

УДК 004.42

Симметричная обработка данных: оптимизация алгоритмов с применением двусвязных списков

Яковенко Яков Владимирович sansskeletons2005@yandex.ru

Борсук Наталья Александровна borsuk.65@yandex.ru

КФ МГТУ им. Н.Э. Баумана, Калуга, Россия

Исследованы двусвязные списки как особая структура данных, обладающая уникальными характеристиками и применимая в широком спектре прикладных задач. Новизна статьи заключается в подходе к оптимизации симметричной обработки данных в двусвязных списках, что повышает производительность структуры при решении задач, требующих двустороннего доступа к данным и возможности динамического обновления структуры. В исследовании также приводится пример реализации алгоритма на C++, позволяющего сократить время обработки и затраты памяти в операциях, предполагающих попарное вычисление разностей значений элементов списка с обоих концов. Применение представленных подходов в задачах с высокими требованиями к эффективности обработки данных подчеркивает значимость двусвязных списков в контексте современных алгоритмов и приложений.

Ключевые слова: двусвязные списки, структуры данных, оптимизация алгоритмов, обработка данных, динамические данные, симметричная обработка, параллельная обработка, вставка и удаление, вычислительная эффективность

Введение. Двусвязные списки представляют собой уникальный тип линейных структур данных, в которых каждый узел включает указатели на следующий и предыдущий элементы, что позволяет перемещаться по списку в обоих направлениях. В отличие от других структур, таких как массивы или однонаправленные списки, двусвязные списки обеспечивают более гибкий доступ к данным, позволяя выполнять операции вставки и удаления с минимальными затратами на перемещение элементов. Эти особенности делают структуру востребованной в задачах, связанных с обработкой динамических данных в режиме реального времени, что особенно важно для приложений, работающих в графических интерфейсах, системах симуляции и моделирования [1].

Двусторонняя организация данных позволяет использовать двусвязные списки в задачах, где важно учитывать взаимосвязи между удаленными элементами, например при реализации алгоритмов с попарной обработкой. Кроме того, двусвязные списки активно применяются в ситуациях, когда данные необходимо непрерывно обновлять в процессе работы программы. Это может быть, например, история действий пользователя в графических редакторах или списки задач в реальном времени, где данные меняются по мере появления новых событий.

Основные особенности и уникальность двусвязных списков. Двусвязные списки обеспечивают уникальную гибкость благодаря своей двусторонней

структуре. Возможность перемещения в обоих направлениях делает эту структуру данных особенно полезной для задач, где требуется доступ к данным с обоих концов списка [2]. Например, в приложениях, требующих частого использования функции отмены действий, таких как графические редакторы, двусвязные списки позволяют хранить последовательность действий, которую пользователь может изменять в любом направлении. Это обеспечивает удобство и повышает производительность при реализации интерфейсов с элементами интерактивности, где необходимо учитывать историю действий.

Помимо этого, двусвязные списки позволяют значительно упростить операции вставки и удаления, так как изменения в середине списка не требуют сдвига элементов. В отличие от массивов, где для вставки нового элемента или удаления существующего необходимо изменять весь массив, двусвязный список позволяет добавлять или удалять элементы, сохраняя связь между соседними элементами и минимизируя количество перемещений. Такая гибкость делает двусвязные списки идеальным выбором для реализации задач, требующих непрерывного обновления структуры, например, в симуляторах или игровых приложениях, где объекты могут появляться и исчезать в процессе выполнения программы.

Двусвязные списки также поддерживают симметричную обработку данных, что особенно полезно для вычислений, где требуется попарное сравнение удаленных элементов. Это позволяет использовать два указателя для одновременного обхода с начала и конца списка, что снижает затраты на вычисления и делает структуру более эффективной. Например, для задачи вычисления произведения разностей между удаленными элементами можно применить двусторонний обход, что позволяет сократить количество операций и значительно повысить производительность при обработке больших объемов данных [3].

Например, на языке C++ была реализована программа, демонстрирующая процесс вычисления произведения разностей между удаленными элементами двусвязного списка (рис. 1).

Пример прикладной задачи. Симметричное вычисление разностей демонстрирует уникальные возможности двусвязного списка для эффективной обработки данных [4]. В данной задаче начинаем обход списка с противоположных концов, выполняя операции попарного вычитания значений элементов, что является эффективным подходом для задач, связанных с симметричным доступом к данным. Реализация данного подхода позволяет не только сократить затраты на вычисления, но и оптимизировать работу структуры за счет сокращения объема данных, которые необходимо обрабатывать. Такой подход может быть применен в графических приложениях, экономических расчетах, а также в задачах, связанных с анализом данных, где требуется одновременный доступ к элементам с обоих концов списка.

Оптимизация алгоритма. В данном алгоритме можно провести несколько оптимизаций для повышения производительности и уменьшения использования памяти. Вот ключевые моменты, на которые стоит обратить внимание.

```

struct List {
    Node* first;
    Node* last;

    List() : first(nullptr), last(nullptr) {}

    bool is_empty() {
        return first == nullptr;
    }

    void push_back(double _val) {
        Node* p = new Node(_val);
        if (is_empty()) {
            first = last = p;
        } else {
            last->next = p;
            p->prev = last;
            last = p;
        }
    }

    void print() {
        Node* p = first;
        while (p) {
            cout << p->val << " ";
            p = p->next;
        }
        cout << endl;
    }
};

```

Рис. 1. Реализация двусвязного списка

```

double calculate_product(int n) {
    double product = 1;
    Node* forward = first;
    Node* backward = last;

    while (forward->next->next && backward->prev->prev) {
        product *= (forward->val - backward->val);
        forward = forward->next->next;
        backward = backward->prev->prev;
    }

    return product;
}

```

Рис. 2. Реализация задания

Проверка выхода за границы списка. В текущей реализации в функции `calculate_product` проверяется, что оба указателя (`forward` и `backward`) находятся на нужных позициях. Однако код `while (forward->next && backward->prev)` может привести к ошибке, если количество узлов в списке меньше, чем ожидалось. Оптимизация этой проверки позволит избежать излишних проверок, повысив устойчивость и производительность алгоритма.

Использование временных переменных для расчета разности значений: Каждый раз, когда `forward` и `backward` перемещаются на следующий шаг, происходит вычитание их значений `forward->val — backward->val`. Чтобы избежать лишних обращений к памяти, можно использовать временные переменные, что ускорит работу программы, особенно при больших объемах данных [5].

Удаление ненужных указателей после завершения работы: в текущем алгоритме указатели `forward` и `backward` продолжают существовать даже после выполнения всех операций. Освобождение памяти для этих указателей сразу после завершения цикла `while` поможет снизить вероятность утечек памяти.

```
double calculate_product(int n) {
    if (n <= 0 || !first) return 0;

    double product = 1;
    Node* forward = first;
    Node* backward = last;

    while (forward && backward) {
        double forwardVal = forward->val;
        double backwardVal = backward->val;

        product *= (forwardVal - backwardVal);
        forward = forward->next;
        backward = backward->prev;

        // Условие для выхода из цикла
        if (!forward || !backward) break;
    }

    // Освобождение памяти для указателей
    forward = nullptr;
    backward = nullptr;

    return product;
}
```

Рис. 3. Оптимизация алгоритма

Параллельная обработка для вычисления разностей: так как разности между узлами forward->val и backward->val вычисляются независимо на каждом шаге, возможно внедрение параллельной обработки. В зависимости от среды выполнения и поддержки многопоточности, это может существенно ускорить процесс вычислений, особенно для длинных списков.

Вынос вычислений вне цикла для сокращения числа обращений к памяти: оптимизация цикла while путем выноса некоторых вычислений, таких как проверка условий на каждом шаге, позволяет сократить количество обращений к данным, что в долгосрочной перспективе снизит затраты на вычисления и ускорит работу программы.

Оптимизации, реализованные в данном алгоритме, позволяют значительно повысить его производительность и сделать более эффективным при обработке больших объемов данных. Упрощение операций за счет использования временных переменных и строгий контроль границ списка подчеркивают важность продуманного подхода к разработке алгоритмов, что особенно актуально в современных приложениях, работающих с динамическими данными.

Заключение. Двусвязные списки представляют собой уникальную и гибкую структуру данных, которая нашла применение во многих прикладных задачах. Их особенности, такие как двусторонний доступ к элементам, поддержка циклической структуры и возможность симметричной обработки данных, делают их незаменимыми для приложений, требующих высокой адаптивности и эффективности. Данная работа показала, что двусвязные списки обладают потенциалом для улучшения производительности в задачах с двусторонним доступом к данным и симметричными вычислениями, что особенно актуально для систем, работающих с динамическими данными в реальном времени.

Новизна исследования заключается в оптимизации алгоритмов обработки данных с использованием двусвязных списков, что ранее не получало должного внимания в литературе. Применение временных переменных для сокращения обращений к памяти и улучшение проверки границ списка повысили как производительность, так и надежность алгоритма. Это открывает новые возможности для использования двусвязных списков в современных высоконагруженных приложениях, таких как системы управления данными, графические интерфейсы и симуляционные модели.

Кроме того, работа подчеркивает важность адаптации алгоритмических подходов к специфике решаемых задач, что позволяет разрабатывать более эффективные и масштабируемые решения. Перспективы дальнейших исследований могут включать изучение применения двусвязных списков в параллельных вычислениях и оптимизацию под специфические требования различных областей, таких как обработка больших данных и машинное обучение.

Таким образом, исследование двусвязных списков и их возможностей не только подтверждает их значимость в информатике, но и открывает новые горизонты для будущих разработок и исследований, направленных на создание более эффективных и адаптивных алгоритмических решений.

Литература

- [1] Абакаров Г.М. Анализ и оценка многопоточности для эффективности приложений. *Вестник науки*, 2023, т. 1, № 8, с. 6–8.
- [2] Байджаев М., Байраммырадов О., Башимов С. Эволюция языков программирования: от императивных к функциональным. *Вестник науки*, 2024, т. 3, № 10 (79), с. 5–8.
- [3] Лукоявнов Е.В., Грузликов А.М. Синтез иерархической диагностической модели потоковой вычислительной системы реального времени. *Научно-технический вестник информационных технологий, механики и оптики*, 2020, т. 20, № 5, с. 5–8.
- [4] Горбушин В.А., Андриенко Ю.А. Исследование реактивного программирования как набора технологий и как завершенной парадигмы. *Философия и культура*, 2024, т. 12, № 8, с. 20–24.
- [5] Зыков С.В., Золотухин С.А., Золотухина М.А. Разработка метода и определение нового условия безопасности движения в состоянии многопоточности на основе математического моделирования. *Модели, системы, сети в экономике, технике, природе и обществе*, 2023, № 3, с. 98–114.

Symmetrical data processing: optimization of algorithms using doubly-linked lists

Yakovenko Y.V.

sansskeletsans2005@yandex.ru

Borsuk N.A.

borsuk.65@yandex.ru

BMSTU (KB), Kaluga, Russia

This paper examines doubly-linked lists as a special data structure with unique characteristics and applicable to a wide range of applied problems. The novelty of the paper lies in the approach to optimizing symmetric data processing in doubly-linked lists, which improves the performance of the structure when solving problems that require two-way access to data and the ability to dynamically update the structure. The study also provides an example of implementing an algorithm in C++ that allows reducing processing time and memory costs in operations that involve pairwise calculation of differences in the values of list elements from both ends. The application of the presented approaches to problems with high requirements for data processing efficiency emphasizes the importance of doubly linked lists in the context of modern algorithms and applications.

Keywords: doubly linked lists, data structures, algorithm optimization, data processing, dynamic data, symmetric processing, parallel processing, insertion and deletion, computational efficiency