

Тема 3. ИСЧЕРПЫВАЮЩИЙ ПОИСК

3.1. Поиск с возвратом

3.1.2. Применение общего алгоритма

Общий алгоритм поиска с возвратом представляет собой только общую схему решения, с которой следует подходить к конкретной задаче. Его непосредственное применение обычно приводит к алгоритмам с недопустимо большим временем работы. Поэтому общий алгоритм должен быть хорошо приспособлен к решению конкретной задачи, чтобы он был пригоден для практического использования. Часто это требует большой изобретательности при поиске способов сокращения перебора.

Для применения общего алгоритма поиска с возвратом к конкретной задаче необходимо выполнить анализ задачи с целью определения различных ограничений и специальных методов и приемов повышения эффективности процесса поиска за счет сокращения перебора. Прежде всего, необходимо решить вопрос о представлении вектора решения, для каждого элемента a_k вектора определить все его возможные значения (т. е. множества A_k), детально проанализировать задачу для выявления свойственных ей ограничений, которые являются основой для вычисления подмножеств S_k кандидатов для расширения частичных решений. Таким образом, в первую очередь необходимо определиться со структурами данных для представления множеств A_k , S_k , векторов решений и их элементов a_k , а также с множеством производимых над ними операций, позволяющих добиться по возможности наибольшей эффективности вычислений. Процесс приспособления общего алгоритма завершается разработкой окончательного алгоритма решения задачи, учитывающего все ограничения и усовершенствования, направленные на сокращение перебора.

Проиллюстрируем это на примере решения одной из простейших комбинаторных задач – задачи о ферзях, которую можно сформулировать следующим образом: найти все варианты расстановки максимального числа ферзей на шахматной доске размера $n \times n$ так, чтобы ни один ферзь не атаковал другого. Поскольку ферзь атакует все поля в своей строке, своем столбце и диагоналях, очевидно, что на шахматной доске можно расставить максимум n не атакующих друг друга ферзей. Таким образом, поставленная задача сводится к определению возможности расстановки n не атакующих друг друга ферзей и, если расстановка возможна, к определению количества таких расстановок. Если же расстановка n ферзей невозможна, то решается задача расстановки $(n - 1)$ ферзей и т. д.

Сначала решим вопрос о представлении вектора решений. Очевидно, что все решения имеют одну и ту же фиксированную длину n , т. е. решение можно представить вектором (a_1, \dots, a_n) . На первый взгляд, элемент a_i ($1 \leq i \leq n$) этого вектора должен представлять собой координату позиции, в которой размещается i -й ферзь, т. е. упорядоченную пару чисел, определяющих соответственно номер строки и номер столбца. Однако поскольку в каждом столбце может находиться только один ферзь, то решение можно представить более простым вектором (a_1, \dots, a_n) , в котором элемент a_i ($1 \leq i \leq n$) есть номер строки ферзя, расположенного в столбце с номером i , т. е. координатой позиции является пара (a_i, i) . Очевидно, что множества значений элементов a_i совпадают, т. е. $A_1 = \dots = A_n = \{1, \dots, n\}$.

Рассмотрим свойственные задаче ограничения. Одно ограничение, связанное с тем, что в столбце может находиться только один ферзь, учтено представлением вектора решения. Другое ограничение заключается в том, что в каждой строке может быть только один ферзь, поэтому если $i \neq j$, то $a_i \neq a_j$. Наконец, поскольку ферзи могут атаковать друг друга по диагонали, мы должны иметь $|a_i - a_j| \neq |i - j|$, если $i \neq j$. Таким образом, для того чтобы определить, можно ли добавить a_k для расширения частичного решения $(a_1, a_2, \dots, a_{k-1})$ до $(a_1, a_2, \dots, a_{k-1}, a_k)$, достаточно сравнить элемент a_k с каждым a_i , $i < k$. Эту проверку можно реализовать в виде функции *QUEEN*, представленной алгоритмом 3.2, которая отвечает на вопрос, включить данную позицию в подмножество S_k кандидатов на выбор a_k или нет.

```

function QUEEN ( $a_k, k$ ) // тип Boolean
// true, если можно поставить ферзь в строку  $a_k$  столбца  $k$ 
   $flag \leftarrow \mathbf{true}$ 
   $i \leftarrow 1$ 

  while  $i < k$  and  $flag$  do {
    if  $a_i = a_k$  or  $|a_i - a_k| = |i - k|$ 
    then  $flag \leftarrow \mathbf{false}$ 
     $i \leftarrow i + 1$ 
  }

  QUEEN  $\leftarrow flag$ 
return

```

Алгоритм 3.2. Функция *QUEEN*

Поскольку областью значений каждого элемента a_i вектора решения является множество целых чисел от 1 до n , нет необходимости в вычислении и явном хранении подмножеств S_k . Проще хранить наименьшее значение из S_k и следующее значение вычислять по мере необходимости. Текущее значение элемента множества S_k обозначим через s_k . Тогда проверке условия $S_k \neq \emptyset$ будет соответствовать условие $s_k \leq n$. В результате процедуру нахождения всех решений задачи о неатакующих друг друга ферзях на доске размера $n \times n$ можно формально представить алгоритмом 3.3.

```

 $s_1 \leftarrow 1$ 
 $k \leftarrow 1$ 
while  $k > 0$  do
  while  $s_k \leq n$  do
     $a_k \leftarrow s_k$ 
     $s_k \leftarrow s_k + 1$ 
    while  $s_k \leq n$  and not  $QUEEN(s_k, k)$ 
      do  $s_k \leftarrow s_k + 1$ 
    if  $k = n$ 
      then {записать  $(a_1, a_2, \dots, a_k)$ 
            как решение}
     $k \leftarrow k + 1$ 
     $s_k \leftarrow 1$ 
    while  $s_k \leq n$  and not  $QUEEN(s_k, k)$ 
      do  $s_k \leftarrow s_k + 1$ 
   $k \leftarrow k - 1$ 

```

Алгоритм 3.3. Решение задачи о ферзях методом поиска с возвратом

Следует отметить, что данный алгоритм корректно обрабатывает и ситуацию, когда $k = n + 1$, поскольку вычисляемое значение s_{n+1} не меньше, чем $n + 1$, и, следовательно, множество S_{n+1} всегда пусто. Включение во внутренний цикл специальной проверки для предотвращения ситуации, когда значение k становится больше n , будет слишком дорогостоящим с точки зрения времени работы алгоритма.

```

определить  $S_1 \subseteq A_1$ 
count  $\leftarrow 0$ 
 $k \leftarrow 1$ 
while  $k > 0$  do
  while  $S_k \neq \emptyset$  do
    // продвижение
     $a_k \leftarrow$  элемент из  $S_k$ 
     $S_k \leftarrow S_k - \{a_k\}$ 
    count  $\leftarrow$  count + 1
    if  $(a_1, a_2, \dots, a_k)$  – решение
      then записать его
     $k \leftarrow k + 1$ 
    определить  $S_k \subseteq A_k$ 
   $k \leftarrow k - 1$  // возвращение
// все решения найдены

```