Алгоритмы с возвращением,

их реализация с помощью рекурсий и динамических структур

Во многих задачах из разных областей ставятся вопросы типа: «Сколько существует способов…», «Подсчитайте количество элементов удовлетворяющих условию …», «Перечислите все возможные варианты», «Есть ли способ…» и т.д.

Для того чтобы ответить на них обычно необходимо провести поиск в некотором множестве всех возможных вариантов, среди которых находятся решения данной задачи. Существуют два общих метода организации исчерпывающего поиска: **перебор с возвратом** и его логическое дополнение — **метод решета**.

- Решение задачи методом перебора с возвратом строится последовательным расширением частичного решения. Если на каком-то шаге такое расширение провести не удается, то происходит возврат к более короткому частичному решению, и попытки его расширить продолжаются. Для ускорения перебора с возвратом вычисления всегда стараются организовать так, чтобы была возможность отметать как можно раньше и как можно больше заведомо неподходящих вариантов.

- При использовании метода решета из множества всевозможных вариантов исключаются все элементы, не являющиеся решениями.

Рассмотрим метод перебора с возвратом. Соединение его с рекурсией определяет специфический способ реализации рекурсивных вычислений, называемый **возвратной рекурсией**.

С возвратной рекурсией мы с вами сталкивались, когда строили алгоритм генерирования всех разбиений множества.

Алгоритм поиска с возвращением

Рассмотрим общий случай, когда решение задачи имеет вид вектора (a_1, a_2, …), длина которого (в общем случае) не определена, но ограничена сверху некоторым числом r,

а каждое a_i является элементом некоторого конечного линейно упорядоченного множества A_i.

Таким образом, при исчерпывающем поиске в качестве возможных решений мы рассматриваем элементы множества $A_1 \times A_2 \times \ldots \times A_i$ для любого $i \leq r$,

и среди них выбираем те, которые удовлетворяют ограничениям, определяющим решение задачи.

В качестве начального частичного решения берется пустой вектор () и на основе имеющихся ограничений выясняется, какие элементы из A_1 являются кандидатами для их рассмотрения в качестве a_1

(множество таких элементов обозначим S_1).

В качестве a_1 выбирается наименьший элемент множества S_1, что приводит к частичному решению (a_1).

В общем случае ограничения, описывающие решения, говорят о том, из какого подмножества S_k множества A_k выбираются кандидаты для расширения частичного решения от (a_1, a_2,…, a_{k-1}) до (a_1, a_2,…, a_k).

Если частичное решение (a_1, a_2,…, a_{k-1}) не предоставляет других возможностей для выбора нового a_k

(т.е. у частичного решения (a_1, a_2,…, a_{k-1}) нет кандидатов для расширения),

то происходит возврат и осуществляется выбор нового элемента a_{k-1} из S_{k-1}.

Если новый элемент a_{k-1} выбрать нельзя,

т.е. к данному моменту множество S_{k-1} уже пусто, то происходит еще один возврат и делается попытка выбрать новый элемент a_{k-2} и т.д.

Общую схему алгоритма, осуществляющего поиск с возвращением для нахождения всех решений, можно представить в следующем виде:

```
k:=1;
Вычислить S1 {Например, в качестве S1 взять A1};
While k>0 do
  Begin
    While не пусто Sk do
      Begin
        В качестве ak взять наименьший элемент из
Sk,
        удалив его из Sk
        If (a1, a2,…, ak) является решением
          Then Вывести это решение
        If k<r then
          Begin
            k:=k+1;
            Вычислить Sk
          End;
      End;
  k:=k-1;  {Возврат}
End;
```

Более коротко общую процедуру поиска с возвращением можно записать в рекурсивной форме:

```
Procedure ПОИСК (X: вектор; i:integer);
Begin
   If X является решением Then вывести его;
   If i<=r Then Вычислить Si
   For all afrom Si do ПОИСК(XX, i+1)
{XX получается из X добавлением элемента a}
End;
```

Вызов ПОИСК((), 1) находит все решения, причем все возвраты скрыты в механизме, регулирующем рекурсию.

Задача о расстановке ферзей

Для иллюстрации того, как описанный метод применяется при решении конкретной задачи, рассмотрим задачу нахождения количества таких расстановок восьми ферзей на шахматной доске, в которых ни один ферзь не атакует другого.

(*Рассмотрим эту задачу для шахматной доски произвольных размеров*).

Очевидно, что в каждой горизонтали (строке) может стоять только один ферзь (иначе они бы били друг друга).

Поэтому мы последовательно будем ставить по одному ферзю сначала в первую строку, затем во вторую, и т.д. и таким образом формировать вектор решений.

Решение расстановки ферзей можно искать в виде вектора $(a_1, a_2, \ldots, a_8)$,

где a_i — номер вертикали (столбца), на которой стоит ферзь, находящийся в i-й горизонтали (строке),

т.е. $A_1=A_2=A_3=\ldots=A_7=A_8=\{1,2,3,4,5,6,7,8\}$.

Каждое частичное решение — это расстановка N ферзей (где $1 \le N \le 8$) в первых N горизонталях таким образом, чтобы эти ферзи не атаковали друг друга.

Для первой строки множество возможных вариантов S_1 совпадает со множеством всех вариантов A_1.

Но уже после установки первого ферзя, оно будет существенно отличаться от исходного (мы должны исключить из множества S_i все клетки, которые находятся «под боем» уже поставленных i-1 ферзей).

Свободные клетки в матрице **a** будут равны 0,

клетки «под боем» уже поставленных ферзей равны 1,

а клетки, где стоят ферзи 2

Место, куда вставляем очередного ферзя, определяется в процедуре **Set_F**.

В нее передается матрица **a**, описывающая положение шахматной доски на данном шаге

и номер строки **x**, в которую вставляется очередной ферзь.

Если все ферзи расставлены, то очередное решение выводится на экран и счетчик решений **k** увеличивается на 1.

В противном случае мы

- находим первую незанятую клетку в строке **x**,
- копируем матрицу **a** в **b** (чтобы не портить ее),
- и вызываем процедуру **Fill_F**, которая ставит ферзя на выбранное место и помечает все клетки, которые оказываются у него «под боем»,
- а затем вызываем процедуру **Set_F**, уже для следующей строки **x**+1 и измененной матрицы **b**.

```
program ferzi;
TYPE
  mas=array [1..15,1..15] of integer;
VAR
  a:mas;
{матрица, описывающая положение шахматной доски}
  i,j,n:integer;
  k:longint;
```

```
PROCEDURE Fill_F(x,y:integer; var a:mas);
 {x, y — координаты вставки ферзя}
var
  i, j:integer;
begin
  for i:= 1 to n do
    begin
      a[x,i]:=1; {строка, где будет стоять ферзь —«под
боем»}
      a[i,y]:=1;     {столбец, где будет стоять ферзь —«под
боем»}
    end;
```

```
i:=x-1;    {переходим в левую верхнюю клетку по диагонали} j:=y-1;    {от (x,y)}
  while (i<>0) and (j<>0) do
    begin
      a[i,j]:=1;   {помечаем диагональ слева и вверх от (x,y) }
      dec(i);
      dec(j);
    end;
  i:=x+1;     {переходим в правую нижнюю клетку по диагонали от (x,y)}
  j:=y+1;
  while (i<>n+1) and (j<>n+1) do
    begin
      a[i,j]:=1;    {помечаем диагональ справа и вниз от (x,y) }
      inc(i);
      inc(j);
    end;
```

```
 i:=x-1;      {переходим в правую верхнюю клетку}
 j:=y+1;
 while (i<>0) and (j<>n+1) do
   begin
     a[i,j]:=1;    {помечаем диагональ справа и вверх от (x,y)}
     dec(i);
     inc(j);
   end;
 i:=x+1;   {переходим в левую нижнюю клетку от (x,y)}
 j:=y-1;
 while (i<>n+1) and (j<>0) do
   begin
     a[i,j]:=1;    {помечаем диагональ слева и вниз от (x,y)}
     inc(i);
     dec(j);
   end;
 a[x,y]:=2;    {ставим ферзя на место (x,y)}
end;
```

```
PROCEDURE Set_F(x:integer; a:mas);
{x — строка, куда добавляем ферзя}
var
  i,j:integer;
  b:mas;
begin
  if x=n+1 then     {если все ферзи расставлены}
    begin
    for i:=1 to n do       {выводим матрицу расстановки}
      begin
        for j:=1 to n do
          write(a[i,j]);
          writeln;
      end;
      writeln;
    inc(k)      {наращиваем счетчик вариантов расстановки}
  end
```

```
else        {в противном случае}
   for i:= 1 to n do      {ищем в строке}
     if a[x,i]=0 then     {первую свободную клетку}
       begin
         b:=a;       {копируем матрицу a в матрицу b}
         Fill_F(x,i,b);  {устанавливаем ферзя в i-й столбец
строки x}
         Set_F(x+1,b);     {вызываем процедуру вставки ферзя
в следующую x+1-ю строку измененной матрицы b}
       end;
end;
```

```
BEGIN
  readln(n);     {вводим размерность доски}
  k:=0;      {количество вариантов расстановок равно 0}
  for i:= 1 to n do
    for J:= 1 to n do
      a[i,j]:=0;       {все клетки матрицы свободны}
  Set_F(1,a);        {вызываем рекурсивную процедуру
установки ферзя (сначала устанавливаем первого
ферзя на свободную доску)}
  writeln(k);    {выводим ответ — число вариантов
расстановки}
  readln;
END.
```

Домашнее задание

1. Составить опорный конспект лекции по теме «Алгоритмы с возвращением, их реализация с помощью рекурсий и динамических структур» на основе презентации.

2. Комбинаторика для программистов. Липский В. М.: «Мир», 1988, стр. 102-108.