Живая группа ГО

Условие задачи. В игре ГО, правила которой, конечно, сейчас изучать не будем, есть понятие живой группы. В задаче требуется, имея конкретную позицию и координаты одного камня, установить, является ли этот камень представителем живой группы.

Возможно, вы не знакомы с правилами этой игры. Если так, то ничего страшного. Фактически сама игра для формулировки задачи не нужна, нужно только определение живой группы. Игра ведется на квадратной доске размером 19x19. Это классическая доска ГО. Фигуры игры называются *камнями*. Ход игрока заключается в установке камня на перекрестие линий (правда мы в решении будем устанавливать камень в клетку). Группой камней называется множество, в котором каждый камень имеет хотя бы одного соседа, принадлежащего к этому же множеству. Соседом называется камень, стоящий на соседнем перекрестии по вертикали или горизонтали. Камни, стоящие рядом по диагонали, соседями не являются. *Живой группой* называется такая группа, в которой есть хотя бы один камень, такой что хотя бы одно соседнее с ним перекрестие пусто. Ниже примеры живой и неживой групп.

Живая группа

ч	ч	ч	ч	ч
ч	б	б	б	ч
ч	б		б	ч
ч	б		б	ч
ч	б	б	б	ч
ч	ч	ч	ч	ч

Не живая группа

ч	ч	ч	ч	ч
ч	б	б	б	ч
	ч	ч	б	ч
		ч	б	ч
			ч	

На рисунке камни изображены буквами. Буквой "**б**" — белый камень и буквой "**ч**" — черный камень. Определение живой группы дано, примеры, демонстрирующие определение, приведены, можно приступать к собственно решению.

На примере этой задачи мы еще раз продемонстрируем различие между рекурсивным и нерекурсивным решением. Вспомним, что в "программисткой" науке есть (правда нестрогое) утверждение о том, что каждую задачу можно решить, как рекурсивно, так и не рекурсивно. Вопрос только в осмысленности и эффективности. Поиск рекурсивных решений часто очень сложен, но бывает и наоборот. Например, в уже рассмотренной нами задаче об экономном обходе графа дано нерекурсивное решение и его логика достаточно запутана. Ее рекурсивное решение было бы проще. Но рекурсивное решение той же задачи потребовало бы большего количества ресурсов. Так часто бывает, рекурсия очень требовательная дама.

Задача, к которой мы сейчас приступаем, имеет хорошее рекурсивное и хорошее нерекурсивное решение. Более того, различие между этими двумя типами решений в задаче о живой группе ГО можно свести к минимуму. Минимальное различие заключается в организации памяти. Для рекурсии нужен стек. Следовательно, чтобы организовать рекурсию, необходимо создать стековую память, например, посредством

массива. Вот именно это и будет сделано. Первым шагом мы получим рекурсивное решение, обсудим его и затем получим второе, нерекурсивное решение.

Первый шаг в построении рекурсивного решения — это построение рекуррентного процесса. В нашей задаче рекуррентный процесс налицо. Сущность процесса заключается в переходе от камня к камню и поиске хотя бы одного свободного поля. Если таковое поле найдено, то группа живая, иначе нет. Рекуррентный характер процесса в том, что поиск свободного поля, либо соседа по группе осуществляется для каждого текущего камня, а если найден сосед, то для него поиск повторяется. Общая схема рекурсии может быть такова:

- если рассматриваемое поле пустое, то поиск прекращается и возвращается сообщение об удачном его завершении;

- если рассматриваемое поле занято камнем противника, то поиск прекращается и возвращается сообщение о неудаче рекурсивного поиска;

- если рассматриваемое поле занято своим камнем, то выполняется новый вызов рекурсивной процедуры.

Это самая общая схема, не лишенная проблем. Первая и самая грубая ошибка заключается в том, что текущий вызов рассматривает только одно поле, а мы знаем, что у каждого камня может быть четыре соседа. Технически это означает, что либо должно быть четыре вызова процедуры, либо процедура должна рассматривать четыре ситуации. Как именно построить работу рекурсивной процедуры — вопрос вкуса и личного стиля. Мы выберем второй вариант — в процедуру передаются координаты поля, на котором находится свой камень, и процедура рассматривает четырех его соседей.

Пусть процедура получает на вход две координаты X, Y, тогда ее внешний вид может быть такой:

```
СИТУАЦИЯ = НЕУДАЧА
Рассматривается сосед слева
  Если пустое поле, то СИТУАЦИЯ = УДАЧА
  Если камень противника, то СИТУАЦИЯ = НЕУДАЧА
  Если камень свой, то вызов процедуры с новыми координатами
Если СИТУАЦИЯ = НЕУДАЧА, то рассматривается сосед сверху
  Если пустое поле, то СИТУАЦИЯ = УДАЧА
  Если камень противника, то СИТУАЦИЯ = НЕУДАЧА
  Если камень свой, то вызов процедуры с новыми координатами
```

И точно так же проработаем соседей справа и снизу. Сразу исправим небольшую ошибку (или просто недоговоренность). Ранее везде говорилось о рекурсивной процедуре и по инерции в алгоритме речь идет о процедуре. В то же время говорится о некоторой переменной под именем СИТУАЦИЯ. Очевидно, данная переменная должна сообщать свое значение предыдущим вызовам, а для этого либо она должна быть глобальной величиной, либо рекурсию необходимо оформить в виде функции.

Следующая более тонкая ошибка нам уже встречалась в задаче о закраске контура, имеются в виду решения, основанные на понятии точки — источнике закраски. Вспомните, там мы столкнулись с тем, что одна и та же точка может оказаться соседом нескольких точек, в результате чего она попадет в массив источников многократно, кроме того, соседом текущей точки является и та точка, которая была источником для текущей, а это уже грозит движением по кругу.

Клетки доски ГО также можно рассматривать как точки, а следовательно проблемы предыдущей задачи полностью переходят и на данную. Но это в общем-то плюс, так как описанные проблемы нами уже решены, а их полная идентичность означает, что и решать

мы их можем тем же методом. То есть, чтобы вторично не попасть в клетку занятую своим камнем, мы этот камень после проведенного анализа, независимо от результата, уберем из клетки. Технически это можно осуществить так: пусть доска — это двумерный массив и:

- ноль — клетка пуста;
- единица — клетка занята собственным камнем;
- двойка — клетка занята камнем противника.

Тогда для того чтобы убрать камень из клетки, можно единицу заменить на любое другое число, кроме нуля. В программе это число три. Свою работу рекурсивная функция начнет с указанного камня, каждый вызов породит столько вызовов, сколько у текущего камня окажется соседей. Возвращать функция будет истину, если она получит хотя бы одну истину от порожденных вызовов функций, и ложь, если все порожденные вызовы функций вернут ложь. Таким образом, сам вызов функции вернет ложное значение только в том случае, если все порождаемые вызовы всех уровней вернут значение ложь.

Примечание

После того как было указано на сходство проблем в двух разных задачах, оставшиеся проблемы имеют характер технических вопросов, а не принципиальных, но, конечно, нужно согласиться, что задача о живой группе ГО и задача о закраске контура выглядят слишком по разному, чтобы искать в них аналогию. Однако ничего странного нет. Аналогию мы увидели совсем не в тексте условий, а в процессе поиска нужной точки. Во-первых, грубая аналогия видна сразу. В обоих, задачах ищется точка с определенными свойствами. Более тонкая аналогия в том, как она ищется. И это в обеих задачах делается одинаково. А именно, для каждой точки просматриваются соседи и для каждого соседа проводится анализ его свойств. Различие в задачах следующее. В задаче о закраске нужны все пустые точки, а в задаче о живой группе — хотя бы одна, но для технологии поиска это различие не имеет принципиального значения.

Обратите внимание, что в программе обрабатываются четыре направления, несмотря на то, что одно из направлений — это направление-источник (клетка из которой мы попали в данную), его рассматривать в качестве кандидата даже нет смысла. Но такой подход позволяет не запоминать клетку-источник и избавляет нас от существенных технических сложностей.

Текст программы приведен в листинге.

Листинг

```
program example;
 uses crt,graph;
 var
  dr,md:integer;
  doska:array[1..19,1..19] of byte;
  i,x,y,n,j:integer;
function Control(x,y:integer):boolean;
var
 q:boolean;
function Return(x,y:integer):boolean;
begin
  case doska[x,y] of
   0: Return:=true;
```

```
   1: begin
       doska[x,y]:=3;
       Return:=Control(x,y);
       floodfill(20*x+10,20*y+10,1);
      end;
   2,3: Return:=false;
  end;
end;
begin
 setfillstyle(1,1);
 q:=false;
 if x-1>=1 then q:=Return(x-1,y);
 if not q then
  if y-1>=1 then q:=Return(x,y-1);
 if not q then
  if x+1<=19 then q:=Return(x+1,y);
 if not q then
  if y+1<=19 then q:=Return(x,y+1);
 Control:=q;
end;
begin
{ read(n);
 for i:=1 to 19 do
  for j:=1 to 19 do doska[i,j]:=0;
 for i:=1 to n do
  begin
   read(x,y);
   doska[x,y]:=1;
  end;
 read(n);
 for i:=1 to n do
  begin
   read(x,y);
   doska[x,y]:=2;
  end;
 read(x,y);}
 doska[12,14]:=1; doska[12,13]:=1;doska[13,13]:=1;
 doska[14,13]:=1; doska[15,13]:=1;doska[15,14]:=1;
 doska[15,15]:=1; doska[16,16]:=1;doska[16,17]:=1;
 doska[16,18]:=1; doska[16,19]:=1;doska[15,18]:=1;
 doska[14,18]:=1; doska[13,18]:=1;doska[13,17]:=1;
 doska[13,16]:=1; doska[14,16]:=1;doska[12,16]:=1;
 doska[12,15]:=1;
 doska[14,12]:=2;doska[13,15]:=2;doska[14,15]:=2;
 doska[13,14]:=2;doska[14,14]:=2;doska[11,14]:=2;
 doska[11,13]:=2;doska[11,12]:=2;doska[12,12]:=2;
 doska[13,12]:=2;doska[15,12]:=2;doska[16,12]:=2;
 doska[16,13]:=2;doska[16,14]:=2;doska[16,15]:=2;
 doska[15,16]:=2;doska[11,15]:=2;doska[11,16]:=2;
 doska[12,17]:=2;doska[14,17]:=2;doska[12,18]:=2;
 doska[12,19]:=2;doska[13,19]:=2;doska[14,19]:=2;
 doska[15,19]:=2;doska[15,17]:=2;doska[17,17]:=2;
 doska[17,16]:=2;doska[17,18]:=2;doska[17,19]:=2;
 x:=12;y:=14;
 dr:=detect;initgraph(dr,md,'');
 for i:=1 to 20 do
  begin
   line(i*20,20,i*20,400);
   line(20,i*20,400,i*20);
  end;
 for i:=1 to 19 do
  for j:=1 to 19 do
   if doska[i,j]>0 then
    begin
```

```
     setcolor(doska[i,j]);
     circle(20*i+10,20*j+10,5);
    end;
 setfillstyle(1,1);
 floodfill(20*x+10,20*y+10,1);
 doska[x,y]:=3;
 if Control(x,y) then outtextxy(400,300,'Эта группа живая') else
outtextxy(400,400,'Эта группа не живая');
 readkey;
end.
```

Только что мы рассмотрели рекурсивный вариант решения. Его соль и самый главный пункт — координаты текущего камня, порождающего рекурсивные вызовы для анализа его соседей. В рекурсивном варианте эти координаты хранятся в стеке. Для создания нерекурсивного варианта необходимо продумать организацию стека из координат. Наиболее естественной структурой данных для этих целей, конечно, будет массив. А так как координат две, то речь пойдет о массиве записей из двух компонентов x и y.

Для того чтобы массив работал, как стек, введем специальную переменную величину (назовем ее вершиной стека), всегда равную длине массива. Она (эта переменная) обеспечит нам доступ к верхушке стека. В процессе обхода позиции при обнаружении камня занесение его координат в массив может быть выполнено двумя операциями:

- вершина стека увеличивается на 1;

- координаты камня записываются в элемент массива с индексом равным вершине стека.

Очередной камень для обработки всегда берется с вершины стека. После обработки длина массива (стека) соответственно уменьшается на 1. Анализ позиции прекращается в двух случаях.

- На каком-то шаге обработки было обнаружено пустое соседнее поле. Даже одноразового такого события достаточно для завершения анализа. В таком случае можно сделать вывод, что исследуемая группа живая.

- Длина массива (стека) стала равна нулю. Это означает, что исследованы все камни группы и ни разу не было обнаружено пустое поле. Следовательно, в этом случае вполне можно сделать вывод, что исследуемая группа не есть живая.

Нерекурсивный вариант решения — листинг.

Листинг

```
program example;
 uses crt,graph;
 var
  dr,md:integer;
  doska:array[1..19,1..19] of byte;
  mas:array[1..300] of record
                        x,y:integer;
                       end;
  i,x,y,n,j,len:integer;
  res:boolean;
function Control(x,y:integer):boolean;
var
 i:integer;
```

```
 q:boolean;
begin
 case doska[x,y] of
  0:Control:=true;
  1:begin
     q:=true;
     for i:=1 to len do
       if (mas[i].x=x) and (mas[i].y=y) then q:=false;
     if q then
      begin
       len:=len+1;
       mas[len].x:=x;
       mas[len].y:=y;
       floodfill(20*x+10,20*y+10,1);
      end;
     Control:=false;
    end;
  2,3:Control:=false;
 end;
end;
begin
{ read(n);
 for i:=1 to 19 do
  for j:=1 to 19 do doska[i,j]:=0;
 for i:=1 to n do
  begin
   read(x,y);
   doska[x,y]:=1;
  end;
 read(n);
 for i:=1 to n do
  begin
   read(x,y);
   doska[x,y]:=2;
  end;
 read(x,y);}
 doska[12,14]:=1; doska[12,13]:=1;doska[13,13]:=1;
 doska[14,13]:=1; doska[15,13]:=1;doska[15,14]:=1;
 doska[15,15]:=1; doska[16,16]:=1;doska[16,17]:=1;
 doska[16,18]:=1; doska[16,19]:=1;doska[15,18]:=1;
 doska[14,18]:=1; doska[13,18]:=1;doska[13,17]:=1;
 doska[13,16]:=1; doska[14,16]:=1;doska[12,16]:=1;
 doska[12,15]:=1;
 doska[14,12]:=2;doska[13,15]:=2;doska[14,15]:=2;
 doska[13,14]:=2;doska[14,14]:=2;doska[11,14]:=2;
 doska[11,13]:=2;doska[11,12]:=2;doska[12,12]:=2;
 doska[13,12]:=2;doska[15,12]:=2;doska[16,12]:=2;
 doska[16,13]:=2;doska[16,14]:=2;doska[16,15]:=2;
 doska[15,16]:=2;doska[11,15]:=2;doska[11,16]:=2;
 doska[12,17]:=2;doska[14,17]:=2;doska[12,18]:=2;
 doska[12,19]:=2;doska[13,19]:=2;doska[14,19]:=2;
 doska[15,19]:=2;doska[15,17]:=2;doska[17,17]:=2;
 doska[17,16]:=2;doska[17,18]:=2;doska[17,19]:=2;
 x:=12;y:=14;
 dr:=detect;initgraph(dr,md,'');
 for i:=1 to 20 do
  begin
   line(i*20,20,i*20,400);
   line(20,i*20,400,i*20);
  end;
 for i:=1 to 19 do
  for j:=1 to 19 do
   if doska[i,j]>0 then
    begin
```

```
     setcolor(doska[i,j]);
     circle(20*i+10,20*j+10,5);
    end;
 setfillstyle(1,1);
 floodfill(20*x+10,20*y+10,1);
 mas[1].x:=x;mas[1].y:=y;
 len:=1;res:=false;
 while (len>0) and (not res) do
  begin
   x:=mas[len].x;y:=mas[len].y;
   doska[x,y]:=3;
   len:=len-1;
   if x-1>=1 then res:=Control(x-1,y);
   if not res then
     if y-1>=1 then res:=Control(x,y-1);
   if not res then
     if x+1<=19 then res:=Control(x+1,y);
   if not res then
     if y+1<=19 then res:=Control(x,y+1);
  end;
 if res then outtextxy(400,400,'Эта группа живая') else
outtextxy(400,400,'Эта группа не живая');
 readkey;
end.
```