему
    //для корня меньшего, корнем становится корень большего
    if (size[i] < size[j]) {
        id[i] = j;
        size[j] += size[i];
    } else {
        id[j] = i;
        size[i] += size[j];
    }
    amount --;
}

// Проверка находятся ли объекты x и y в одном множестве
bool UF::connected(int x, int y) {
    return find(x) == find(y);
}

// Возвращение количества множеств
int UF::count() {
    return amount;
}
```