Вопросы и Ответы

*К проекту: Linear algorithm for solution n-Queens Completion problem*

***Григорян Э****.*

***1.В комментариях довольно часто говорится о матрице решения, но в программе такой матрицы нет. С чем это связано?***

Ответ. В самом деле, довольно часто можно встретить термин «матрица решения», который используется в качестве аналога шахматной доски. При решении задачи, все события разворачиваются на матрице решения. Это как сцена театра. Выбирая некоторую строку для расположения ферзя, или проводя поиск свободной позиции в какой-либо строке, мы всегда ориентируемся на матрицу решения. Когда ведем учет индексов оставшихся свободных строк и числа свободных позиций в этих строках, мы также ориентируемся на матрицу решения. Точно также, мы ориентируемся на матрицу решения, когда ведем учет диагональных ограничений, которые накладываются на другие свободные ячейки, при расположении ферзя в некоторую ячейку. Все события по расположению ферзя в выбранной позиции и последствия этих событий, фиксируются на матрице решения. Однако, в основной части программы не создается и не используется матрица решения, так как в этом нет необходимости. Тем более, для больших значений n, где для создания матрицы решения, потребовался бы большой объем оперативной памяти (например, для шахматной доски размером **106 x 106**, потребовалось бы выделить **1012**байт памяти). Только после того, как большая часть ферзей расставлена на шахматной доске, мы создаем матрицу решения меньшего размера, куда «проекционно» сводим оставшиеся свободные строки и столбцы. (Например, для **n=106** , на последнем этапе решения задачи, создается матрица решения размера **547 х 547** только после того, как **999453** ферзей правильно расположены в тех или иных позициях). Такой подход не загружает сильно память, так как это всего лишь небольшая часть матрицы решения. Однако, это дает нам существенное преимущество в формировании алгоритма и выигрыш в скорости решения.

***2.В чем смысл базового уровня, и как они вычисляются?***

Ответ. Все возможное множество ветвей поиска решения данной задачи, можно разделить на два подмножества. Одно из них, это подмножество ветвей, которые приводят в тупик. Поэтому, когда в процессе решения задачи возникает тупиковая ситуация, мы должны вернуться назад, (т.е. выполнить процедуру **Back Tracking**(**BT**)) на один из предыдущих уровней, и снова строить ветвь поиска решения, начиная с этого уровня. В связи с этим, возникает два вопроса:

1) Сколько уровней для возврата назад должно быть?

2) На какой из предыдущих уровней нам следует возвращаться?

Здесь и далее, под уровнем мы понимаем количество правильно расставленных ферзей, независимо от последовательности их расстановки.

Очевидно, что выполнение процедуры **BT** является дополнительной нагрузкой для программы:

- для каждого уровня возврата, нужно сохранить копии всех необходимых массивов и переменных;

- на основе сохраненных копий, необходимо восстановить значения активных массивов и переменных после возвращения на данный уровень;

- нужно вновь продолжить формирование новой ветви поиска с данного уровня, в надежде, что «на этот раз повезет».

Очевидно, что наличие тупиковых ветвей поиска является объективным свойством данной задачи. И раз мы вынуждены вернуться назад, следует выбрать оптимальный уровень, учитывая, что:

- чем чаще будет выполняться процедура **BT**, и чем дальше расположен уровень для возврата назад, тем больше времени потребуется для решения задачи;

- чем ближе к **n** значение уровня возврата, тем меньше вероятность того, что формирование ветви поиска решения с этого уровня будет продуктивным.

В данном алгоритме мы формируем три уровня **estartEventBound**, **eventBound1** и **eventBound2**, для возврата назад. Мы называем эти уровни **базовыми**. Здесь **estartEventBound -** это начальный базовый уровень, который соответствует началу решения задачи. Это тот уровень, когда после ввода данных, была проведена проверка правильности композиции и выполнены необходимые операции для начала поиска решения.

*Как были получены значения базовых уровней?*

Рассмотрим список значений размера шахматной доски для моделирования n= (*10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 300, 500, 800, 1000, 3000, 5000, 10000, 30000, 50000, 80000, 105 , 3 ∗ 105 , 5 ∗ 105 , 106 , 3 ∗ 106 , 5 ∗ 106 , 107 , 3 ∗ 107 , 5 ∗ 107 , 8 ∗ 107 , 108* ), который назовем «**базовым списком значений n** для проведения вычислительных экспериментов. Для каждого значения **n** из этого списка, выполним следующие операции:

1.Для шахматной доски размера **n x n** найдем решение **n-Queens Problem**, используя только алгоритм поиска решения **randSet & randSet**. В результате, с большой вероятностью будет сформирована тупиковая ветвь, где на шахматной доске будут правильно расставлены **k** ферзей. Сформируем большую выборку таких решений. Определим среднее значение выборки (**xMean**) и стандартное отклонение (**xStd**). Определим величину **baseLevel1= xMean –3\* xStd**. Рассмотрим полученное значение **baseLevel1** в качестве приближенного значения базового **уровня 1**. Проведем аналогичные расчеты для всего списка экспериментальных значений **n** и сохраним полученные результаты в массиве **baseLevelAr1**.

2.Проведем аналогичные расчеты для всего базового списка значений **n** , но используем для решения **n-Queens Problem** только алгоритм **rand & rand**. Сохраним полученные результаты в массиве **baseLevelAr2**. Таким образом, мы получим приближенные значения двух базовых уровней для всего списка значений **n**.

3.Далее следует оптимизировать полученные значения базовых уровней, изменяя их значения с некоторым шагом в большую или меньшую сторону. Критерием оптимальности выбора будут такие значения **baseLevel1** и **baseLevel2**, при котором решение задачи комплектации большой выборки положительных композиций, приводит к минимальному числу **False Negative** решений и минимальному среднему времени счета. Сохраним оптимизированные таким образом значения базовых уровней для всего списка значений **n** в массивах **eventBoundAr1** и **eventBoundAr2**.

4.Проведем регрессионный анализ для определения зависимости значения базового уровня 2 от величины **n**. Аналогично, проведем регрессионный анализ и установим зависимость значения базового уровня 3 от величины **n**. Таким образом, мы получаем возможность определить значения базовых уровней для произвольного значения **n**.

В программе решения **n-Queens Completion Problem**, после ввода массива исходных данных, определяется размер **n** исходной шахматной доски, и на его основе, с помощью уравнений регрессии определяются значения базовых уровней: **eventBound1** и **eventBound2**.

***3.В чем смысл правил «минимального риска» и «минимального ущерба», которые используются на последнем этап решения задачи.***

Ответ. Эти два правила имеют принципиальное значение для работы алгоритма, в особенности, правило «минимального риска». Оба правила подробно описаны в публикации *Grigoryan E., Linear algorithm for solution n-Queens Completion problem,* [*https://arxiv.org/abs/1912.05935*](https://arxiv.org/abs/1912.05935)

Правило минимального риска. Пусть, в процессе решения мы расположили на шахматной доске большую часть ферзей, и на доске осталось лишь небольшое число свободных строк. Если определить число свободных позиций в каждой из оставшихся свободных строк, то может оказаться, что среди них есть строка, в которой осталась только одна свободная позиция, а все остальные позиции в данной строке закрыты. (В процессе решения, чем меньше останется свободных строк, тем больше вероятность появления подобной ситуации). Если, на данном шаге, для расположения ферзя выбрать не ту строку, в которой осталась всего одна свободная позиция, а любую другую, то такой выбор приведет к большому риску. Причина в том, что диагональные ограничения, связанные с выбором такой позиции, могут закрыть единственную свободную позицию в рисковой строке, и тем самым, привести решение в тупик. Чтобы исключить риск возникновения подобной ситуации, мы на каждом шаге находим строку с минимальным числом свободных позиций, и именно там выбираем позицию для расположения ферзя. Если, в списке свободных строк оказываются две или три строки с одинаковым минимальным значением числа позиций в строке, то мы случайно выбираем одну из этих двух (трех) строк.

Правило минимального ущерба. После того как, согласно правилу минимального риска, мы выбрали индекс строки, нам предстоит выбрать позицию в этой строке для расположения ферзя. Если в строке осталась только одна позиция, то мы выбираем ее. Если в строке остались две или больше позиций, то мы выбираем ту позицию, которая из-за диагональных ограничений наносит минимальный ущерб всем свободным позициям в оставшихся свободных строках. Если окажется, что две позиции в выбранной строке наносят оставшимся позициям одинаковый минимальный ущерб, то случайно выбирается индекс одной из этих двух позиций.

***4.Зачем в программе ведется учет общего числа случаев применения процедуры BT?***

Ответ. Учет числа случаев применения процедуры **BT** (числа случаев, когда ветвь поиска приводит в тупик), имеет важное значение для работы алгоритма. В процессе решения задачи учитываются все случаи, когда прерывается формирование ветви поиска решения, и совершается возврат назад на один из базовых уровней. Когда суммарное значение (**totSimCount**) превышает заданное пороговое значение (**totSimBound**), то вычисления прерываются и процедура комплектации повторяется снова. Максимальное количество таких повторов равно десяти. Если в результате этих действий не удается комплектовать композицию, то принимается решение, что рассматриваемая композиция является отрицательной. Именно **totSimCount** является тем критическим показателем, на основе которого принимается решение прервать вычисления и повторить их заново.

Полезное свойство счетчика **totSimCount** состоит еще и в том, что он позволяет определить, насколько эффективно работает алгоритм. Если, в процессе комплектации композиции, ни разу не возникает тупиковая ситуация, то это означает, что алгоритм работает правильно. Это те уникальные решения, о которых можно говорить, что алгоритм «детерменированно решает не детерменированную задачу». Поэтому, очевидно, что при прочих равных условиях, чем меньше число случаев применения процедуры **BT**, тем более эффективным является работа данного алгоритма.

***5.Если композиция оказывается отрицательной, то программа выводит сообщение: «Данная композиция не может быть комплектована. Вероятность ошибки такого решения меньше 0.0001». Как было получено это значение?***

Ответ. Вначале важно отметить, что такое сообщение для отрицательных композиций появляется только в том случае, если размер шахматной доски **n < 100**. Фактически в этом интервале ошибка принятия такого решения меньше 0,0001. Если 1**00 <= n <800**, то ошибка решения о том, что рассматриваемая композиция является отрицательной, меньше **0,00001**, а для значений **n > = 800** соответствующая ошибка решения меньше **0,000001**.

Алгоритм построен таким образом, что здесь невозможно появление **False Positive** решений, т.е. если решение получено, то оно не может быть неверным, так как контроль правильности выбранной позиции проверяется на каждом шаге. Однако алгоритм не исключает возможности появления **False Negative** решений, т.е. композиций, которые являются положительными, но программа, после долгих попыток принимает ошибочное решение и считает, что композиция является отрицательной.

*Какова вероятность получения False Negative решений в работе данной программы?*

Для того, чтобы ответить на этот вопрос, был выполнен достаточно объемный вычислительный эксперимент.

1.Рассмотрим базовый список различных размеров шахматной доски для проведения вычислительных экспериментов.

2.Для каждого значения **n** из этого списка сформируем и сохраним большие выборки полных решений задачи **n-Queens Problem**.

3.На основе каждой выборки решений **n-Queens Problem** сформируем и сохраним большую выборку случайных композиций. Для *n = (20, 30, ..., 90, 100, 200, 300, 500, 800, 1000, 3000, 5000)* размер выборки композиций был равен *100 000*, и далее размер выборки уменьшался с увеличением **n**.

Очевидно, что каждая такая композиция может быть комплектована до полного решения хотя бы одним способом, так как каждая композиция является частью какого-то полного решения.

4.Рассмотрим каждую выборку композиций, и для каждой композиции запустим программу комплектации для получения полного решения. В процессе анализа каждой выборки, определим число случаев, когда программа формирует **False Negative** решения.

5.В результате анализа всех данных было установлено:

a) Программа успешно решает задачу комплектации почти всех композиций.

b) В диапазоне значений **7 <= n < 100** программе не удалось завершить некоторые композиции и доля таких **False Negative** решений в рассмотренных выборках была меньше 0,0001.

c) В диапазоне значений **100 <= n < 800** доля **False Negative** решений в рассмотренных выборках была меньше **0,00001**.

d) В диапазоне значений **n > = 800** все композиции были завершены, и не было ни одного **False Negative** решения. Однако очевидно, что это не исключает возможности появления **False Negative** решений при многократном увеличении объема выборки. В любом случае в диапазоне значений *n = (800, 1000, 3000, 5000, 10000)* значение ошибки принятия решения о том, что рассматриваемая композиция является отрицательной, будет меньше **0,000001** и с увеличением значения **n** эта ошибка уменьшится.

В процессе компьютерного моделирования учитывались различные граничные значения общего числа применений процедуры **BT**. Самым оптимальным значением было **1000**. Причем (это важно), если после тысячного применения процедуры БТ не удается комплектовать рассматриваемую композицию, то делается вторая попытка завершить композицию (с начала). Количество таких попыток равно десяти. Увеличение этого значения увеличит время, по истечении которого появится сообщение о том, что композиция является отрицательной, а доля **False Negative** решений немного уменьшится. Если мы уменьшим это значение, то соответственно уменьшится время принятия решения и немного увеличится доля **False Negative** решений.

Таким образом, на основе данной серии вычислительных экспериментов было установлено следующее правило:

«если при анализе произвольной композиции процедура **BT** повторяется **1000** раз и не удается завершить комплектацию композиция, то процедура решения повторяется снова с самого начала. Общее количество таких повторных вычислений равно 10. Если в результате всех этих действий не удается комплектовать композицию, то такая композиция считается отрицательным. Вероятность ошибки при отнесении положительной композиции к группе отрицательных композиций, т.е. вероятность **False Negative** решения зависит от размера шахматной доски **n**:

**7 < n < = 100**, то значение ошибки решения меньше **0,0001**,

**100 < = n < 800**, значение ошибки принятия решения меньше **0,00001**,

**n > = 800**, значение ошибки решения меньше **0,000001**

Очевидно, что эта ошибка относится только к положительным композициям. Любую отрицательную композицию, программа определяет с гарантией **100%**.

Выбрав значения 1000 в качестве максимально допустимого количества применения процедуры BT и числа 10 в качестве максимального числа повторных вычислений - это результат компромисса между скоростью работы программы и значениями ошибки принятия решения. Если есть необходимость снизить вероятность ошибки формирования **False Negative** решений в диапазоне значений 7 <n <= 800, то вместо указанных значений можно рассматривать гораздо большие числа. Тогда при обработке большой выборки произвольных композиций увеличится время обработки только негативных композиций, а также только тех позитивных композиций, которые программа ошибочно отнесет к негативным. Скорость обработки всех остальных композиций останется прежней.

Важно отметить, что в предыдущей версии программы, на основе которой проводилось исследование, и результаты были опубликованы на arhiv.org, количество повторных решений было 3. Здесь количество повторных решений равно 10. Граничное значение числа использования процедуры BT осталось прежним и равно 1000.

***6. Как в программе определяется индекс события?***

Ответ. Индекс события зависит от ряда показателей:

- размера шахматной доски (**n**),

- размера композиции (**k**),

- от результата сравнения этих величин с:

а) фиксированными значениями **nFix1=50**, и **nFix2=100**,

b) и граничными значениями **eventBound1** и **eventBound2**, которые вычисляются на основе уравнения регрессии для заданного значения **n**.

Для наглядности представления, выделим на числовой оси натурального ряда три участка

**1**  **2** **3**

- -! - - - - - - - - - - - - !- - - - - - - - - - - - - - !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

! ! !

**7** **50** **100**

1. **7 < = n < 50**, eventInd = **4**

Здесь расчеты начинаются с **4**-го подготовительного блока и далее выполняются в блоке **5**.

1. **50 < = n < 100**, eventInd = **2**

Здесь расчеты начинаются со **2**-го подготовительного блока и далее выполняются в блоках **3, 4, 5**

1. **n > = 100**,

Здесь выбор значения индекса события зависит от соотношения размера композиции (**k**) и граничных значений блоков событий **eventBound1** и e**ventBound2**:

1. если **k < eventBound1**, то **eventInd = 1**;
2. если **eventBound1 < = k < eventBound2**, то **eventInd = 2**;
3. если **k > eventBound2**, то **eventInd = 4**.

Можно заметить, что индекс события не принимает значения **3** или **5**. Это связано с тем, что для перехода в блок **3** необходимо предварительно в блоке **2** выполнить небольшое число подготовительных операций, и после этого, совершить переход в блок **3**. Аналогичная ситуация со связкой блоков **4** и **5**. Здесь, как и в предыдущем случае, перед переходом в блок **5** необходимо выполнить подготовительные операции в блоке **4**.

***7. Можно ли использовать данный алгоритм для значений n > 100 000 000***

Ответ. В процессе исследования алгоритм разрабатывался для широкого диапазона значений размера шахматной доски (**n**): от **7** до **100** миллионов. На самом деле, указанная здесь верхняя граница интервала принципиального значения не имеет. Она была выбрана только исходя из удобства моделирования, учитывая временные затраты, связанные с формированием и анализом выборок для больших значений **n**. У каждого пользователя, значение верхней границы может ограничиваться только размером оперативной памяти компьютера. На компьютере с размером оперативной памяти **32 GB,** расчеты проводились для размера шахматной доски **800** миллионов и более.

О нижней границе интервала. Хотя, в качестве нижней границы интервала указано значение ***7***, любители «нажать красную кнопку» могут провести расчеты для значений **n=(5,6)**, программа допускает такие значения.

**8. Как провести оценку эффективности полученного алгоритма?**

Ответ. Приведем несколько важных показателей, которые используются для оценки эффективности любого алгоритма:

1. Алгоритм должен работать быстро, т.е. время решения задачи должно быть минимальным.
2. Важно, чтобы алгоритм был линейным по времени ***O(n)*** (или близко к этому).
3. Алгоритм должен надежно работать в широком диапазоне изменения основных параметров решаемой задачи.
4. Алгоритм должен быть сконструирован таким образом. чтобы «свободно» работать в пределах разумного объема оперативной памяти (например, **32 GB**), т.е не должно быть дополнительных операций связанных с нехваткой оперативной памяти.
5. Все, что нужно для решения задачи, алгоритм «должен находить сам», не требуя дополнительных действий со стороны пользователя.

Это очевидные для каждого программиста характеристики, которые можно дополнить чем-то еще. Однако, это внешние атрибуты работы программы. Для нас важно определить тот внутренний критерий работы алгоритма, который делает его эффективным. В недетерминированных задачах, одним из важных критериев является число случаев применения процедуры ***Back Tracking (BT)*** в процессе решения задачи. Чем меньше число случаев применения процедуры ***BT***, тем лучше. Будет идеально, если в процессе решения процедура ***BT*** вообще ни разу не будет использоваться. Это будет означать, что последовательно выбирая некоторую свободную строку и некоторую свободную позицию в этой строке, мы доходим до цели, ни разу не допуская критической ошибки – все выполненные шаги оказываются верными. В этом смысле, предложенный алгоритм является достаточно эффективным. Например, для **n=1000**, в эксперименте, где проводилась комплектация одного миллиона композиций, в **60.5 0**% случаях процедура ***BT*** ни разу не использовалась. В оставшихся решениях - в **22.72**% случаях процедура ***BT*** использовалась только один раз, в **9.21**% случаях процедура ***BT*** использовалась два раза, и в **3.95**% случаях процедура ***BT*** использовалась три раза. Все вместе это составляет **96,38**%. В оставшейся части, что составляет **3.62**%, использовалось различное количество случаев применения процедуры ***BT*** . Описанная эффективность работы алгоритма не связана только со значением **n=1000**. Похожая закономерность, в целом характерна для широкого диапазона значений **n**. По этой причине скорость предложенного алгоритма довольно высока. Например, для рассматриваемого значения **n = 1000** среднее время комплектации одной произвольной композиции составляет **0,035** секунды. И, как упоминалось выше, на компьютере с **32 ГБ** оперативной памяти алгоритм позволяет комплектовать произвольную композицию на шахматной доске размером **800** миллионов и более. Поэтому в целом по вышеперечисленным критериям алгоритм можно считать достаточно эффективным.