Пояснительная записка

I. **Подходы к решению и выбор решения**.

(a) **Метод перебора всех возможных вариантов**. Суть данного метода заключается в переборе всех возможных значений от 1 до n^2 (для поля размером $n \times n$) и последующей проверкой заполненного пазла судоку на валидность (т.е. соответствие всем правилам). Ясно, что такой алгоритм хоть и позволяет найти все возможные решения, но требует слишком большого объема вычислительных и временных ресурсов.

(b) **Алгоритм с возвратом (backtracking algorithm [1])**. Суть данного подхода легче всего представить в виде деревовидного графа, в котором каждый узел сопоставляется некоторой ячейке, тогда ветви из этого узла визуализируют возможные варианты подстановки чисел в этом графе. При этом ветви исследуются в общем случае в неопределённом порядке, однако в прикладных программах порядок часто закладывается на программном уровне (это не влияет на корректность алгоритма). Тогда алгоритм поиска решения с возвратом представляет суть поиск в глубину в названном выше графе. Пример такого решения [2].

(c) **Алгоритм X**. Алгоритм X [7] был представлен Дональдом Кнутом в работе «Dancing Links» [3] для решения NP-полной задачи точного полного покрытия [5]. Формулировку проблемы точного полного покрытия в общем случае можно найти, например, в [4]. Для решения задачи полного точного покрытия составляется матрица ограничений. Эта матрица связывает всех возможных кандидатов с ограничениями, которые они удовлетворяют.
Рассмотрим пример из [6].
Есть множество $S = \{1, 2, 3, 4, 5\}$, Y - это набор подмножеств этого множества, $Y = \{A = \{1, 2\}, B = \{2, 3\}, C = \{1, 5\}, D = \{1, 4\}, E = \{5\}\}$.
Задача состоит в том, чтобы выбрать из Y такие элементы $\widetilde{Y}$, что каждый элемент из S содержится только в одном из множеств, входящих в $\widetilde{Y}$.
Для данного примера матрица ограничений может быть представлена в виде таблицы 1.
Для задачи решения судоку ограничения будут иметь четыре класса.

i. Ограничения на строку: в каждой строке должны быть все цифры из диапазона $1 \dots n$ и в одной строке не могут встречаться две одинаковые цифры.

ii. Ограничения на столбец: в каждом столбце должны быть все цифры из диапазона $1 \dots n$ и в одном столбце не могут встречаться две одинаковые цифры.

iii. Ограничения на ячейку: в каждой ячейке должны быть все цифры из диапазона $1 \dots n$ и в одной ячейке не могут встречаться две одинаковые цифры.

iv. Ограничения на поле: в каждой поле должна стоять одна и только одна цифра из диапазона $1 \dots n$.

Таблица 1: Матрицы ограничений для задачи из множеств

Y_j/S_j	1	2	3	4	5
A	1	1	0	0	0
B	0	1	1	0	0
C	1	0	0	0	1
D	1	0	0	1	0
E	0	0	0	0	1

Уже на примере матрицы 1 видно, что количество нулей в матрице ограничений значительно превосходит количество единиц. Такие матрицы называют *разряженными*. В DLX разряженная матрица ограничений представляется в виде двумерного двусвязного списка единиц матрицы. Т.е. те клетки матрицы, где стоят единицы представляют из себя узлы вертикально связанные с заголовком матрицы и с другими узлами соответствующими единицам в той же столбце матрицы. Рассмотрим количество узлов в столбцах на примере матрицы 1. Нумерацию строк и столбцов будем начинать с нуля. В нулевом столбце матрицы (в заголовке столбца стоит число «1») будет 4 узла: один - для заголовка и 3 узла, соответствующие единицам в первом столбце. В первом столбце матрицы (в заголовке столбца стоит число «2») будет 2 узла и т.д.

Иначе говоря, вертикально связаны узлы, стоящие в одном столбце.

Узлы в структуре DLX также связаны горизонтально (поэтому список и называется двумерным). Горизонтально связаны узлы находящиеся в одной строке. Так в нулевой строке (соответствующей множеству A горизонтально связаны 2 узла).

Использование такого представления для разряженной матрицы позволяет кратно снизить затраты времени на поиска нужного узла. Так для задачи судоку размером 9×9 размер матрицы ограничений будет следующим: на каждую клетку накладывается 4 ограничения (ограничение строки, столбца, ячейки и поля), всего 81 клетка, т.о. матрица ограничений будет содержать $81 \times 4 = 324$ столбца. В строках такой матрицы ограничений будут располагается всевозможные варианты расположения цифр из диапазона $1 \dots n$ в 81 клетке. Т.о. матрица имеет $81 \times 9 = 729$ строк. Таким образом получаем, что общее количество 0 и 1 матрицы будет $324 \times 729 = 236196$, при этом в каждом столбце всего 9 единиц, а в каждой строке — всего 4 единицы, т.е. единиц в матрицы всего $9 \times 324 = 4 \times 729 = 2916$, что составляет $\approx 1,24\%$ от общего числа значений в матрицы. Таким образов эффективность такой структуры хранения не вызывает сомнений.

После создания структуры хранения данных алгоритм описывается следующим образом:

i. Для тех значений, которые даны в качестве известных в клетках таблицы покрываются столбцы и строки в которых они находятся, а сами узлы добавляются в стек решения.

ii. Если матрица после этого оказалась пуста — решение найдено.

iii. Ищем столбец DLX, в котором наименьшее число узлов по вертикали (в терминах матрицы: столбец, в котором наименьшее число единиц).

iv. Узлы связанные с полученным заголовкам столбца суть строки решения. Для каждой узла найденного столбца повторяем следующие шаги:

   A. Добавляем узел в стек

   B. Покрываем столбец в котором находится выбранный узел.

   C. Формируем DLX структуру в которой покрыты все столбцы, у которых имеется общий в выбранным узлом узел (в терминах матрицы: удаляем все столбцы, в которых единица стоит в той же строке, что и у выбранного узла).

v. Если получаем, что покрытие выполнить невозможно, то выбрасываем выбранных узел из решения, и раскрываем связанные с ним покрытые столбцы.

vi. Если покрытие удалось выполнить: рекурсивно переходим к шагу 2 алгоритма с вычеркнутыми столбцами и новым стеком решения.

II. **Решение**

Дополнительная информация содержится непосредственно в коде решения и комментариях.

Решение реализованы в виде библиотеки на языке C++, для проверки корректности работы были написаны 4 теста для наиболее часто встречающихся размеров матриц судоку: $4 \times 4, 9 \times 9, 16 \times 16, 25 \times 25$.

Исходный код на доступен по ссылке: `https://github.com/Danon128/OliTask-1`
Информация о ПО и инструкция по запуску содержится в файле 'readme.md' непосредственно в репозитории.

III. **Анализ возможностей распараллеливания алгоритма**.

Стоит сразу заметить, что Алгоритм X представляет суть алгоритм поиска в глубину с возвратом. На практике это означает, что до полной проверки нити вычисления мы не можем заранее предугадать, куда нам нужно двигаться дальше и какие варианты проверять. Следствием этого является то, что такие алгоритмы практически не параллелизуемы.

Тем не менее самым эффективной параллелизацией алгоритмы было бы решение нескольких судоку одновременно, при этом максимальная эффективность достигается при совпадении числа одновременно решаемых судоку с доступным числом потоков процессора. В таком случае каждый из потоков работает независимо, конкуренции не наблюдается.

В случае параллелизации на уровне алгоритма:
в методе *FindColumnWithMinNumberOfNodes* подсчет количества узлов внутри каждого столбца можно выделить в отдельную задачу, предварительно создав пул потоков, например, при помощи OpenMP для предполагаемого вычисления на CPU. Стоит сказать, что время затрачиваемое на создание дополнительных потоков превосходит временной выигрыш от распараллеливания решения таким методом.

Список литературы

[1] HARVEY W.D., *NONSYSTEMATIC BACKTRACKING SEARCH* — Stanford University, 1995

[2] ЕВСТРАТОВ В.В., *Рекурсивный алгоритм решения судоку с проверкой найденного решения на единственность* — Молодой ученый, 2020, № 51 (341), С. 8-11

[3] `https://www.ocf.berkeley.edu/~jchu/publicportal/sudoku/0011047.pdf` (Дата обращения: 10.02.2024)

[4] `https://en.wikipedia.org/wiki/Exact_cover` (Дата обращения: 10.02.2024)

[5] `https://www.mathreference.com/lan-cx-np,excov.html` (Дата обращения: 10.02.2024)

[6] `https://habr.com/ru/articles/462411/` (Дата обращения: 10.02.2024)

[7] https://en.wikipedia.org/wiki/Knuth27sAlgorithmX (Дата обращения: 10.02.2024)