**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 1**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **«Множества»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3311 | Баймухамедов Р. Р. |  |
| Преподаватель | Манирагена В. |  |

Санкт-Петербург

2024

**Введение**

**Цель работы:**

Исследование четырёх способов хранения множеств в памяти ЭВМ

**Задание на обработку множеств:** (Вариант 21 - Десятичные цифры):

Множество, содержащее цифры, имеющиеся в *A* или в *B* или являющиеся общими для *C* и *D*

**Формализация задания:**

E = A ∪ B ∪ (С ∩ D)

**Контрольные тесты:**

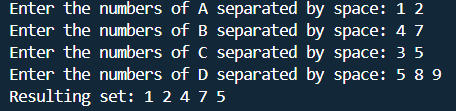


Рисунок – Массивы

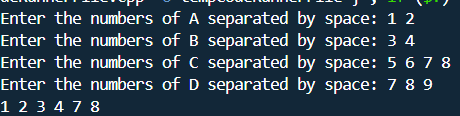


Рисунок – Списки

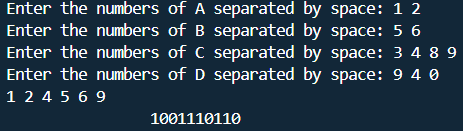


Рисунок – Массивы битов

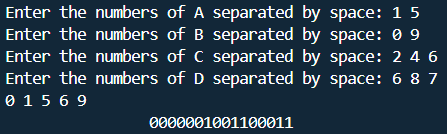


Рисунок – Машинное слово

Выше представлены контрольные примеры, использующие разные способы хранения множеств в памяти ЭВМ. Данные о множествах были введены вручную. На рис. 3 и рис. 4 также представлен ответ в виде массивов битов и машинного слова (двоичного кода) соответственно

**Временная сложность**

**Массивы:**

1. Временная сложность is\_in\_set():

Проверка, содержится ли элемент в массиве, требует прохода по всему массиву. Это операция с временной сложностью O(n) (где n — размер массива).

1. Временная сложность association() (Объединения):

Для каждого элемента массива additional проверяется его наличие в массиве result с помощью is\_in\_set(), что занимает O(n) для каждого элемента. Всего итераций будет m (где m — размер второго массива). Следовательно, временная сложность объединения — O(n \* m).

1. Временная сложность intersection() (Пересечения):

Проход по каждому элементу массива result с проверкой его наличия в массиве additional через is\_in\_set() (сложность O(m) для каждого элемента). Всего итераций будет n. Временная сложность пересечения — O(n \* m).

Общая временная сложность для массивов:

Алгоритм объединения и пересечения для массивов выполняется последовательно:

E = A ∪ B — объединение двух массивов — O(n \* m)

C ∩ D — пересечение двух массивов — O(p \* q)

E = E ∪ (C ∩ D) — снова объединение массивов — O(k \* l)

В каждом пункте у нас получается прохождение множества для элемента другого множества и следовательно общая сложность алгоритма равна O(n2)

**Списки:**

Аналогично массивам временная сложность не изменяется, только теперь переход к следующему элементу получается не по увеличению индекса, а по указателю

**Массивы битов:**

Временная сложность association() и intersection():

Эти операции выполняются на уровне битов через побитовые операторы | и &. Для каждого бита операция выполняется за O(1), и общая сложность равна размеру массива, то есть O(n), где n — количество битов (размер множества). Общая временная сложность для массивов битов равна O(n). В нашем случае это O(10) = O(1)

**Машинные слова:**

Временная сложность association() и intersection():

Как и в битовых массивах, эти операции выполняются на уровне машинных слов через побитовые операторы | и &. Операция выполняется за O(1) для каждого машинного слова, так как операции над машинными словами поддерживаются процессором аппаратно. Таким образом общая временная сложность для машинных слов равна O(1)

**Результат измерения времени обработки для каждого из способов:**

Зависимость времени обработки от размера данных является линейной для каждого из способов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер попытки | Время выполнения алгоритма для представленного способа (сек) | | | |
| Массивы | Списки | Массивы битов | Машинное слово |
| 1 | 0,267\*10-5 | 0,601\*10-5 | 0,895\*10-7 | 0,611\*10-7 |
| 2 | 0,314\*10-5 | 0,634\*10-5 | 0,930\*10-7 | 0,612\*10-7 |
| 3 | 0,277\*10-5 | 0,651\*10-5 | 0,915\*10-7 | 0,674\*10-7 |
| 4 | 0,324\*10-5 | 0,643\*10-5 | 0,911\*10-7 | 0,655\*10-7 |
| 5 | 0,314\*10-5 | 0,657\*10-5 | 0,933\*10-7 | 0,633\*10-7 |
| 6 | 0,283\*10-5 | 0,642\*10-5 | 0,920\*10-7 | 0,690\*10-7 |
| 7 | 0,346\*10-5 | 0,634\*10-5 | 0,932\*10-7 | 0,686\*10-7 |
| 8 | 0,352\*10-5 | 0,639\*10-5 | 0,912\*10-7 | 0,668\*10-7 |
| 9 | 0,320\*10-5 | 0,656\*10-5 | 0,927\*10-7 | 0,696\*10-7 |
| 10 | 0,418\*10-5 | 0,666\*10-5 | 0,936\*10-7 | 0,679\*10-7 |
| Среднее значение | 0,322\*10-5 | 0,642\*10-5 | 0,921\*10-7 | 0,660\*10-7 |

**Выводы о результатах испытания способов представления множеств в памяти:**

В ходе испытаний различных способов представления множеств в памяти (списки, массивы, битовые поля и машинные слова) были получены следующие результаты:

1. **Массивы**

**Достоинства:**

1. Простота: Массивы просты в использовании и реализации. Их можно легко индексировать.
2. Быстрый доступ по индексу: Доступ к элементу массива по индексу выполняется за 𝑂(1), что делает их идеальными для ситуаций, где необходим быстрый доступ к произвольным элементам.
3. Предсказуемость памяти: Массивы выделяют память заранее, что делает их эффективными с точки зрения управления памятью при малом размере данных.

**Недостатки:**

1. Фиксированный размер: Массивы имеют фиксированный размер, который необходимо задать заранее. Из-за этого расширять массивы без полной перераспределения памяти невозможно.
2. Проблемы с памятью: Если выделен слишком большой массив, неиспользуемая память будет пустовать. Если массив мал, придётся выделять новый и копировать данные.

**Области применения:**

Эффективны в задачах, где известен размер множеств и требуется быстрый доступ к данным. Подходят для обработки статических данных или множеств с небольшими изменениями, а также в задачах, в которых важно быстро обращаться к элементам по индексу, например, в матричных вычислениях

2**. Списки**

**Достоинства:**

1. Гибкость: списки могут динамически изменяться по размеру
2. Память: эффективное использование памяти компьютера

**Недостатки**

1. Затраты на хранение дополнительных данных, таких как ссылки на элементы

**Область применения**

Подходят для работы с небольшими наборами данных, где гибкость важнее скорости, а размеры множеств заранее неизвестны. Особенно полезны, когда требуется частое добавление или удаление элементов.

**Массивы битов**

**Достоинства:**

1. Компактное хранение данных: каждый элемент представляется одним битом.
2. Быстрые побитовые операции (объединение и пересечение выполняются за O(N), где N — количество битов).
3. Экономия памяти при больших диапазонах данных

**Недостатки:**

1. Сложность работы с нечисловыми или сильно разреженными данными.

**Область применения**

Эффективны для работы с большими множествами, где элементы могут быть представлены как битовые маски. Идеальны для задач, где нужно обрабатывать данные в фиксированном диапазоне значений (например, обработка флагов или характеристик).

**Машинное слово**

**Достоинства:**

1. Очень высокая скорость операций: объединение и пересечение выполняются за постоянное время O(1).
2. Минимальное использование памяти для небольших множеств.
3. Поддержка операций на уровне процессора, что значительно ускоряет вычисления.

**Недостатки:**

1. Ограничение на количество элементов, которое можно представить в одном машинном слове (например, 32 или 64 бита).
2. Неприменимость для больших множеств, выходящих за пределы одного машинного слова.

**Область применения:**

Подходят для работы с небольшими множествами, где важны высокая скорость и минимальные затраты на память. Могут использоваться для оптимизации работы с булевыми массивами или флагами в системах реального времени.

**Заключение:**

В ходе выполнения задания успешно были применены операции над множествами для различных типов представления данных: списков, массивов, массивов битов и машинной формы. Мы провели объединение и пересечение множеств, сформировав итоговое множество, которое содержит уникальные элементы из множеств A и B, а также общие элементы из множеств C и D.

Для каждого представления были реализованы соответствующие алгоритмы, что позволило укрепить навыки работы с различными структурами данных и их эффективной обработки. Применение списков продемонстрировало гибкость и наглядность работы с последовательностями, массивы позволили эффективно управлять структурой данных с фиксированным размером, битовые операции показали преимущества работы с данными на побитовом уровне, что особенно полезно в задачах с ограниченными ресурсами, а машинная форма обеспечила минимизацию пространства и максимальную скорость выполнения операций за счёт низкоуровневого представления данных.

Решение задачи позволило продемонстрировать эффективность использования различных подходов к работе с множествами для решения практических задач по объединению, пересечению и фильтрации данных. Сравнение времени выполнения операций для различных представлений множеств даёт наглядное понимание того, как выбор структуры данных влияет на производительность и эффективность программы в целом.