коэффициенты приоритетности при оптимизации. В данном демонстрационном примере больший приоритет оптимизации имеют учебные группы. При необходимости, больший приоритет могут иметь преподаватели, или приоритеты могут быть равны.

Литература

1. Прихожий, А.А. Распределенная и параллельная обработка данных / А.А. Прихожий. – Минск: БНТУ, 2016. – 90 с.

2. Prihozhy, A.A. Analysis, transformation and optimization for high performance parallel computing / A.A. Prihozhy. – Minsk: BNTU, 2019. – 229 p.

3. Prihozhy, A.A., Karasik O.N. Inference of shortest path algorithms with spatial and temporal locality for big data processing. Восьмая Межд. научно-практическая конференция «BIG DATA and Advanced Analytics», Минск, 11-12 мая 2022 года. – Минск: Бестпринт, 2022. – С. 56-66.

4. Prihozhy A. A. Optimization of data allocation in hierarchical memory for blocked shortest paths algorithms / A.A. Prihozhy // System analysis and applied information science. – 2021, no. 3. – P. 40 – 50.

УДК 519.172.1

ОПТИМИЗАЦИЯ СТРУКТУРЫ ДАННЫХ «ДЕРЕВО ОТРЕЗКОВ»

Кихтенко О.Ю.
Научный руководитель – Борисова И.М., ст. преподаватель

Основная идея структуры данных «дерево отрезков» –алгоритм, который разбивает массив на отрезки, в которых уже что-то посчитано. Дерево отрезков позволяет эффективно (т.е. за асимптотику O(log n)) реализовать операции следующего вида: нахождение суммы/минимума элементов массива в заданном отрезке (a[l...r], где l и r поступают на вход алгоритма), при этом дополнительно возможно изменение элементов массива: как изменение значения одного элемента, так и изменение элементов на целом подотрезке массива (т.е. разрешается присвоить всем элементам a[l...r] какое-либо значение, либо прибавить ко всем элементам массива какое-либо число).

Есть несколько способов реализации и хранения данной структуры данных, которые имеют свои преимущества и недостатки и в данной работе будут отражены самые основные из них.

Древовидная структура может быть реализована на указателях или на массивах с пред просчётом координат вершин. Ниже приведена универсальная математическая модель для построения дерева суммы. Модель имеет вид бинарного дерева (Рис. 1).

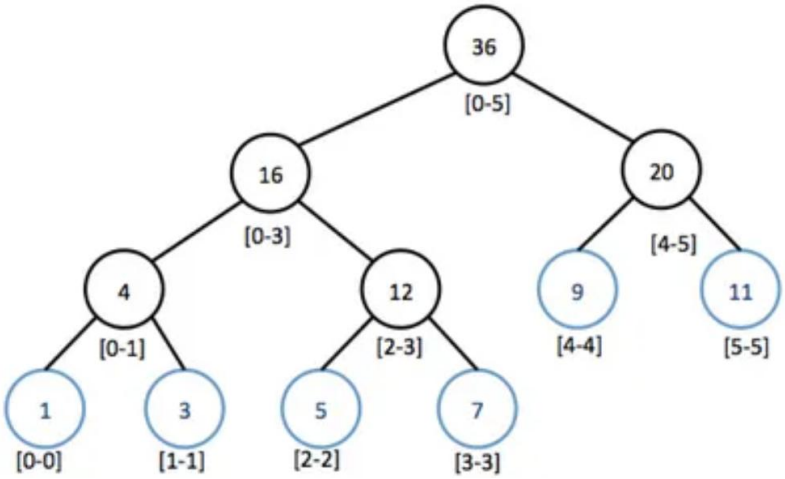

Рис.1. Математическая модель построения дерева суммы

Самый простой способ реализовать дерево отрезков – это явно сохранить все, что нам нужно, в узле: включая границы сегмента массива, сумму и указатели на его дочерние элементы. Преимуществом такого подхода является быстрота написания кода при хороших знаниях ООП и простая идея хранения и доступа к данным, однако минусами будут сложность поиска ошибок, скорость обработки и доступа к данным, так как используются ссылки.

С использованием ООП, можно рекурсивно реализовать дерево отрезков самым явным образом (Рис. 2):

```
struct segtree {
    int lb, rb;
    int s = 0;
    segtree *l = nullptr, *r = nullptr;

    segtree(int lb, int rb) : lb(lb), rb(rb) {
        if (lb + 1 < rb) {
            int m = (lb + rb) / 2;
            l = new segtree(lb, m);
            r = new segtree(m, rb);
        }
    }

    void add(int k, int x) { /* react to a[k] += x */ }
    int sum(int k) { /* compute the sum of the first k elements */ }
};
```

Рис.2. Фрагмент программного кода хранения структуры данных

Однако поскольку дерево отрезков является типом двоичного дерева, можно использовать макет для хранения его узлов в одном большом массиве и использовать индексную арифметику для навигации по нему.

Более формально определяется узел 1 как корень, содержащий сумму всего массива [0, n). Затем для каждого узла v, соответствующего диапазону [l, r], мы определяем:

– Узел 2v должен быть его левым дочерним элементом, соответствующим диапазону: $[l, \lfloor \frac{l+r}{2} \rfloor)$

– узел (2v+1) должен быть его правым дочерним элементом, соответствующим диапазону: $[\lfloor \frac{2l+r}{} \rfloor, r)$

Когда n – совершенная степень двойки, этот макет очень хорошо упаковывает все дерево. Однако, когда n не является степенью двойки, макет перестает быть компактным: хотя все еще есть ровно (2n−1) узлов, независимо от того, как мы разделяем сегменты, они больше не отображаются идеально в диапазоне [1, 2n).

Преимуществом этой реализации является простота и легкость написания кода, легкое чтение кода, а доступ к элементам требует меньше времени, однако к достаточно существенным минусам можно отнести количество памяти, которое необходимо для работы алгоритма и замедленная скорость написания.

Важной частью научной работы является визуализация графической составляющей алгоритма. Проект написан с использованием Microsoft Visual Studio 2019 и технологии пользовательского интерфейса .NET – Windows Forms.

Программа имеет два окна с пользовательским графическим интерфейсом. Первое окно для входных данных, а второе является основным и графически демонстрирует работу алгоритма с поэтапным обходом и установкой меток (Рис. 2).

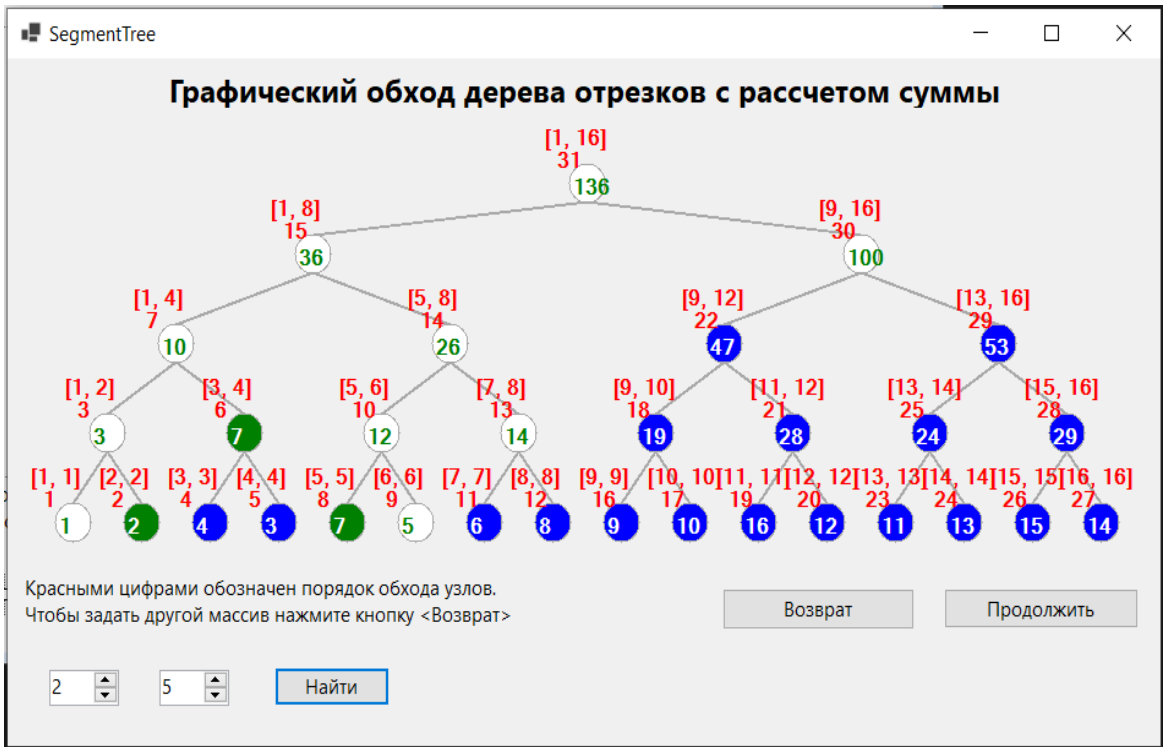

Рис 3. Скриншот выполнения программного кода

Данная работа имеет научный интерес, и основная суть исследования в оценке сложности и оптимальности различных способов хранения структуры данных, а также реализация визуального обхода дерева для лучшего понимания работы алгоритма.

Литература

1. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2017. – 1280 с.

2. Сайт e-maxx.ru, – [Электронный ресурс] Режим доступа: URL: https://e-maxx.ru/algo/segment_tree – Дата доступа: 21.04.2023

3. Сайт en.algorithmica.org. – [Электронный ресурс] Режим доступа: URL: https://en.algorithmica.org/hpc/data-structures/segment-trees/ – Дата доступа: 21.11.2022