Тема 3. ИСЧЕРПЫВАЮЩИЙ ПОИСК

3.1. Поиск с возвратом

3.1.5. Способы программирования поиска с возвратом

Первый способ – это прямое программирование общего алгоритма 3.1 поиска с возвратом (приспособленного к конкретной задаче) в виде двух вложенных циклов: внутреннего цикла для расширения частичного решения и внешнего – для возвращения (итерационный подход).

Второй способ основан на рекурсивном подходе. Рекурсивный вариант поиска с возвратом представлен алгоритмом 3.5, где символ || означает конкатенацию векторов:

$$(a_1, \ldots, a_n) \,||\, (b_1, \ldots, b_m) = (a_1, \ldots, a_n, b_1, \ldots, b_m),\ (\,) \,||\, (a) = (a),$$

где ( ) – пустой вектор. Очевидно, что при первом обращении к процедуре параметр *vector* представляет собой пустой вектор ( ), а параметр $i = 1$, т. е. первое обращение есть *BACKTRACK*(( ), 1).

```
procedure BACKTRACK (vector, i)
   if vector является решением then записать его
   определить S_i
   for a ∈ S_i do BACKTRACK (vector || (a), i + 1)
return
```

Алгоритм 3.5. Рекурсивный вариант поиска с возвратом

В рекурсивном алгоритме все возвращения скрыты в механизме, реализующем рекурсию. Рекурсивный подход, как известно, требует дополнительных расходов памяти и времени по сравнению с итерационным подходом.

Третий способ использует язык ассемблера со средствами макрорасширения и основывается на предположении, что наиболее важным аспектом программы является ее быстродействие. Идея состоит в применении средств макрорасширения ассемблера для создания высокоспециализированных программ, в которых все или некоторые циклы не являются вложенными (т. е. циклы преобразуются в последовательные линейные конструкции). Такой подход устраняет определенные логические проверки и уменьшает число операций для контроля циклов. Если все решения имеют длину n, то это можно сделать, например, написав макрокоманду (назовем ее CODE_i) со следующим телом:

определить S_i
L_i: **if** $S_i = \varnothing$ **then goto** L_{i-1}
$a_i :=$ элемент из S_i
$S_i := S_i - \{a_i\}$

Макрокоманда CODE_i повторяется для i = 1, 2, ..., n, порождая программу вида

CODE_1
CODE_2
⋮
CODE_n
записать $(a_1, a_2, \ldots, a_n)$ как решение
goto L_n
L_0: // все решения найдены

Применительно к задаче о ферзях макрокоманда CODE_i размещает на доске i-й ферзь.

Такой подход требует, чтобы все решения имели одну и ту же фиксированную длину. Это часто встречающийся случай, и преимущество описанного подхода очевидно: макрокоманду CODE_i можно организовать так, что шаги будут приспособлены для S_i. Например, в задаче о ферзях нужно ограничение $2 \le a_1 \le \lceil n/2 \rceil$, и если n нечетно и $a_1 = \lceil n/2 \rceil$, то $1 \le a_2 \le \lceil n/2 \rceil - 2$. Включение этих проверок в общий алгоритм, написанный с циклами **while**, неэффективно, так как некоторые проверки нужно осуществлять каждый раз во внутреннем цикле, даже несмотря на то, что они редко что-либо дают. Используя макроподход, команды для проверок можно включить в программу только там, где это необходимо.

Существуют модификации макроподхода, не требующие, чтобы все решения имели одну и ту же длину. Кроме того, этот подход можно использовать и без макросредств, что позволяет использовать его при программировании на языках без макросредств (в том числе и на языках высокого уровня). Несмотря на некоторые неудобства, он заметно повышает быстродействие программы.