Метод минимакса

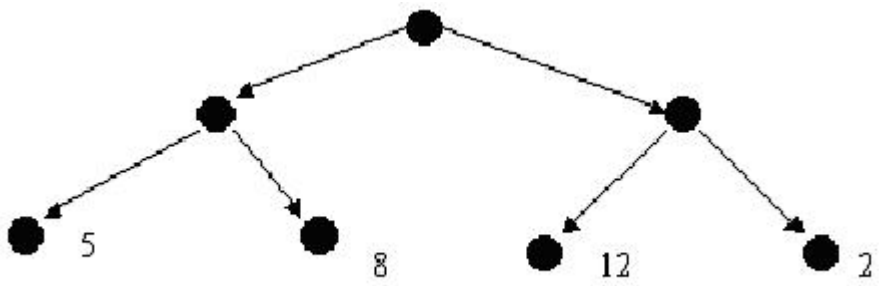
Минимакс — это метод принятия решений используемый тогда, когда выбирать приходится из множества вариантов, устроенного в виде дерева. Так, например, устроено множество вариантов шахматной или шашечной партии. Ситуация выбора такого типа встречается достаточно часто, но в наиболее чистом виде она представлена все же в играх. Представьте себе любую игру, в которую умеете играть, и любую игровую ситуацию. Предположим, что продолжить эту ситуацию можно некоторым количеством вариантов. Каждый закачивается положением, которое можно оценить с точки зрения того, насколько оно хорошо для принимающего решение. Как именно будет получена такая оценка сейчас не важно, договоримся лишь о том, что оценка это число.

Оценки конечных позиций всех возможных вариантов — это единственная информация, которую можно использовать для определения желательного варианта (другой информации просто нет). Метод минимакса как раз и описывает, как, используя информацию об оценках конечных позиций, принимать решения о наилучшем ходе.

Для дальнейших рассуждений придумаем гипотетическую игру. Смысл этой игры не имеет значения, договоримся лишь, что из каждой позиции возможно только два варианта продолжения. Будем такую игру называть двоичной.

Метод анализа строится из тех соображений, что оба противника играют наилучшим образом. Назовем противников ***Первым*** и ***Вторым***. Пусть оценочная функция строится для ***Первого*** (как именно она будет выглядеть сейчас неважно). Тогда из нашего предположения следует, что ***Первый*** стремится к наилучшей оценке, но ***Второй*** оставляет ему только наихудшие, и следовательно, реально ***Первый*** может рассчитывать только на лучшую из худших оценок.

Чтобы понять, как это, построим дерево вариантов для двоичной игры. Как уже было сказано в этой игре из каждой позиции возможно только два продолжения. Построим для нее дерево вариантов:



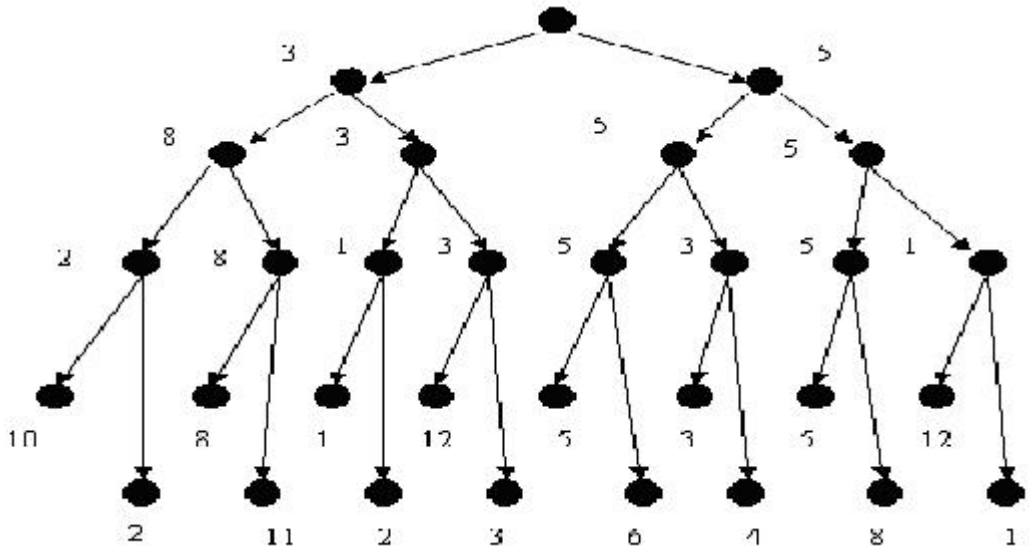
В нижних точках проставлены значения оценочной функции. ***Первый*** игрок должен решить выбрать ли ему продолжение влево или продолжение вправо. Может показаться, что ***Первый*** должен ходить вправо, так как это направление обещает позицию с максимально большой оценкой 12. Но дело в том, что он не сразу попадает в кружок окончательной позиции. Вначале игра переходит в позицию, в которой решение принимает ***Второй***, и он естественно выберет вариант с оценкой 2, в результате ***Первый*** вместо ожидаемых 12 очков получит только два. В варианте левой ветви, ***Второй*** даст ***Первому*** вариант с 5 очками (это наихудшая). Таким образом, на наилучшие оценки 8 и 12 ***Первый*** может не рассчитывать, его реальный выбор из оценок 5 и 2. 5 лучшая и, следовательно, ***Первому*** предпочтительнее ход влево. Поэтому метод и называется методом минимакса

Опишем процесс поиска минимаксной оценки. Для этого назовем альтернативами позиции (узлы дерева вариантов), имеющие одного и того же непосредственного предка (то есть узел уровнем выше). Будем осуществлять подъем по дереву вариантов с самого нижнего уровня. На каждом шаге подъема для каждой группы альтернатив выполним следующие операции:

- определим наибольшую и наименьшую оценку;

- если в узле предке решение принимал первый игрок (для которого рассчитывается минимаксная оценка), то оценка узла предка равна максимальной оценке, полученной из альтернатив, иначе оценка узла предка равна минимальной.

Продемонстрируем эту процедуру на более сложном примере



На первом шаге снизу решение принимает ***Второй*** игрок и естественно, он выбирает тот вариант, при котором оценка будет минимальной. На втором шаге выбор за ***Первым***, и он естественно выбирает из оценок, полученных на предыдущем уровне наилучшие (это оценки 8, 3, 5, 5). На следующем шаге опять решение принимает ***Второй*** игрок, и он опять оставит из полученных на предыдущем шаге наихудшие, то есть 3 и 5. И наконец, последний шаг подъема — решение принимает опять ***Первый***, он выбирает наилучшую оценку 5. Таким образом, из полученной минимаксной оценки можно сделать вывод, что для первого игрока более предпочтительным является вариант с ходом вправо, с оценкой 5.

Разобравшись с постановкой задачи можно переходить к ее решению. Решение разделим на две независимые части. Во-первых, необходимо построить двоичное дерево и завершающие узлы заполнить оценками (в качестве оценок будем брать случайные числа). Во-вторых, нужно организовать собственно минимакс.

Функция, в которой реализуется минимакс, отличается от процедуры создания дерева наличием возвращаемого значения. Механизм возврата работает различными способами, для последнего узла и для промежуточных. В последнем узле в качестве возвращаемого значения, конечно, будет значение оценки, других возможностей просто

нет. Функция, обрабатывающая промежуточный узел, вызывает две функции для обработки дочерних узлов и соответственно получает в качестве вариантов два значения, из которых она должна сделать выбор. Выбор же зависит от того, какого из игроков представляет данный вызов функции. То есть какой из игроков на данном уровне дерева вариантов принимает решение. ***Первый*** игрок, для которого проводится анализ, принимает решение на нечетных уровнях, ***Второй*** — на четных. Поэтому какое значение передается выше (большее или меньшее) определяется четностью номера вызова.

Текст программы приведен в листинге.

Листинг

```
program example;
 uses crt;
 type
  node=^zapis;
  zapis=record
             ocenka:integer;
             left,right:node;
            end;
 var
  uk:node;
procedure Create_tree(uk:node;n:integer);
begin
 if n<5 then
  begin
   new(uk^.left);
   Create_tree(uk^.left,n+1);
   new(uk^.right);
   Create_tree(uk^.right,n+1);
  end
 else uk^.ocenka:=random(20);
end;
function Minimax(uk:node;n,x:integer):integer;
 var
  left,right,k:integer;
begin
 if n<5 then
  begin
   left:=Minimax(uk^.left,n+1,x-(20 div n+1)+n);
   right:=Minimax(uk^.right,n+1,x+(20 div n)-n);
   if n mod 2=0 then
    if left<right then k:=left else k:=right
   else
    if left<right then k:=right else k:=left;
  end
else k:=uk^.ocenka;
gotoxy(x,n*4);write(k);
Minimax:=k;
end;
begin
 randomize;
 clrscr;
 new(uk);
 Create_tree(uk,1);
 Minimax(uk,1,40);
end.
```

В заключение

Решенная задача не очень сложна. Но относительно простой она получилась только потому, что удалось использовать уже имеющиеся результаты. А именно, мы использовали процедуру обхода древовидной структуры данных. Кстати говоря, эту процедуру вполне можно принять в качестве стандартного средства построения и обхода различных деревьев. Поэтому если вам вдруг встретится задача, связанная с обработкой древовидных структур данных, вы можете считать ее уже частично решенной.