МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

ОТЧЕТ

по лабораторной работе № 1

по дисциплине «Алгоритмы и структуры данных»

Тема: «Множества»

Студент гр. 3311 Шарпинский Д. А.

Преподаватель Манирагена Валенс

Санкт-Петербург

2024

Цель работы

Исследование четырёх способов хранения множеств в памяти ЭВМ.

Задание (вариант 33)

Универсум: десятичные цифры.

Составить множество E, содержащее цифры, общие для множеств A и B, но не встречающиеся ни в C, ни в D.

Формула для вычисления нового множества

Для вычисления множества E в общем случае можно воспользоваться формулой: E = A & B & !(C | D) или ее аналогом E = A & B & !C & !D, где & - логическое И, | - логическое ИЛИ, ! – логическое отрицание.

Теперь рассмотрим применение этой формулы по отдельности для каждого из способов хранения множеств в памяти.

1. Массивы

Множества могут храниться в виде массивов символов, представляющих цифры от 0 до 9. В этом случае элементы множества представлены в виде строки, а операция пересечения множеств выполняется с помощью линейного поиска элементов в другом массиве.

Операция вычисления множества E для массива реализована с помощью цикла, в котором проверяются условия принадлежности элементов к множествам A, B, C и D.

2. Связные списки

Множества также могут быть представлены в виде связных списков, где каждый элемент множества хранится в узле списка. Операции пересечения и объединения множеств выполняются путём последовательного обхода списков и проверки каждого элемента на наличие в других множествах.

Этот способ позволяет динамически изменять размер множества, но требует дополнительных затрат на управление памятью.

3. Битовые векторы

Множества можно хранить в виде битовых векторов, где каждая цифра представляется отдельным значением в массиве bool. Если элемент присутствует в множестве, соответствующий элемент устанавливается в true. Операции пересечения и объединения выполняются с помощью проверки соответствующего индекса в массиве, что делает этот способ хранения очень эффективным с точки зрения времени выполнения операций.

4. Слова (Word)

Множество можно хранить в виде целого числа (слова), где каждый бит числа соответствует элементу множества. Это также позволяет выполнять побитовые операции над множествами, обеспечивая высокую производительность.

Контрольные примеры

```
Array E: [2, 8]
Char[]
Array A: [2, 8, 9]
Array B: [0, 2, 4, 5, 8, 9]
Array C: [3, 5, 7, 9]
Array D: [4, 6]
Result is array E: E = (A && B) \ (C || D)
Array E: [2, 8]
Time to process arrays 100000000 times: 10.561 seconds.

Lists
List A: [2, 8, 9]
List B: [0, 2, 4, 5, 8, 9]
List C: [3, 5, 7, 9]
List D: [4, 6]
Result is list E: E = (A && B) \ (C || D)
List E: [2, 8]
Time to process lists 100000000 times: 29.691 seconds.

Words
Array A: [2, 8, 9]
Array B: [0, 2, 4, 5, 8, 9]
Array C: [3, 5, 7, 9]
Array D: [4, 6]
Result is word E: E = (A && B) \ (C || D)
Array E: [2, 8]
Time to process words 100000000 times: 0.543 seconds.

Bit vectors
Set A: [2, 8, 9]
Set B: [0, 2, 4, 5, 8, 9]
Set C: [3, 5, 7, 9]
Set D: [4, 6]
Result is set E: E = (A && B) \ (C || D)
Set E: [2, 8]
Time to process bit sets 100000000 times: 4.168 seconds.
```

```
Array E: []
Char[]
Array A: [3, 6, 8]
Array B: [3, 7, 8, 9]
Array C: [3, 4, 9]
Array D: [3, 5, 7, 8]
Result is array E: E = (A && B) \ (C || D)
Array E: []
Time to process arrays 100000000 times: 6.25 seconds.

Lists
List A: [3, 6, 8]
List B: [3, 7, 8, 9]
List C: [3, 4, 9]
List D: [3, 5, 7, 8]
Result is list E: E = (A && B) \ (C || D)
List E: []
Time to process lists 100000000 times: 5.776 seconds.

Words
Array A: [3, 6, 8]
Array B: [3, 7, 8, 9]
Array C: [3, 4, 9]
Array D: [3, 5, 7, 8]
Result is word E: E = (A && B) \ (C || D)
Array E: []
Time to process words 100000000 times: 0.539 seconds.

Bit vectors
Set A: [3, 6, 8]
Set B: [3, 7, 8, 9]
Set C: [3, 4, 9]
Set D: [3, 5, 7, 8]
Result is set E: E = (A && B) \ (C || D)
Set E: []
Time to process bit sets 100000000 times: 4.306 seconds.
```

```
Array E: [7]
Char[]
Array A: [2, 5, 7, 8]
Array B: [0, 1, 2, 4, 5, 7]
Array C: [2, 4]
Array D: [0, 5, 8]
Result is array E: E = (A && B) \ (C || D)
Array E: [7]
Time to process arrays 100000000 times: 9.35 seconds.

Lists
List A: [2, 5, 7, 8]
List B: [0, 1, 2, 4, 5, 7]
List C: [2, 4]
List D: [0, 5, 8]
Result is list E: E = (A && B) \ (C || D)
List E: [7]
Time to process lists 100000000 times: 19.071 seconds.

Words
Array A: [2, 5, 7, 8]
Array B: [0, 1, 2, 4, 5, 7]
Array C: [2, 4]
Array D: [0, 5, 8]
Result is word E: E = (A && B) \ (C || D)
Array E: [7]
Time to process words 100000000 times: 0.532 seconds.

Bit vectors
Set A: [2, 5, 7, 8]
Set B: [0, 1, 2, 4, 5, 7]
Set C: [2, 4]
Set D: [0, 5, 8]
Result is set E: E = (A && B) \ (C || D)
Set E: [7]
Time to process bit sets 100000000 times: 4.076 seconds.
```

Оценка сложности

1. Массивы:

Так как происходит линейный перебор каждого из массивов для формирования нового множества E, сложность будет зависеть от количества элементов в массиве. Предположим, что каждый массив содержит n элементов. Поиск элемента в другом массиве требует линейного времени $O(n)$. Операция пересечения множеств A и B, а также исключение элементов множеств C и D также требует обхода всех элементов.

Итого, сложность операции для массивов: $O(n^2)$, где n — размер массивов.

2. Связные списки:

В случае со связными списками для каждого элемента списка A выполняется поиск в списках B, C и D. Каждая операция поиска в списке требует линейного обхода, что аналогично массивам. Для поиска элемента в другом списке требуется O(n) времени.

Таким образом, для пересечения и исключения элементов нужно обрабатывать все элементы каждого списка.

Общая сложность для связных списков:

Итого, сложность операции для списков: $O(n^2)$, где n — размер списков.

Связные списки могут иметь дополнительные накладные расходы, связанные с динамическим выделением памяти и управлением указателями, что может увеличивать фактическое время выполнения.

3. Битовые векторы:

Битовые векторы позволяют моментально получить доступ к информации о том, содержится ли в множестве данный элемент. Поскольку алгоритм проверки всегда осуществляется для n = 10 = const, где n – размер полученного в задании универсума, то временная сложность от константы будет составлять O(1).

4. Слова (Word):

Машинные слова для данного универсума являются самым эффективным способом хранения информации. Поскольку все операции (пересечение, объединение, исключение) делаются за O(1), поскольку требуют единичного побитового сравнения двух чисел.

Результаты измерения времени

Рассматривались измерения времени для 100000000 (100 млн) итераций.

Ниже приведены результаты одного из тестов.

Массивы: 5.268 scекунд.

Списки: 6.407 секунд.

Битовые векторы: 3.854 секунд.

Слова: 0.537 секунд.

Выводы

По итогам измерений самым эффективным способ – использование машинных слов. Поскольку в данном варианте рассматривается небольшой универсум (10 элементов), то использование short int размером 2 байта позволяет максимально эффективно вычислять пятое множество. Очевидно, что в данном случае, связные списки будут иметь худшую эффективность, поскольку требуют работы с памятью и полного обхода каждого списка. Аналогично с массивами. Битовые векторы показывают себя более эффективными по сравнению с массивами и списками, но имеют худшее время относительно машинных слов. Это связано с большим размером вектора в памяти и тем, что работа с массивом все еще сложнее, чем простое битовое сравнение. Важно понимать, что подобные выводы применимы только к данному универсуму.