Тема 3. ИСЧЕРПЫВАЮЩИЙ ПОИСК

Класс комбинаторных задач характеризуется тем, что в них рассматриваются задачи на существование, эффективное построение, перечисление и оптимизацию объектов, образованных из сравнительно большого числа элементов. При решении подобных задач приходится иметь дело с алгоритмами, ищущими решение методом проб и ошибок. Комбинаторные задачи обычно требуют исчерпывающего поиска множества всех возможных решений или наилучшего решения, а алгоритмы решения имеют экспоненциальную временную сложность. Общие схемы решения должны быть хорошо приспособлены к конкретной задаче так, чтобы за счет сокращения перебора полученный алгоритм стал пригодным для практического использования.

Одним из широко используемых общих методов исчерпывающего поиска является *поиск с возвратом* (backtrack), а также такие его разновидности, как *метод ветвей и границ* для поиска оптимальных решений, *метод альфа-бета отсечений* для игровых задач. Для теоретико-числовых задач могут быть полезны *методы решета*.

3.1. Поиск с возвратом

3.1.1. Общий алгоритм поиска с возвратом

В общем случае предполагается, что решение задачи представляет собой вектор $(a_1, a_2, \ldots)$ конечной, но не определенной длины, удовлетворяющий некоторым ограничениям. Каждый элемент a_i является элементом конечного линейно упорядоченного множества A_i. Таким образом, при исчерпывающем поиске должны рассматриваться элементы множества $A_1 \times A_2 \times \ldots \times A_i$ для $i = 0, 1, 2, \ldots$ в качестве возможных решений. В качестве исходного частичного решения выбирается пустой вектор ( ) и на основе имеющихся ограничений определяется, какие элементы из множества A_1 являются кандидатами в a_1; подмножество таких кандидатов обозначим через S_1. В качестве a_1 выбирается наименьший элемент множества S_1; в результате получается частичное решение (a_1). В общем случае различные ограничения, описывающие решения, определяют, из какого подмножества S_k множества A_k должны выбираться кандидаты для расширения частичного решения от $(a_1, a_2, \ldots, a_{k-1})$ до $(a_1, a_2, \ldots, a_{k-1}, a_k)$. Если частичное решение $(a_1, a_2, \ldots, a_{k-1})$ не предоставляет возможностей для выбора элемента a_k, т. е. $S_k = \varnothing$, то необходимо вернуться и выбрать новый элемент a_{k-1}. Если новый элемент a_{k-1} выбрать невозможно, придется вернуться еще дальше и выбрать новый элемент a_{k-2} и т. д.

Процесс поиска с возвратом удобно представить в виде дерева поиска, в котором исследуемое подмножество множества $A_1 \times A_2 \times \ldots \times A_i$ для $i = 0, 1, 2, \ldots$ представляется следующим образом. Корню дерева (нулевой уровень) ставится в соответствие пустой вектор. Его сыновья образуют множество S_1 кандидатов для выбора a_1. В общем случае вершины k-го уровня образуют множества S_k кандидатов на выбор a_k при условии, что $a_1, a_2, \ldots, a_{k-1}$ выбраны так, как указывают предки этих вершин. Пример дерева поиска представлен на рис. 3.1, где пунктирные линии показывают порядок прохождения вершин в процессе поиска с возвратом. Вопрос о том, имеет ли задача решение $(a_1, a_2, \ldots)$, равносилен вопросу, являются ли какие-нибудь вершины дерева решениями. Для поиска всех решений необходимо получить все такие вершины.

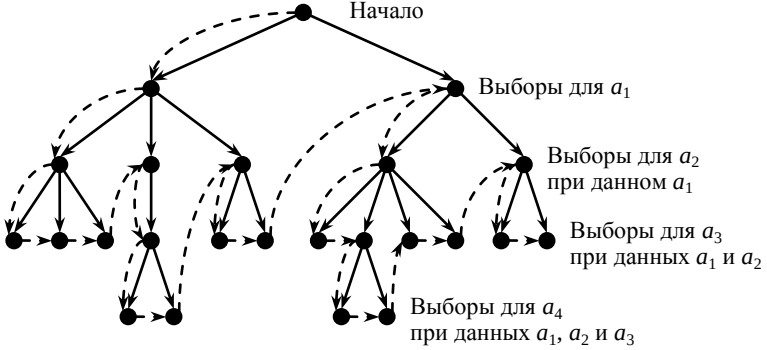

Рис. 3.1. Дерево поиска

Рассмотренный общий метод поиска с возвратом для нахождения всех решений формально описывается алгоритмом 3.1. Внутренний цикл осуществляет расширение частичного решения (продвижение на следующий уровень в дереве поиска), если оно возможно (т. е. при $S_k \neq \varnothing$). Внешний цикл реализует возврат к предыдущему частичному решению (возврат на предыдущий уровень в дереве поиска) в случае невозможности дальнейшего продвижения (при $S_k = \varnothing$). Переменная *count* в алгоритме не имеет принципиального значения для поиска, она носит информативный характер и служит для подсчета числа исследованных вершин в дереве в процессе поиска.

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

Алгоритм 3.1. Общий алгоритм поиска с возвратом

Если требуется найти только одно решение, алгоритм можно легко модифицировать так, чтобы он завершал работу после записи первого найденного решения; в этом случае останов в конце внешнего цикла **while** означает, что решений нет.