

Поиск путей по дереву игры

Наумов Д.А., доц. каф. КТ

Алгоритмы и структуры данных, 2021

Содержание лекции

- 1 Поиск пути. Дерево игры. Дерево поиска
- 2 Оценочная функция
- 3 Концепция поиска путей
- 4 Минимаксный метод
- 5 Альфа-бета отсечение

Поиск пути

Чтобы решить задачу, когда нет ясного вычисления допустимых решений, используется поиск пути.

- с помощью **деревьев игр** для игр с двумя игроками;
- с помощью **деревьев поиска** для игр для одного игрока.

Эти подходы опираются на дерево состояния:

- *корневой узел* - начальное состояние
- *ребра* - потенциальные ходы, которые преобразуют состояние в новое состояние.

Дерево игры

Поиск - сложная проблема, базовая структура не вычисляется полностью из-за взрывного роста количества состояний. (деревья строятся по требованию по мере необходимости).

Шашки: $5 \cdot 10^{20}$ различных позиций на доске.

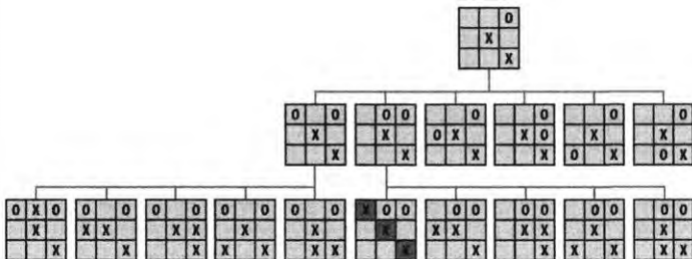
Дерево игры

- Два игрока по очереди выполняют ходы, которые изменяют состояние игры из ее первоначального состояния.
- Есть много состояний, в которых любой игрок может выиграть в игре.
- Могут быть некоторые состояния «ничьей», в которых не выигрывает никто.
- Алгоритмы поиска пути увеличивают шанс, что игрок выиграет или обеспечит ничью.

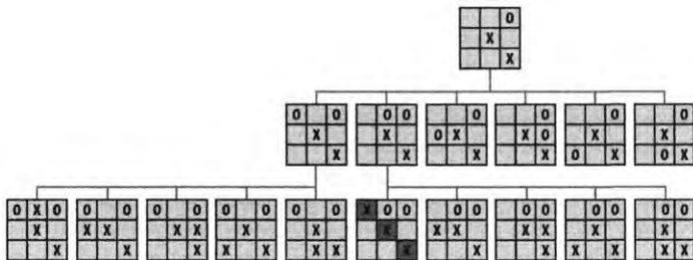
Дерево поиска

- Один игрок начинает игру с некоторого начального состояния и делает допустимые ходы до достижения желаемого целевого состояния.
- Алгоритм поиска пути определяет точную последовательность ходов, которые преобразуют исходное состояние в целевое.

Крестики-нолики: **765** уникальных позиций (без учета отражений и поворотов), **26830** возможных игр.



Дерево игры известно также как дерево И/ИЛИ, поскольку оно образуется из двух различных типов узлов.



- Верхний узел является узлом ИЛИ, так как целью игрока О является выбор только одного из шести доступных ходов на среднем уровне.
- Узлы среднего уровня являются узлами И, потому что цель (с точки зрения игрока 0) состоит в том, чтобы убедиться, что все ходы противника (показанные как дочерние узлы на нижнем уровне) по-прежнему приведут к победе 0 или ничьей.

- В сложной игре дерево игры никогда не может быть вычислено полностью из-за его размера.
- Целью алгоритма поиска пути является определение на основе состояния игры хода игрока, который максимизирует его шансы на победу в игре (или даже гарантирует ее).
- Таким образом, мы преобразуем множество решений игрока в задачу поиска пути в дереве игры.

Пример: игра в шашки

- доска размером 8x8 с начальным набором из 24 шашек (12 белых и 12 черных).
- может ли игрок, делающий первый ход, обеспечить ничью или победу?
- Schaeffer, J., N. Burch, Y. Bjornsson, A. Kishimoto, M. Muller, R. Lake, P. Lu, and S. Sutphen, «Checkers is solved», Science Magazine, September 14, 2007, 317(5844): 1518-1522,
<http://www.sciencemag.org/cgi/content/abstract/317/5844/1518>.

Два типа подходов для поиска задач поиска:

Тип А

- Рассмотрим различные разрешенные ходы для обоих игроков для фиксированного количества будущих ходов и определим наиболее благоприятное положение, получающееся в результате для исходного игрока.
- Затем выберем начальный ход, который движет игру в этом направлении.

Shannon, C., «Programming a computer for playing chess», Philosophical Magazine, 41(314): 1950, [http: //tinyurl.com/ChessShannon-pdf](http://tinyurl.com/ChessShannon-pdf).

Тип Б

- Добавим некоторые адаптивные решения, основанные на знании игры, а не на статических оценках.
- Говоря более точно,
 - ① оценим перспективные позиции на столько ходов вперед, сколько необходимо для выявления устойчивой позиции, в которой вычисления действительно отражают силу полученной позиции,
 - ② выберем подходящие доступные ходы.
- Этот подход пытается предотвратить возможность бессмысленной траты драгоценного времени

Рассмотрим семейство алгоритмов типа А

- Minimax;
- AlphaBeta;
- NegMax.

который предоставляет подход общего назначения для поиска в дереве игры наилучшего хода для игрока в игре с двумя игроками.

Существует несколько способов сделать поиск более интеллектуальным:

- Выбор порядка и количества применяемых разрешенных ходов
 - При рассмотрении доступных ходов для данного состояния игры сначала следует вычислить ходы, которые вероятнее других приведут к успешным результатам.
 - Кроме того, можно отказаться от определенных ходов, которые заведомо не приводят к успешным результатам.
- Выбор состояния игры для «обрезки» дерева поиска
 - По мере выполнения поиска может быть обнаружена новая информация, которую можно использовать для устранения состояний игры, которые (в некоторый момент времени) были выбраны в качестве части поиска.

Функция статических оценок (оценочная функция)

оценивает состояния игры в промежуточных точках в процессе вычислений с последующим упорядочением множества доступных ходов так, чтобы сначала испытывались ходы, с более высокой вероятностью ведущие к выигрышу

Функция статической оценки должна:

- учитывать различные особенности позиции в дереве игры
- возвращать целочисленный балл, который отражает относительную силу позиции с точки зрения игрока.

Samuel, A., «Some studies in machine learning using the game of checkers», IBM Journal 3(3): 210-229, 1967,
<http://dx.doi.org/10.1147/rd.116.0601>.

- оценка позиции на доске путем рассмотрения двух десятков параметров, таких как сравнение количества шашек у игрока и его противника или возможностей размена шашек.

Крестики-нолики

Функция *BoardEvaluation*

определенная Нилом Нильссоном. (Nilsson, N., Problem-Solving Methods in Artificial Intelligence. McGraw-Hill, 1971)

Пусть $nc(gs, p)$ — количество строк, столбцов или диагоналей в состоянии игры gs , в которых игрок p все еще может выстроить в ряд три своих значка.

Затем мы определим $score(gs, p)$ следующим образом:

- $+\infty$ - если игрок p победил в игре в состоянии gs ;
- $-\infty$ - если противник игрока p победил в игре в состоянии gs ;
- $nc(gs, p) - nc(gs, opponent)$ - если ни один игрок в состоянии игры gs не победил.

Представление состояния

Каждый узел дерева игры или дерева поиска содержит всю информацию о состоянии, известную как позиция в игре.

Пример: *шахматы, рокировка возможна, если:*

- ❶ ни одна из этих фигур еще не делала хода,
- ❷ промежуточные клетки доски пустые и в настоящее время не находятся под боем фигур противника
- ❸ король в настоящее время не находится под шахом.

Состояние игры:

- должно храниться наиболее компактно;
- может быть значительно уменьшено путем удаления эквивалентных состояний, которые можно получить простым поворотом или отражением

Сложные представления для шахмат или шашек: Pepicelli, G., «Bitwise optimization in Java: Bitfields, bitboards, and beyond,» O'Reilly on Java.com, February 2, 2005,

<http://www.onjava.com/pub/a/onjava/2005/02/02/bitsets.html>.

Вычисление доступных ходов

Чтобы найти наилучший ход, в каждом состоянии должно быть возможно вычисление доступных ходов игрока.

Коэффициент ветвления

среднее количество ходов, которые разрешены из отдельного состояния.

- кубик Рубика - 13,50
- четыре в ряд - 7,00
- шашки - 6,14 (1,20 для позиций со взятием, 7,94 без взятия)
- Го - 361,00

Если коэффициент ветвления у игры оказывается высоким, а ходы не упорядочены должным образом на основе некоторой вычисляемой меры успеха, слепой поиск в дерева оказывается неэффективным.

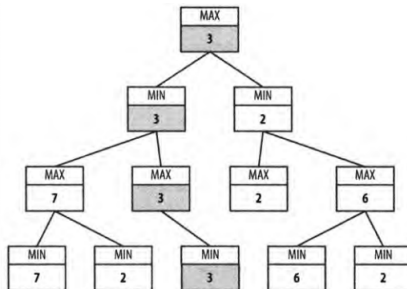
Максимальная глубина расширения

- Из-за ограниченных ресурсов памяти некоторые алгоритмы поиска ограничивает степень расширения деревьев поиска и игр.
- Этот подход проявляет свою слабость, в первую очередь, в играх, в которых продуманная стратегия формируется с помощью последовательности ходов.
- Фиксированная глубина расширения формирует «горизонт», за который поиск не может заглянуть, и это часто препятствует успешному поиску.
- Для игр с одним игроком фиксация максимальной глубины означает, что алгоритм не в состоянии найти решение, которое лежит сразу за горизонтом.

Например, в шахматах нередко жертва фигуры для получения потенциальных преимуществ. Если эта жертва происходит на краю максимального расширения, выгодное состояние игры может не быть обнаружено.

Minimax

- Для заданной конкретной позиции в дереве игры с точки зрения начального игрока программа поиска должна найти ход, который привел бы к наибольшим шансам на победу (или хотя бы на ничью).
- Но вместо только текущего состояния игры и доступных для этого состояния ходов программа должна рассматривать любые ходы противника, которые он делает после хода нашего игрока.



Minimax (к предыдущему рисунку)

- оценочная функция возвращает целое число, представляющее оценку состояния игры с точки зрения игрока;
- дерево игры расширяется путем рассмотрения будущих состояний игры после последовательности из n ходов.
- каждый уровень дерева по очереди представляет собой уровень MAX (на котором цель заключается в обеспечении выгоды для исходного игрока путем максимизации вычисленного состояния игры) и MIN (на котором цель заключается в обеспечении выгоды для противника путем минимизации вычисленного состояния игры).
- на чередующихся уровнях программа выбирает ход, который максимизирует оценочную функцию, но на следующем уровне она предполагает, что противник будет выбирать ход, который минимизирует ее значение.

Алгоритм Minimax

Наилучший, средний и наихудший случаи: $O(b^{\text{ply}})$

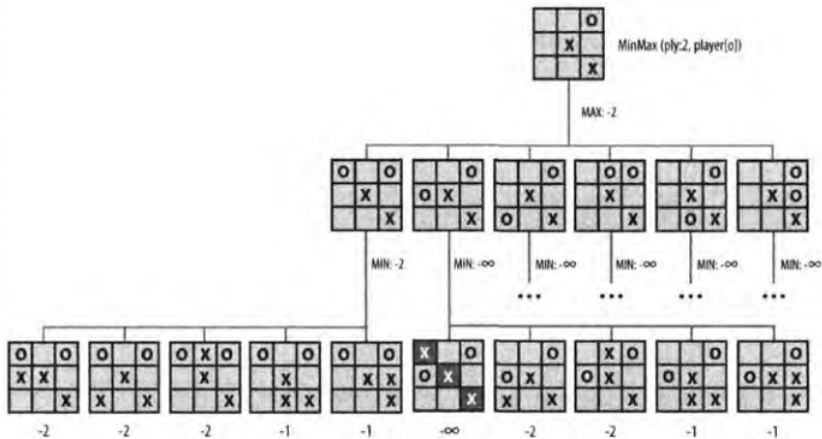
```

bestmove (s, player, opponent)
    original = player
    [move, score] = minimax (s, ply, player, opponent)
    return move
end

minimax (s, ply, player, opponent)
    best = [null, null]
    if ply равен 0 или допустимых ходов нет then
        score = оценка состояния s для исходного игрока
        return [null, score]
    foreach допустимый ход m игрока player в состоянии s do
        Выполнение хода m в состоянии s
        [move, score] = minimax(s, ply-1, opponent, player)
        Отмена хода m в состоянии s
        if player == original then
            if score > best.score then best = [m, score]
        else
            if score < best.score then best = [m, score]
        return best
    end

```

- ❶ Запоминаем исходного игрока, так как оценка состояния всегда выполняется с его точки зрения.
- ❷ Если больше ходов не остается, игрок побеждает (или проигрывает), что эквивалентно достижению целевой глубины предпросмотра.
- ❸ При каждом рекурсивном вызове выполняется обмен игрока и противника, отражающий чередование ходов.
- ❹ Последовательное чередование уровней между MAX и MIN.

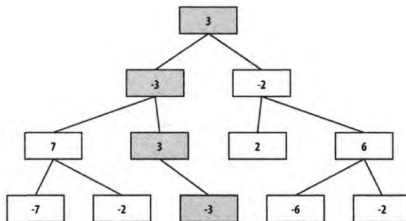


Анализ алгоритма Minimax

- Если для каждого состояния игры имеется фиксированное количество ходов b (или даже когда количество доступных ходов уменьшается на единицу с каждым уровнем), общее количество состояний игры, в которых выполняется поиск при предпросмотре глубиной d , представляет собой $O(b^d)$ демонстрируя экспоненциальный рост.
- Ограничения глубины предпросмотра могут быть устранены, если дерево игры достаточно мало для полной оценки за приемлемый промежуток времени.
- Поскольку мы предполагаем, что и игрок, и противник играют без ошибок, мы должны найти способ остановить расширение дерева игры после того, как алгоритм определяет, что дальнейшее изучение данного поддерева бессмысленно.

NegMax

- Алгоритм NegMax заменяет чередование уровней MAX и MIN алгоритма Minimax единым подходом, используемым на каждом уровне дерева игры.
- Состояние игры всегда оценивается с точки зрения игрока, делающего начальный ход (что требует от функции оценки хранения этой информации).
- Алгоритм NegMax последовательно ищет ход, который дает максимум из значений дочерних узлов состояния с обратным знаком.



NegMax

Алгоритм NegMax**Наилучший, средний и наихудший случаи: $O(b^{\text{ply}})$**

```

bestmove (s, player, opponent)
    [move, score] = negmax (s, ply, player, opponent)
    return move
end

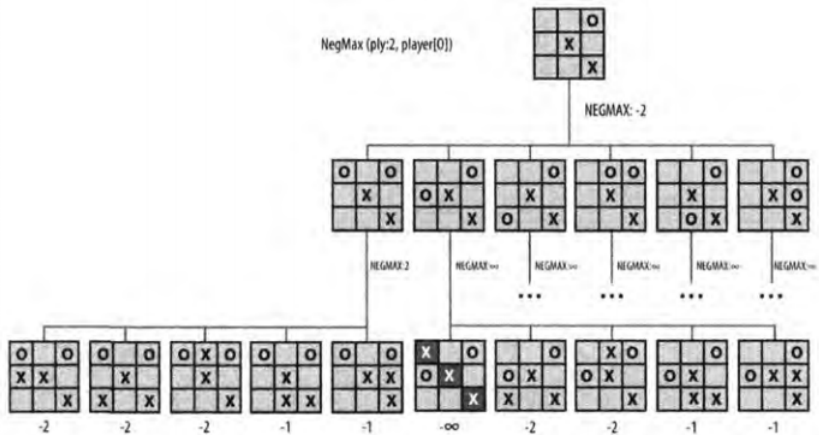
negmax (s, ply, player, opponent)
    best = [null, null]
    if ply == 0 или больше нет допустимых ходов then
        score = Вычисление s для игрока
        return [null, score]

    foreach Допустимый ход m для игрока в состоянии s do
        Выполнение хода m для s
        [move, score] = negmax (s, ply-1, opponent, player)    ❶
        Отмена хода m для s
        if -score > best.score then best = [m, -score]          ❷
    return best
end

```

- ❶ Алгоритм NegMax обменивает игроков на каждом следующем уровне.
- ❷ Выбор наибольшей среди оценок дочерних узлов с обратным знаком.

NegMax



AlphaBeta-процедура

AlphaBeta-процедура

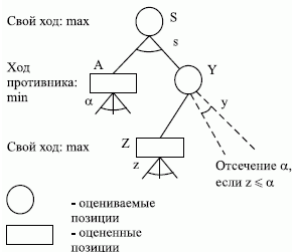
эквивалентная минимаксу процедура, при которой из дерева игры исключаются целые ветви без проведения анализа.

В основе этой процедуры лежит идея Дж. Маккарти об использовании двух переменных, обозначенных α, β .

Основная идея метода: сравнение наилучших оценок, полученных для полностью изученных ветвей, с наилучшими предполагаемыми оценками для оставшихся.

- McCarthy J. Human Level AI Is Harder Than It Seemed in 1955 (LaTeX2HTML 27 November 2006).
- Newell A., Simon H. A. Computer Science as Empirical Inquiry: Symbols and Search (англ.) // Communications of the ACM, Vol. 19, No. 3 : journal. — 1976. — March.

AlphaBeta-процедура



- Пусть позиция A полностью проанализирована и найдено значение ее оценки α .
- Допустим, что один ход из позиции Y приводит к позиции Z, оценка которой по методу минимакса равна z .
- Предположим, что $z \leq \alpha$.
- После анализа узла Z, когда справедливо соотношение $y \leq z \leq \alpha \leq s$, ветви дерева, выходящие из узла Y, могут быть отброшены (альфа-отсечение).

AlphaBeta-процедура

- Если мы захотим опуститься до узла Z , лежащего на уровне произвольной глубины, принадлежащей той же стороне, что и уровень S , то необходимо учитывать минимальное значение оценки β , получаемой на ходах противника.
- Отсечение типа β можно выполнить всякий раз, когда оценка позиции, возникающая после хода противника, превышает значение β .
- Алгоритм поиска строится так, что оценки своих ходов и ходов противника сравниваются при анализе дерева с величинами α и β соответственно.
- В начале вычислений этим величинам присваиваются значения $+\infty$ и $-\infty$, а затем, по мере продвижения к корню дерева, находится оценка начальной позиции и наилучший ход для одного из противников.

AlphaBeta-процедура

Правила вычисления α и β :

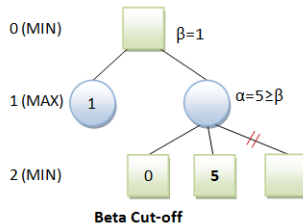
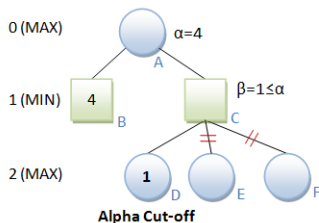
- 1 у MAX вершины значение α равно наибольшему в данный момент значению среди окончательных возвращенных значений для ее дочерних вершин;
- 2 у MIN вершины значение β равно наименьшему в данный момент значению среди окончательных возвращенных значений для ее дочерних вершин.

Правила прекращения поиска:

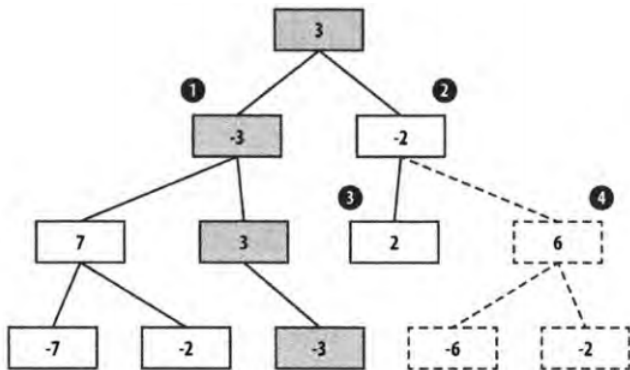
- 1 можно не проводить поиска на поддереве, растущем из всякой MIN вершины, у которой значение β не превышает значения α всех ее родительских MAX вершин;
- 2 можно не проводить поиска на поддереве, растущем из всякой MAX вершины, у которой значение α не меньше значения β всех ее родительских MIN вершин.

AlphaBeta

Алгоритм AlphaBeta определяет стратегию последовательного обрезания всех непродуктивных поисков в дереве



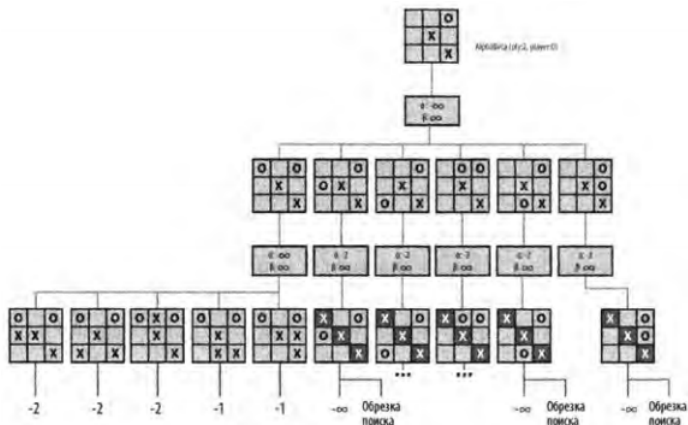
AlphaBeta



AlphaBeta

- После оценки поддерева игры с корнем в (1) алгоритм AlphaBeta знает, что если этот ход сделан, то противник не может сделать позицию хуже, чем -3.
- Это означает, что лучшее, что может сделать игрок, — добиться состояния с оценкой 3.
- Когда AlphaBeta переходит в состояние игры (2), то его первый дочерний узел — состояние (3) — имеет оценку, равную 2.
- Это означает, что если выбрать ход, приводящий в (2), то противник может заставить игрока перейти в состояние игры с оценкой, меньшей, чем лучшая найденная к этому моменту (т.е. 3).
- Таким образом, не имеет смысла проверять поддерево с корнем в (4), и оно полностью отбрасывается.

Предпросмотр AlphaBeta глубиной 2



Предпросмотр AlphaBeta глубиной 2

- Когда алгоритм AlphaBeta ищет лучший ход, он помнит, что X может достичь оценки не выше 2, если O делает ход в левый верхний угол.
- Для каждого другого хода O AlphaBeta определяет, что X имеет хотя бы один ход, который превосходит первый ход O (на самом деле для всех прочих ходов O игрок X может победить).
- Таким образом, дерево игры расширяется только до 16 узлов, что представляет собой экономию более 50% по сравнению с алгоритмом Minimax.

Алгоритм AlphaBeta

Наилучший и средний случаи: $O(b^{ply/2})$; наихудший случай: $O(b^{ply})$

```

bestmove (s, player, opponent)
  [move, score] = alphaBeta (s, ply, player, opponent, -∞, ∞) ❶
  return move
end

alphaBeta (s, ply, player, opponent, low, high)
  best = [null, null]
  if ply == 0 или нет допустимых ходов then ❷
    score = вычисление s для игрока player
    return [null, score]

  foreach Корректный ход m для игрока player в состоянии s do
    execute move m on s
    [move, score] = alphaBeta(s, ply-1, opponent, player, -high, -low)
    Отмена хода m для s

    if -score > best.score then
      low = -score
      best = [m, -low]
      if low ≥ high then return best ❸
    end
  end
  return best
end

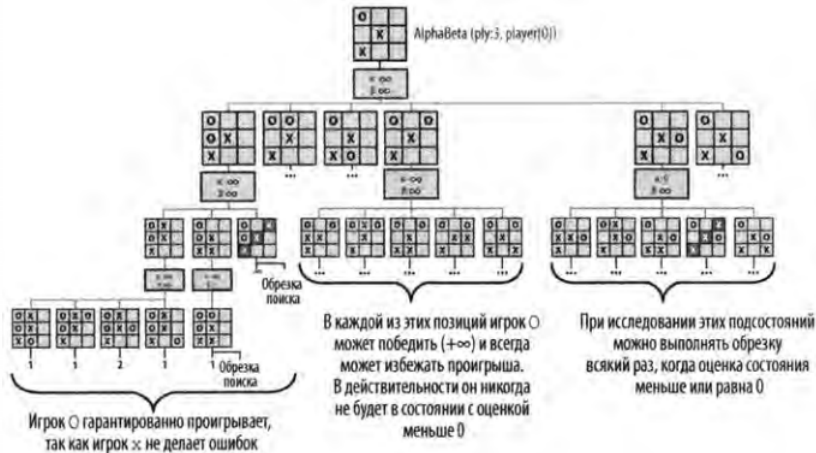
```

- ❶ В начале худшее, что может сделать игрок, – проиграть ($low = -\infty$).
Лучшее, что может сделать игрок, – выиграть ($high = +\infty$).
- ❷ AlphaBeta оценивает листья, как и алгоритм NegMax.
- ❸ Прекращение исследования "братьев", когда наихудшая возможная оценка противника не ниже нашего максимального порога.

Алгоритм AlphaBeta

- Алгоритм рекурсивно выполняет поиск в дереве игры и поддерживает два значения, α и β , которые определяют «окно возможностей» для игрока до тех пор, пока $\alpha < \beta$.
- Значение α представляет нижнюю границу состояний игры, найденную для игрока до настоящего времени (или $-\infty$, если ничего не было найдено) и объявляет, что игрок нашел ход, гарантирующий, что он может достичь как минимум указанного значения. Более высокие значения α означают, что игрок поступает правильно; при $\alpha = +\infty$ игрок выиграл, и поиск можно завершать.
- Значение β представляет верхнюю границу состояний игры, найденную для игрока до настоящего времени (или $+\infty$, если ничего не было найдено) и объявляет максимальное значение, которого игрок может достичь. Если значение β уменьшается все сильнее и сильнее, значит, противник поступает верно и делает наилучшие ходы, ограничивающие доступные для варианты.

Алгоритм AlphaBeta



Анализ алгоритмов

Пусть каждый раз игроки могут сделать b ходов, а алгоритм просматривает на d ходов вперед.

- число вершин MinMax, NegMax - b^d ;

AlphaBeta оценивает b состояний игры для начинающего игрока на каждом уровне, но только одно состояние игры — для его противника.

- число вершин для AlphaBeta - $b^{d/2}$.

Алгоритм AlphaBeta может также исследовать то же общее количество состояний игры, что и алгоритм Minimax, расширяя глубину дерева игры до $2d$.

Глубина	Состояния Minimax	Состояния AlphaBeta	Совокупное уменьшение
6	549 864	112 086	80%
7	549 936	47 508	91%
8	549 945	27 565	95%