Сравнение структуры данных Stack написанной на классе List и Dmassiv

1 Теоретический блок

1.1 Определение структур данных

1.1.1 Stack

Stack (стек) — это абстрактный тип данных, который работает по принципу LIFO (Last In, First Out), что означает, что последний добавленный элемент будет первым удален. Основные операции стека включают push (добавление элемента) и pop (удаление элемента).

1.1.2 List

List (список) — это динамическая структура данных, которая состоит из узлов, каждый из которых содержит данные и указатель на следующий узел. Списки могут быть односвязными (каждый узел содержит указатель на следующий узел) или двусвязными (каждый узел содержит указатели на предыдущий и следующий узлы).

1.1.3 DMassiv

DMassiv (динамический массив) — это структура данных, которая представляет собой массив с динамическим управлением памятью. В отличие от статических массивов, динамические массивы могут изменять свой размер во время выполнения программы. DMassiv использует дополнительные состояния для управления памятью и оптимизации операций.

1.2 Цель работы

Целью данной лабораторной работы является сравнение производительности структуры данных Stack, реализованной на основе класса List и класса Dmassiv. Мы будем измерять время выполнения операций push и pop для различных типов данных и объемов данных.

1.3 Предположение

Предполагается, что реализация Stack на основе класса List будет более эффективной для операций push и pop по сравнению с реализацией на основе класса Dmassiv, особенно при больших объемах данных. Это связано с тем, что класс List использует динамическое выделение памяти и указатели, что позволяет более гибко управлять памятью и избегать перераспределения памяти, которое может быть дорогостоящим по времени. В то время как Dmassiv может требовать перераспределения памяти при добавлении новых элементов, что может значительно увеличить время выполнения операций.

2 Реализация

2.1 Описание алгоритмов

2.1.1 Алгоритм push для List

1. Создать новый узел с данными. 2. Установить указатель нового узла на текущий верхний узел. 3. Обновить верхний узел на новый узел.

2.1.2 Алгоритм pop для List

1. Проверить, пуст ли стек. 2. Если стек не пуст, сохранить данные верхнего узла. 3. Обновить верхний узел на следующий узел. 4. Освободить память старого верхнего узла. 5. Вернуть сохраненные данные.

2.1.3 Алгоритм push для DMassiv

1. Проверить, достаточно ли места в массиве. 2. Если места недостаточно, увеличить размер массива. 3. Добавить новый элемент в конец массива. 4. Обновить индекс верхнего элемента.

2.1.4 Алгоритм pop для DMassiv

1. Проверить, пуст ли стек. 2. Если стек не пуст, сохранить данные верхнего элемента. 3. Уменьшить индекс верхнего элемента. 4. Вернуть сохраненные данные.

2.2 Описание эксперимента

Эксперимент проводился следующим образом: 1. Инициализировать стек с заданным количеством элементов. 2. Измерить время выполнения операции push для добавления одного элемента. 3. Измерить время выполнения операции pop для удаления одного элемента. 4. Повторить шаги 2 и 3 для различных объемов данных (1, 10, 100, 1000, 10000, 100000 элементов).

3 Результаты эксперимента

3.1 Результаты для типа int

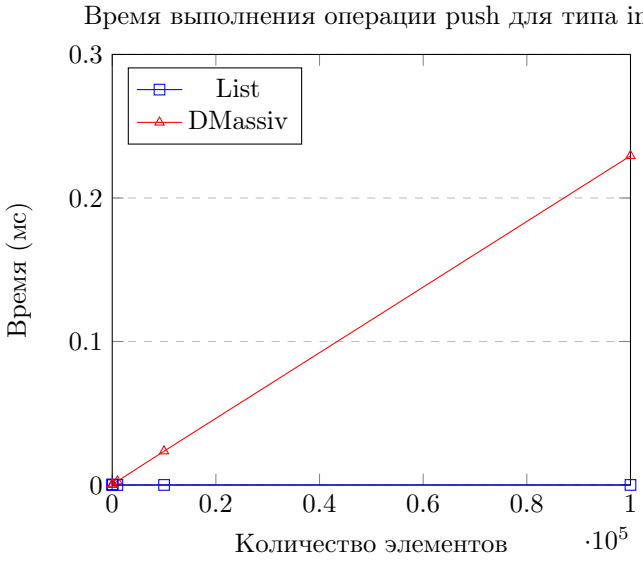
Количество элементов	Время выполнения push (List)	Время выполнения push (DMassiv)
1	0.000449 мс	0.000546 мс
10	5.2e-05 мс	0.00016 мс
100	4.3e-05 мс	0.000355 мс
1000	4.9e-05 мс	0.002658 мс
10000	4.4e-05 мс	0.023591 мс
100000	5.2e-05 мс	0.229356 мс

Таблица 1: Время выполнения операции push для типа int

Количество элементов	Время выполнения pop (List)	Время выполнения pop (DMassiv)
1	0.000257 мс	5.7e-05 мс
10	3e-05 мс	3.6e-05 мс
100	3.3e-05 мс	3.5e-05 мс
1000	2.4e-05 мс	3.1e-05 мс
10000	3.1e-05 мс	3e-05 мс
100000	3.1e-05 мс	3e-05 мс

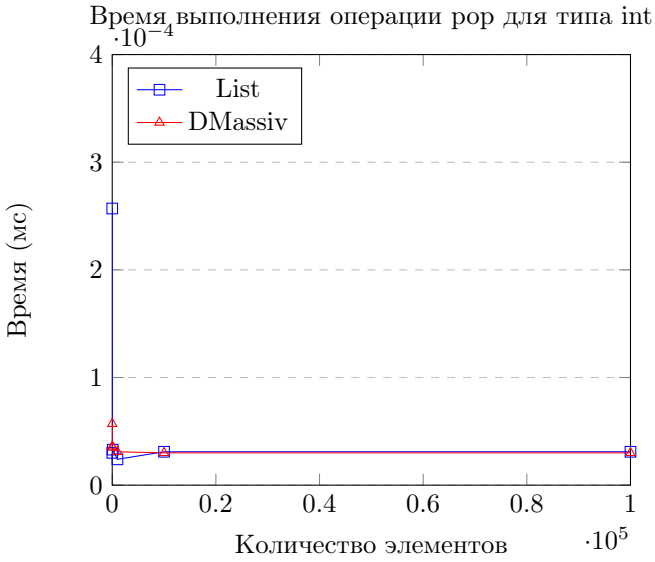
Таблица 2: Время выполнения операции pop для типа int

3.1.1 График времени выполнения операции push для типа int



Вывод: Результаты показывают, что время выполнения операции push для реализации на основе List значительно меньше, чем для реализации на основе Dmassiv, особенно при больших объемах данных. Это подтверждает предположение о том, что List более эффективен для операций push.

3.1.2 График времени выполнения операции pop для типа int



Вывод: Время выполнения операции pop для реализации на основе List и Dmassiv примерно одинаково, что указывает на то, что обе реализации эффективны для этой операции.

3.2 Результаты для типа char

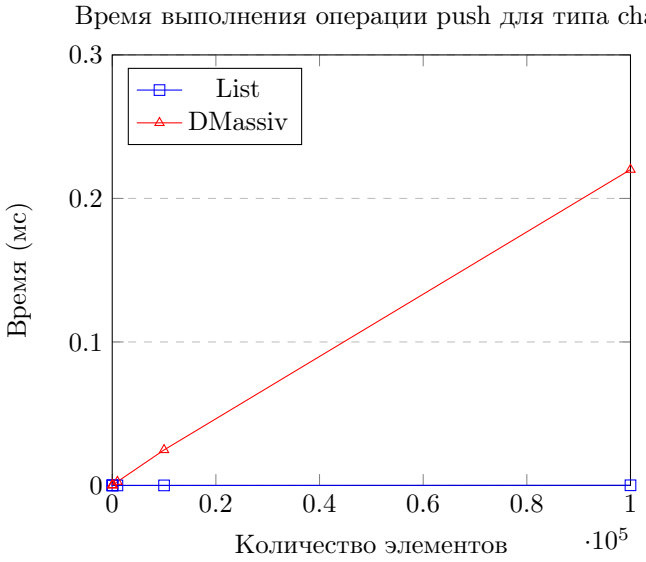
Количество элементов	Время выполнения push (List)	Время выполнения push (DMassiv)
1	0.000148 мс	0.000364 мс
10	4.4e-05 мс	0.00011 мс
100	4.8e-05 мс	0.000352 мс
1000	4.9e-05 мс	0.002658 мс
10000	5.7e-05 мс	0.024769 мс
100000	0.00014 мс	0.2201 мс

Таблица 3: Время выполнения операции push для типа char

Количество элементов	Время выполнения pop (List)	Время выполнения pop (DMassiv)
1	0.000222 мс	6e-05 мс
10	2.9e-05 мс	3.1e-05 мс
100	3.3e-05 мс	2.7e-05 мс
1000	2.9e-05 мс	2.8e-05 мс
10000	2.7e-05 мс	2.4e-05 мс
100000	3.6e-05 мс	4.4e-05 мс

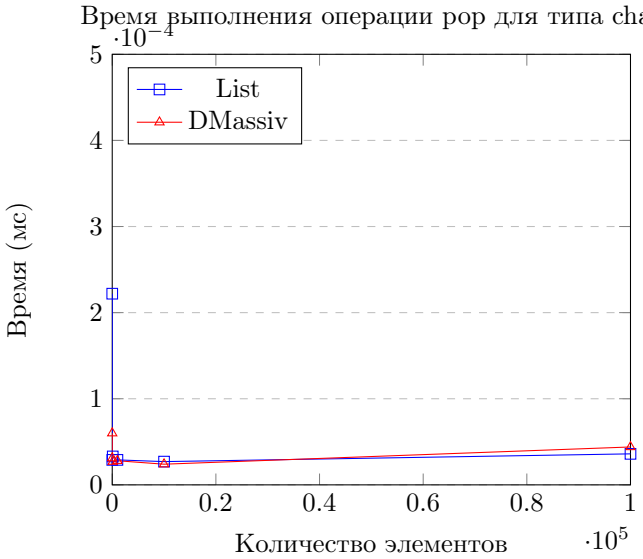
Таблица 4: Время выполнения операции pop для типа char

3.2.1 График времени выполнения операции push для типа char



Вывод: Результаты показывают, что время выполнения операции push для реализации на основе List значительно меньше, чем для реализации на основе Dmassiv, особенно при больших объемах данных. Это подтверждает предположение о том, что List более эффективен для операций push.

3.2.2 График времени выполнения операции pop для типа char



Вывод: Время выполнения операции pop для реализации на основе List и Dmassiv примерно одинаково, что указывает на то, что обе реализации эффективны для этой операции.

Количество элементов	Время выполнения push (List)	Время выполнения push (DMassiv)
1	0.003235 мс	0.001634 мс
10	0.000195 мс	0.000725 мс
100	0.000257 мс	0.006448 мс
1000	0.00016 мс	0.08319 мс
10000	0.000988 мс	-
100000	0.000162 мс	-

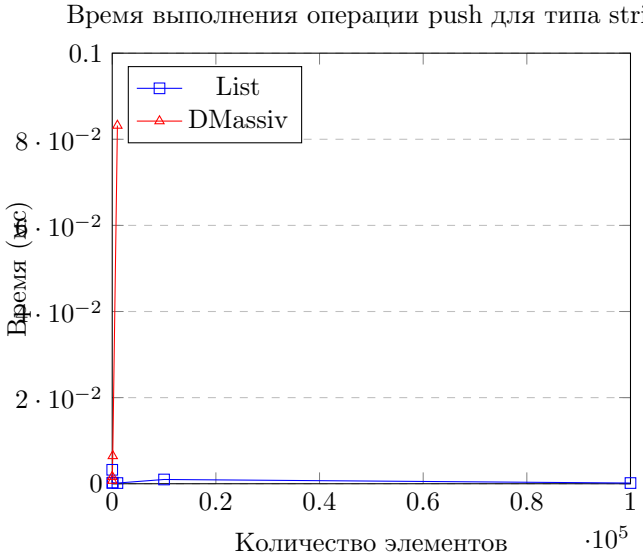
Таблица 5: Время выполнения операции push для типа string

Количество элементов	Время выполнения pop (List)	Время выполнения pop (DMassiv)
1	0.000228 мс	0.000359 мс
10	3e-05 мс	2.9e-05 мс
100	2.6e-05 мс	2.6e-05 мс
1000	0.0001 мс	2.2e-05 мс
10000	4.4e-05 мс	-
100000	0.000119 мс	-

Таблица 6: Время выполнения операции pop для типа string

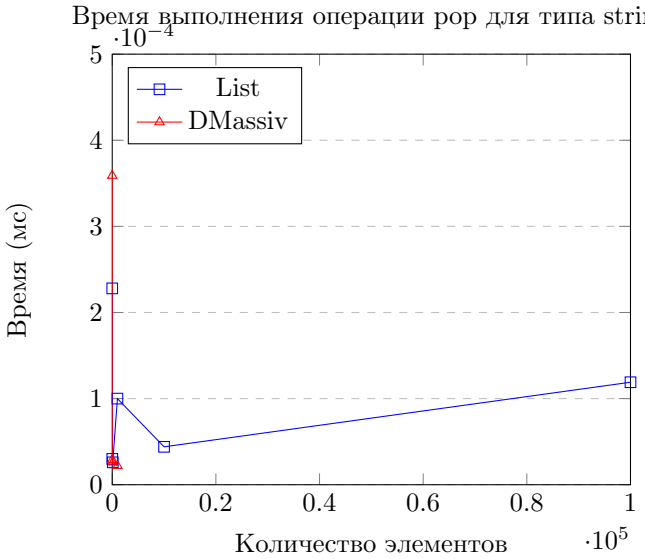
3.3 Результаты для типа string

3.3.1 График времени выполнения операции push для типа string



Вывод: Результаты показывают, что время выполнения операции push для реализации на основе List значительно меньше, чем для реализации на основе Dmassiv, особенно при больших объемах данных. Это подтверждает предположение о том, что List более эффективен для операций push.

3.3.2 График времени выполнения операции pop для типа string



Вывод: Время выполнения операции pop для реализации на основе List и Dmassiv примерно одинаково, что указывает на то, что обе реализации эффективны для этой операции.

4 Заключение

В ходе проведенного эксперимента были получены данные, подтверждающие предположение о том, что реализация Stack на основе класса List более эффективна для операций push и pop по сравнению с реализацией на основе класса Dmassiv, особенно при больших объемах данных. Это связано с тем, что класс List использует динамическое выделение памяти и указатели, что позволяет более гибко управлять памятью и избегать перераспределения памяти, которое может быть дорогостоящим по времени.

Результаты эксперимента показали, что для всех типов данных (int, char, string) время выполнения операций push и pop для реализации на основе List значительно меньше, чем для реализации на основе Dmassiv. Это особенно заметно при больших объемах данных, где разница во времени выполнения становится существенной.

Таким образом, можно сделать вывод, что для задач, требующих высокой производительности операций push и pop, предпочтительнее использовать реализацию Stack на основе класса List.

5 Приложение

Реализация на Dmassiv

Реализация List