## Рекурсивное решение в задаче о самой лучшей рубке шашек

### Условие задачи

На стандартном поле 8x8 расставлены черные шашки и одна белая. Необходимо найти самый длинный путь рубки для белой шашки. В решении использовать указатели.

Требование использовать именно динамическую память на первый взгляд можно рассматривать как каприз составителя задачи. Но на более внимательный второй взгляд это не каприз, а хорошая косвенная подсказка. Ясно, что совокупность всех возможных путей рубки представляет собой ориентированный граф, а графы очень хорошо представляются связными списками.

Поэтому сейчас, прежде чем мы начнем обсуждение задачи, немного вспомним технику работы со связными списками. В листингах ниже без подробных объяснений приведены два примера: создание линейного связного списка и создание двоичного дерева.

Листинг. Пример простого связного списка

```
Program example1;
 Uses crt;
 Type
  shablon=^zapis;
  zapis=record
         a:integer;
         next:shablon;
        end;
 Var
  uk1,uk2:shablon;
  i:integer;
begin
 clrscr;
 {Создается указатель на первый элемент списка и запоминается его адрес в
дополнительном указателе}
 new(uk1);uk2:=uk1;
 {Связный список заполняется числами}
 uk1^.a:=1;
 for i:=2 to 10 do
  begin
   new(uk1^.next);
   uk1:=uk1^.next;
   uk1^.a:=i;
  end;
 {Вспоминается адрес первого элемента}
 uk1:=uk2;
 {Связный список проходится еще раз, начиная с первого элемента}
 for i:=1 to 10 do
  begin
   write(uk1^.a,' ');
   uk1:=uk1^.next;
  end;
end.
```

Листинг. Пример построения двоичного дерева с использованием связного списка

```
Program example2;
 Uses crt;
 Type
 shablon=^zapis;
 zapis=record
        a:integer;
```

```
        left:shablon;
        right:shablon;
       end;
 Var
  tree1,tree2:shablon;
  max:integer;
procedure Create_Tree(tree:shablon;n:integer);
var
 new_tree:shablon;
begin
 if n<=max then
  begin
   {Переход на левый узел}
   new(tree^.left);
   new_tree:=tree^.left;
   new_tree^.a:=random(10);
   Create_Tree(New_tree,n+1);
   {Переход на правый узел}
   new(tree^.right);
   new_tree:=tree^.right;
   new_tree^.a:=random(10);
   Create_Tree(New_tree,n+1);
  end;
end;
procedure View_tree(tree:shablon;x,n:integer);
begin
 gotoxy(x,2*n);write(tree^.a);
 if n<=max then
  begin
   {Переход на левый узел}
   View_tree(tree^.left,x-(20 div n),n+1);
   {Переход на правый узел}
   View_tree(tree^.right,x+(20 div n),n+1);
  end;
end;
begin
 randomize;
 clrscr;
 read(max);
 {Создается и заполняется числовым значением корневой элемент дерева}
 new(tree1);tree1^.a:=random(10);
 {Запоминается адрес корневого элемента}
 tree2:=tree1;
 {Создается дерево}
 Create_Tree(tree1,1);
 {Повторный проход дерева}
 View_Tree(Tree2,40,1);
end.
```

После того как мы освежили свои представления о технике работы со связными списками, можно вернуться к исходной задаче. Ранее было сказано, что совокупность путей рубки хорошо представляется ориентированным графом, следовательно, иная более программистская постановка задачи звучит так: дан ориентированный граф, найти его самую длинную ветвь (или все самые длинные ветви, что не намного сложнее).

Конечно, новая формулировка не совсем идентична первоначальной. Новая формулировка исходит из данности графа, а в исходной постановке задачи дана позиция на шашечной доске, а это совсем не одно и то же.

Поэтому мысль разделить задачу на две — построение графа и поиск на графе — будет вполне естественной. Получив позицию, мы преобразуем ее в граф, а затем в соответствии с новой формулировкой подумаем о том, как обойти этот граф в поисках самой длинной ветви.

*Примечание.* Обратите внимание, что мы, как бы не торопимся с поиском решения. Наши рассуждения носят, с первого взгляда подготовительный характер и не имеют прямого отношения к идее решения. О том, как собственно решать задачу, еще не было сказано ни одного слова. А вывод о возможности разбиения задачи на две нетерпеливому человеку может показаться даже лишним. Он может сказать: "Я хочу сразу придумать, как именно искать самый длинный путь". И скорее всего, если этот человек не без способностей, то так оно и будет. А дальше будет одно большое НО. Придумать идею можно, но ее надо еще и реализовать! И нужно сказать, что зачастую возникающие технические проблемы настолько велики, что эффект от идеи сходит на нет. Что же мы фактически сделали, переформулировав задачу? А мы как раз и озаботились хотя бы обозначением этих будущих технических проблем, приблизили постановку к тем структурам данных, которыми мы реально располагаем в языке программирования.

Внимательный читатель мог бы сказать, что ранее мы начинали именно с идеи, выраженной в общепонятных терминах, и зачастую даже не математических. Но здесь нет противоречия. Надо просто видеть разницу в задачах. Если задача заключает в себе очень большую логическую проблему, то конечно, главные усилия должны быть сосредоточены на разработке логики. Рассматриваемая задача — то, что называется технически сложная, поэтому и подход к ней иной. Конечно, что сложно технически, а что логически — это решается каждым разработчиком относительно себя. Что для одного сложно, то для другого легко! Что для одного сложная логика, то для другого технические детали!

Если вы согласны с тем, что было сказано в примечании, то вы должны быть согласны и с тем, что наш следующий шаг — это описание структуры данных, представляющих собой граф. Структуру данных нам определяет условие. Требуется использовать динамическую память, следовательно, использование связных списков — дело решенное и осталось только определить структуру записи связного списка.

Очевидно, что элемент связного списка нужен для описания текущей позиции рубящей шашки, следовательно, необходимо просто посмотреть, что известно про позицию. А известно следующее:

- координаты текущей позиции;

- адреса элементов связного списка, описывающих позиции, в которые рубящая шашка может попасть из текущей.

Достижимых позиций не более трех (позицию, из которой шашка пришла, можно исключить), а координат две. Следовательно, в структуре элемента связного списка должно быть три поля, являющихся указателями на элемент связного списка, и два числовых поля, имеющих смысл координат. После обсуждения структуры данных можно обдумать, как эта структура данных должна быть построена. Если вы невнимательно разобрали пример с построением двоичного дерева, то посмотрите его еще раз. Его отличия от нашей задачи заключаются в следующем:

- количество указателей в нашей структуре данных также фиксировано, но они не обязаны все указывать на реально существующие адреса;

- ветви в нашей задаче строятся до тех пор, пока это возможно, различные ветви могут иметь различную длину.

Эти отличия существенны, но они не касаются общей структуры программы. Следовательно, общую конструкцию мы можем вполне взять из примера двоичного дерева. Это рекурсивная процедура, получающая на вход координаты текущей позиции и выполняющая следующие действия:

```
Для каждого из 4-х возможных направлений рубки
Если в рассматриваемом направлении рубка возможна, то
   Осуществляем рубку.
   Ищем свободный указатель и от него создаем указатель на новый
   элемент списка, который и будет содержать информацию о новой
   позиции рубящей шашки.
   Вызываем процедуру с координатами нового положения шашки.
```

Далее будем поступать так, как мы уже поступали, а именно, зададим себе вопрос, что в нашем алгоритме неопределенного. Сформулировав неопределенности, мы получим информацию для дальнейшего анализа. Неопределенности есть и достаточно существенные, например:

- не вполне понятно, как определить, возможно данное направление или нет. И что такое направление вообще;

- в чем заключается процедура рубки;

- предполагается, что возможных направлений рубки 4. Это не так. Их только 3, так как с одного из направлений шашка пришла, и его, как направление рубки, рассматривать нельзя. Но если мы желаем одно из направлений исключить, то мы должны придумать, как его отделить от других трех;

- и самое главное — из сказанного ранее совершено не ясно, как мы собираемся обходить построенный граф в поисках самого длинного продолжения, хотя может быть пока это и не важно.

Попробуйте подумать хотя бы над первым вопросом. Даже интуитивно ясно, что в нашей структуре данных нет достаточной информации для ответа на этот вопрос. Это от того, что была совершена серьезная логическая ошибка. Описав необходимую структуру графа и начав рассуждать об алгоритме (программе), мы ничего не сказали о том, как данный граф будет построен. Повторимся еще раз. Необходимо понять не только, что граф будет из себя представлять, но и то, как он строится из первичной информации, то есть из той информации, которая дана непосредственно в условии.

Первичная информация — это позиция на доске. Ее можно представить, например, двумерным массивом. Договоримся, что пустая клетка будет представлена нулем. Шашка, осуществляющая рубку, пусть изображается числом 2, а ее противники пусть будут 1.

Теперь можно ответить на поставленные вопросы.

- Направление — это вектор, такой что, если его прибавить к координатам (индексам) элемента двумерного массива доски, то мы получим элемент массива, представляющий клетку, в которую возможно осуществить ход. Таких направлений четыре: (–2, –2), (–2, 2), (2, –2), (2, 2).

- Для того чтобы выяснить допустимость направления, необходимо установить истинность двух утверждений: первое — ячейка массива, полученная прибавлением к координатам (индексам) текущего элемента вектора направления, содержит ноль (клетка доски пуста); второе — ячейка массива, полученная прибавлением половины вектора направления, содержит единицу (клетка доски содержит шашку противника).

- Процедура рубки заключается в двух действиях:

- изменении координат (индексов) текущего элемента на вектор направления;

- обнулении элемента, полученного прибавлением половины вектора направления.

Проблема отбрасывания одного из направлений решается двумя способами. Можно запоминать откуда пришла шашка, для чего в элементе связного списка придется вводить по крайней мере одно дополнительное поле, описывающее откуда был совершен переход. А усложнение структур данных всегда приводит к усложнению логики. Но можно ничего этого и не делать, если немного подумать о том, как проводится проверка на допустимость направления.

Только что было сказано, что одно из условий допустимости — это наличие единицы в ячейке через "половину вектора направления", а в описании процедуры рубки сказано, что эта единица обнуляется. Это означает, что направление, откуда шашка пришла при расчете следующего хода будет недопустимым (там после выполнения хода стоит ноль). Следовательно, мы можем проблему допустимости проигнорировать и проверять все четыре направления. Одна из проверок окажется лишней, но в качестве компенсации отпадет необходимость в дополнительных операциях по проверке допустимости, которые почти наверняка окажутся более сложными, а следовательно, мы получим и еще выигрыш в упрощении логики программы.

*Проблема восстановления данных.* Переборные алгоритмы, а мы имеем дело именно с перебором, требуют «отката», то есть восстановления данных в некоторой точке перебора с целью построения нового варианта. В нашей ситуации «откат» заключается в восстановлении одной единственной шашки, срубленной данным ходом. Из смысла процедуры рубки ясно, что срубленная шашка находится ровно посредине между исходной и конечной позицией рубящей шашки (при рубке одной шашки противника). То есть координаты поля, на котором стояла шашка (индексы элемента массива, в котором находилась единица), можно найти как среднее арифметическое координат исходного и конечного поля. Это в свою очередь потребует запоминания информации об исходном поле.

Все, что обсуждалось до сих пор, касается только построения графа (связного списка), а теперь мы подходим к главному, к проблеме поиска наидлиннейшего пути рубки или в наших терминах наидлиннейшей ветви графа или, что то же самое, наидлиннейшей ветви связного списка.

Решение этой задачи сводится к необходимости помечать элементы связного списка, через которые проходит наидлиннейший путь или наидлиннейшие пути. Можно, конечно, создать дополнительную структуру, в которую наидлиннейший путь будет записываться, но, как уже было сказано, дополнительные структуры данных — это дополнительное усложнение логики и может быть существенное. Поэтому мы лучше создадим еще одно поле в элементе связного списка для сохранения новой информации. Это тоже дополнительная структура, но она уже локальная.

Конечно, хотелось бы, чтобы эта дополнительная информация была одним числом, (что проще) и это логично, так как существует число, характеризующее длину маршрута — сама длина. Если в каждом элементе будет записано число, являющееся длиной самого протяженного маршрута рубки, проходящего через данный элемент, то алгоритм поиска станет очень прост:

```
Числовую характеристику начального узла объявить максимальной.
Пока не достигнут тупик, делать
  Проверить все поля ссылок
```

***Если ссылка не тупиковая, то***
***Если элемент, на который указывает ссылка, содержит характеристику, равную максимальной, то перейти по этой ссылке.***

Нетрудно заметить, что данный алгоритм обеспечивает поиск не одного маршрута, а всех существующих. Трудность в понимании, наверное, вызывает первый пункт, в котором числовая характеристика начального узла объявляется максимальной. Не вполне понятно, на каком основании. Для полного понимания необходимо рассмотреть механизм получения такой характеристики, но даже сразу можно заметить, что наидлиннейший маршрут обязан проходить через начальный элемент, а так как числовая характеристика длины только одна, то она длиннейший маршрут и определяет.

А теперь все-таки о механизме. Значение длины маршрута известно только на последнем поле (последнем элементе связного списка). Следовательно, механизм определения характеристики длины должен заключатся в прописывании этого конечного числа в дополнительные поля, при возврате по маршруту. Пусть шашка находится в некоторой точке А и предстоит принять решение о числовой характеристике для пункта А. Предположим, что из А выходит максимальное количество ненулевых указателей (то есть три), это означает, что характеристику для А необходимо выбирать из трех возможных значений, которые возвращены в А рекурсивными вызовами. Какое из них выбрать? Конечно же, максимальное.

И последнее, когда принимать решение о числовой характеристике для текущего элемента (поля доски)? Конечно же, тогда, когда будут отработаны все допустимые направления рубки. Из этого следует, что формировать числовую характеристику мы можем в процессе построения графа (связного списка). Все существенные технические проблемы обсуждены, можно записать алгоритм. Головной алгоритм видимо будет состоять из двух процедур:

- алгоритм построения связного списка — графа;
- алгоритм поиска длиннейшего маршрута.

**Алгоритм построения связного списка:**

1. ***Ввод данных в виде двумерного массива***
2. ***Создание начального элемента списка с записью в него исходного положения рубящей шашки***
3. ***Вызов рекурсивной процедуры построения связного списка***
4. ***Запись в начальный элемент полученной длины длиннейшего маршрута***

**Рекурсивная процедура построения связного списка:**

```
Входные данные: адрес на текущий элемент списка, длина уже
пройденного маршрута
Запомнить адрес на текущий элемент
Обработка направления (-2, -2)
  Максимальная длина = возвращенной длине
Восстановить адрес на текущий элемент
Обработка направления (2, -2)
  Длина = возвращенной длине
  Если Длина > Максимальной То Максимальная = Длине
Восстановить адрес на текущий элемент
Обработка направления (2, 2)
  Длина = возвращенной длине
  Если Длина > Максимальной То Максимальная = Длине
Восстановить адрес на текущий элемент
Обработка направления (-2, 2)
  Длина = возвращенной длине
  Если Длина > Максимальной То Максимальная = Длине
Восстановить адрес на текущий элемент
Записать Максимальную длину, как характеристику длины маршрута,
для текущего элемента связного списка.
```

**Обработка направления:**

```
Проверить допустимость хода в заданном направлении
Если Ход допустим, То
  Ищем свободный (то есть указывающий в никуда) указатель на
адрес
  элемента списка
  Создаем новый элемент списка
  Записываем в новый элемент координаты новой позиции рубящей
шашки
  Вызываем процедуру построения связного списка с новыми
  фактическими параметрами
  Восстанавливаем позицию
  Возвращаем длину, равную пройденной длине
Иначе
  Возвращаем длину, меньшую пройденной длины на 1 (так как ход
  оказался недопустимым)
```

**Программа поиска длиннейшего пути.**

Алгоритм мы уже построили. Сейчас заметим только, что процесс обхода связного списка должен отображаться на экране. Для этого распечатаем массив доски и на каждом шагу после перехода к очередному пункту маршрута в координатах сохраненных в полях элемента связного списка будем распечатывать элемент доски, но другим цветом. В программе для выделения используется зеленый цвет. Текст программы приведен в листинге ниже.

Листинг

```
program example;
 uses crt,graph;
 type
  nodes=^zapis;
  zapis=record
         x,y,num:integer;
         uk1,uk2,uk3:nodes;
        end;
 var
  desk:array[1..8,1..8] of integer;
  k,n,i,x,y,l,max:integer;
  list
  :nodes;
function hod(list:nodes;num:integer):integer;
var
 from:nodes;
 n,max:integer;
function move(x,y:integer):integer;
var
 num1:integer;
begin
 if (list^.x+x>0) and (list^.x+x<9) and (list^.y+y>0) and (list^.y+y<9)
  then
    if (desk[list^.x+x,list^.x+x]=0)
          and (desk[list^.x+(x div 2),list^.y+(y div 2)]=1) then
  begin
   desk[list^.x+(x div 2),list^.y+(y div 2)]:=0;
   if list^.uk1=nil then
    begin
     new(list^.uk1);
     list:=list^.uk1;
    end
   else
   if list^.uk2=nil then
    begin
     new(list^.uk2);
     list:=list^.uk2;
    end
   else
    begin
     new(list^.uk3);
     list:=list^.uk3;
    end;
   list^.x:=from^.x+x;list^.y:=from^.y+y;
   num1:=hod(list,num+1);
   desk[(list^.x+from^.x) div 2,(list^.y+from^.y) div 2]:=1;
   list:=from;
   move:=num1;
  end
  else move:=num-1
 else move:=num-1;
end;
begin
 from:=list;
 max:=move(-2,-2);
 n:=move(2,-2);
 if n>max then max:=n;
 n:=move(2,2);
 if n>max then max:=n;
 n:=move(-2,+2);
 if n>max then max:=n;
 list^.num:=max;
 hod:=max;
end;
```

```
procedure search(list:nodes);
var
 i,j,num:integer;
 node:nodes;
procedure print(node:nodes);
var
 mynode:nodes;
begin
 gotoxy(2*node^.x,2*node^.y);write(desk[node^.x,node^.y]);
 if node^.uk1<>nil then
   begin
    mynode:=node^.uk1;
    if mynode^.num=num then print(mynode);
   end;
 if node^.uk2<>nil then
    begin
     mynode:=node^.uk2;
     if mynode^.num=num then print(mynode);
    end;
  if node^.uk3<>nil then
     begin
      mynode:=node^.uk3;
      if mynode^.num=num then print(mynode);
     end
end;
begin
 textcolor(15);
 for i:=1 to 8 do
  for j:=1 to 8 do
   begin
    gotoxy(i*2,j*2);write(desk[i,j]);
   end;
 textcolor(1);
 num:=list^.num;
 print(list);
end;
begin
 clrscr;
 for i:=1 to 8 do
  for k:=1 to 8 do
    desk[i,k]:=0;
 read(n);
 for i:=1 to n do
  begin
   read(x,y);
   desk[x,y]:=1;
  end;
 read(x,y);
 clrscr;
 desk[x,y]:=2;
 textcolor(15);
 for i:=1 to 8 do
  for k:=1 to 8 do
   begin
    gotoxy(i*2,k*2);write(desk[i,k]);
   end;
 readkey;
 new(list);
 list^.x:=x;list^.y:=y;
 list^.num:=hod(list,1);
 search(list);
 readkey;
end.
```

**В заключение**

Мы решили достаточно сложную задачу на построение и обход графа. Есть смысл еще раз выделить ключевые моменты логических построений.

- Переборная задача на графе — это перебор всех возможных путей.

- Перебор всех возможных путей осуществляется двумя взаимообратными операциями: путь в глубину и возврат.

- Для того чтобы обеспечить возврат, необходимо обеспечить запоминание данных перед тем, как совершать очередной шаг в глубину.

- Если нас интересует характеристика, значение которой будет известно только в конечной точке, а использование этой характеристики предполагается в начале графа, то есть смысл ее значение при возврате переносить из конца в начало.