

Поиск в пространстве состояний

Пространство состояний удобно представлять в виде графа, **узлы** которого соответствуют различным состояниям окружающего мира или состояниям решаемой задачи, а **дуги** – допустимым операциям, переводящим некий мир из одного состояния в другое, или шагам, выполняемым в процессе решения задачи. Для решения поставленной задачи **необходимо найти** на графе пространства состояний путь от исходного состояния мира к требуемому состоянию.

Пространство состояний представляется четырьмя множествами:

- вершины графа N – возможные состояния мира;
- дуги между вершинами A – возможные операции (действия);
- начальные состояния S (возможны одно или несколько);
- целевые состояния D (возможны одно или несколько).

Допустимым считается любой путь из вершины множества S в вершину множества D .

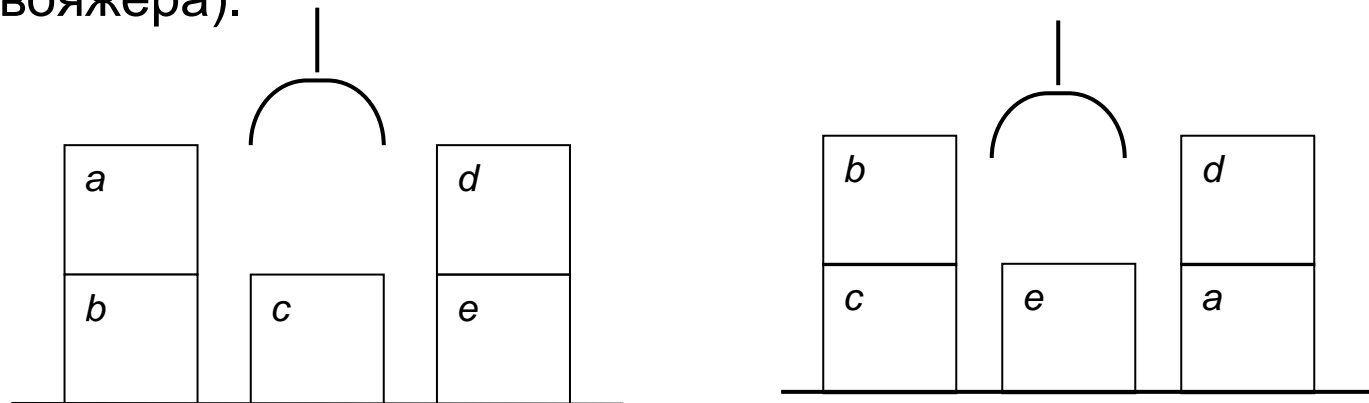
Замечание. Связь между дугами и вершинами можно описать различными способами, например, в виде таблицы, строки и столбцы которой – все узлы, а в ячейке записана информация о дуге, связывающей узлы, соответствующие строке и столбцу.

Начальное и целевое состояния

Начальное состояние задается описанием, например, на языке предикатов как в «мире блоков».

Целевое состояние может быть задано разными способами:

- описание конкретного состояния (например, «мир блоков»)
- описание свойств состояния (например, игра «крестики-нолики»)
- характеристика пути от исходного состояния к цели (например, задача коммивояжера).

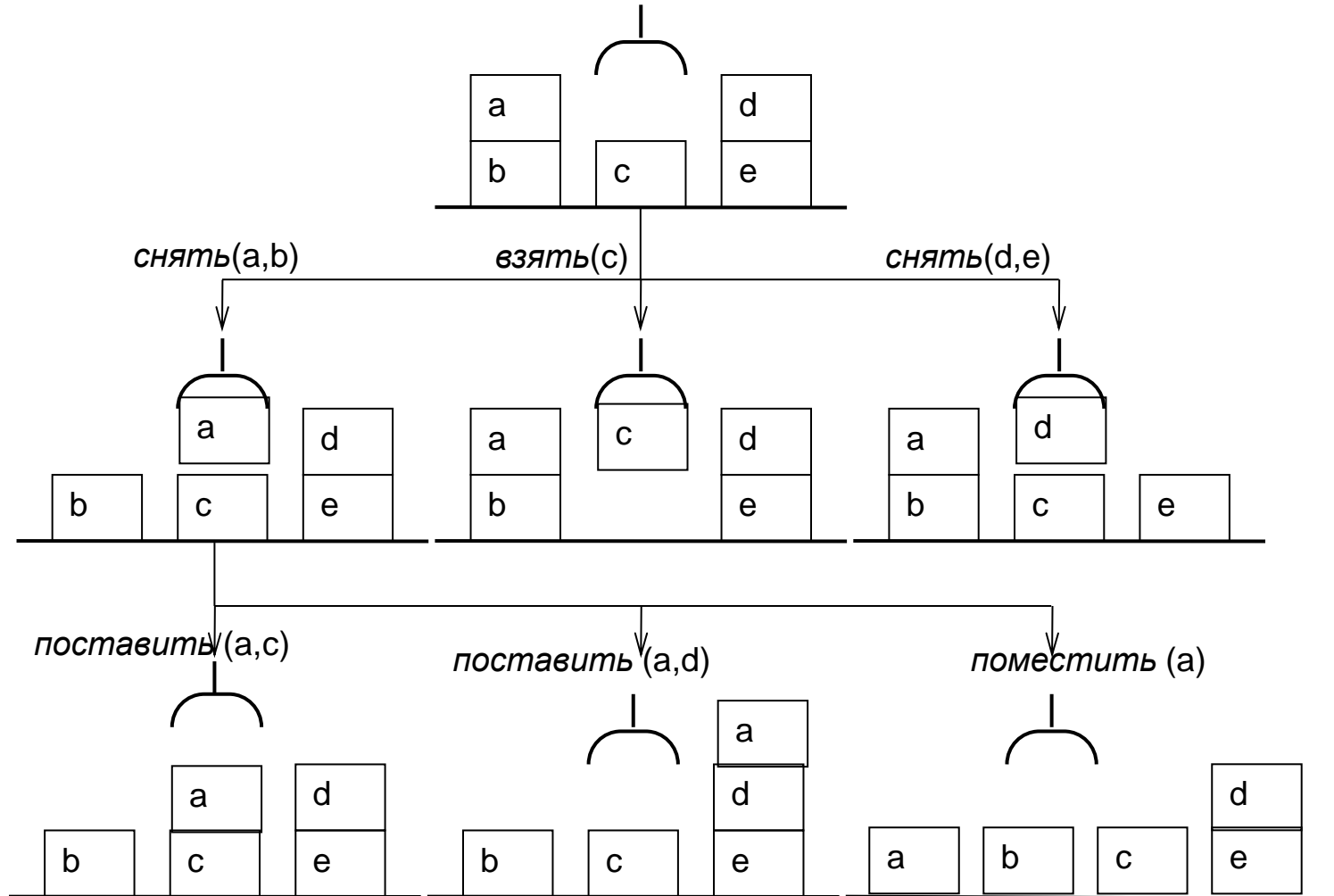


Исходное и целевое состояния в «мире блоков»

Замечание. От способа задания зависит сложность проверки достижения цели.

Примеры графов состояний

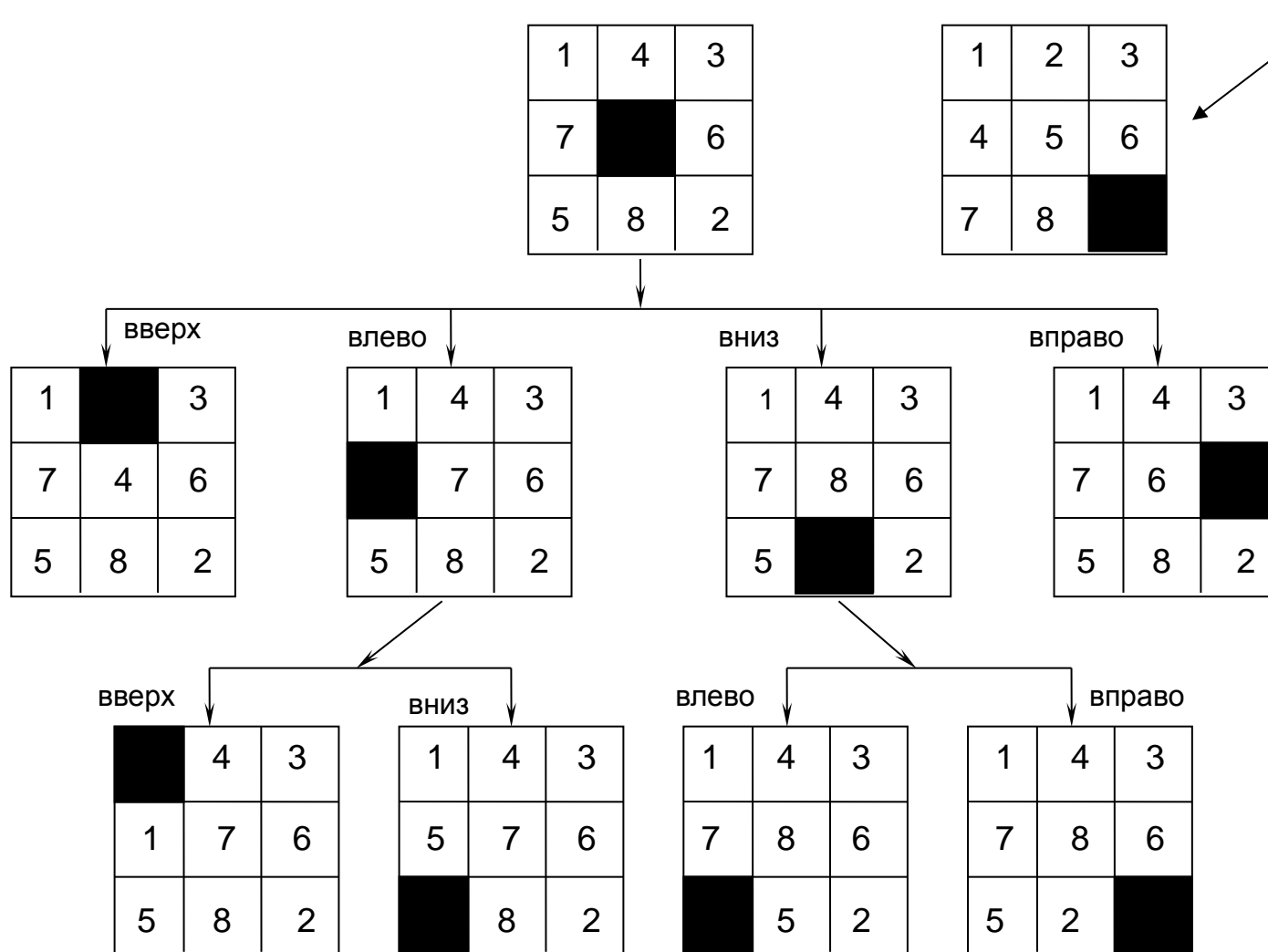
Фрагмент графа возможных перемещений блоков



Замечание. Возможное действие *поставить(a,b)* не приведено, т.к. приведет к циклу

Примеры графов состояний

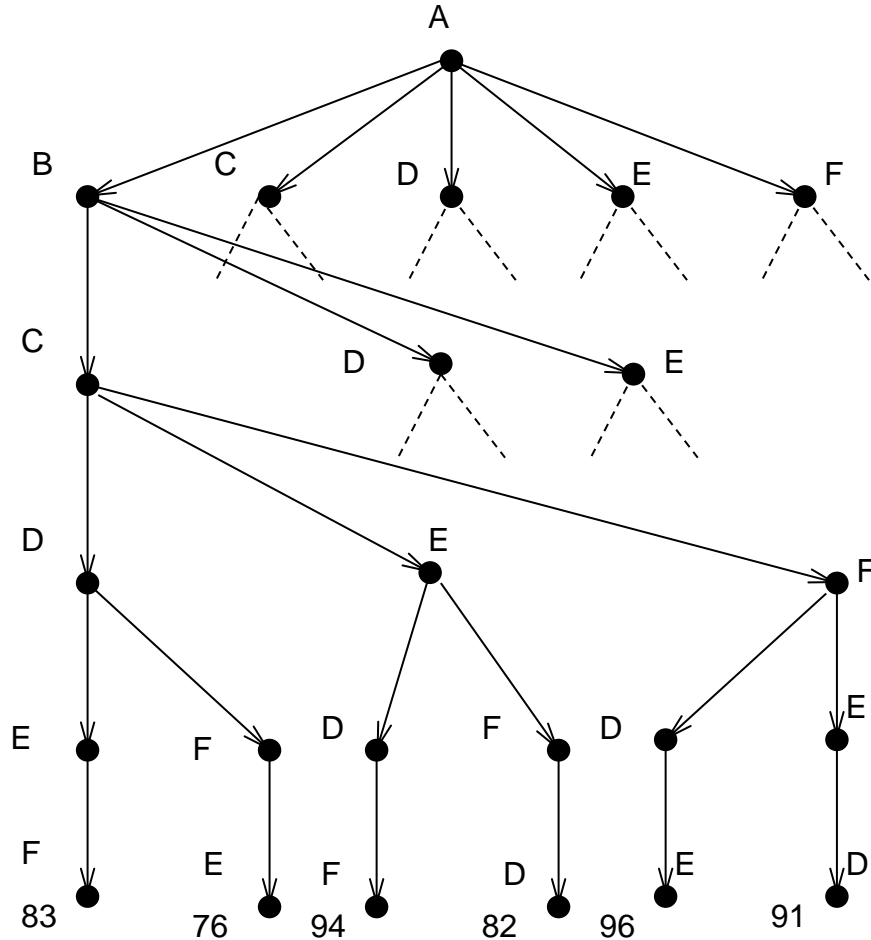
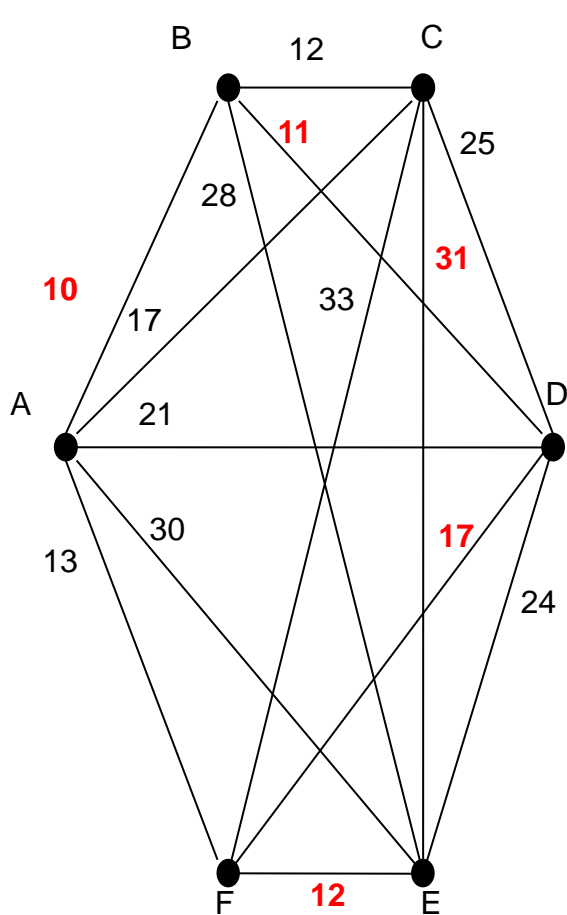
Фрагмент графа игры «головоломка 8» (цель – поставить цифры в заданном порядке)



Возможное действие – перемещение пустой клетки

Примеры графов состояний

Исходные данные и фрагмент графа состояний для задачи коммивояжера



Маршрут минимальной длины – дуги с красными значениями

Поиск последовательным перебором

Алгоритм поиска в пространстве состояний – **поиск с возвратами**. В процессе поиска последовательно генерируются новые возможные состояния и формируются **три списка состояний**:

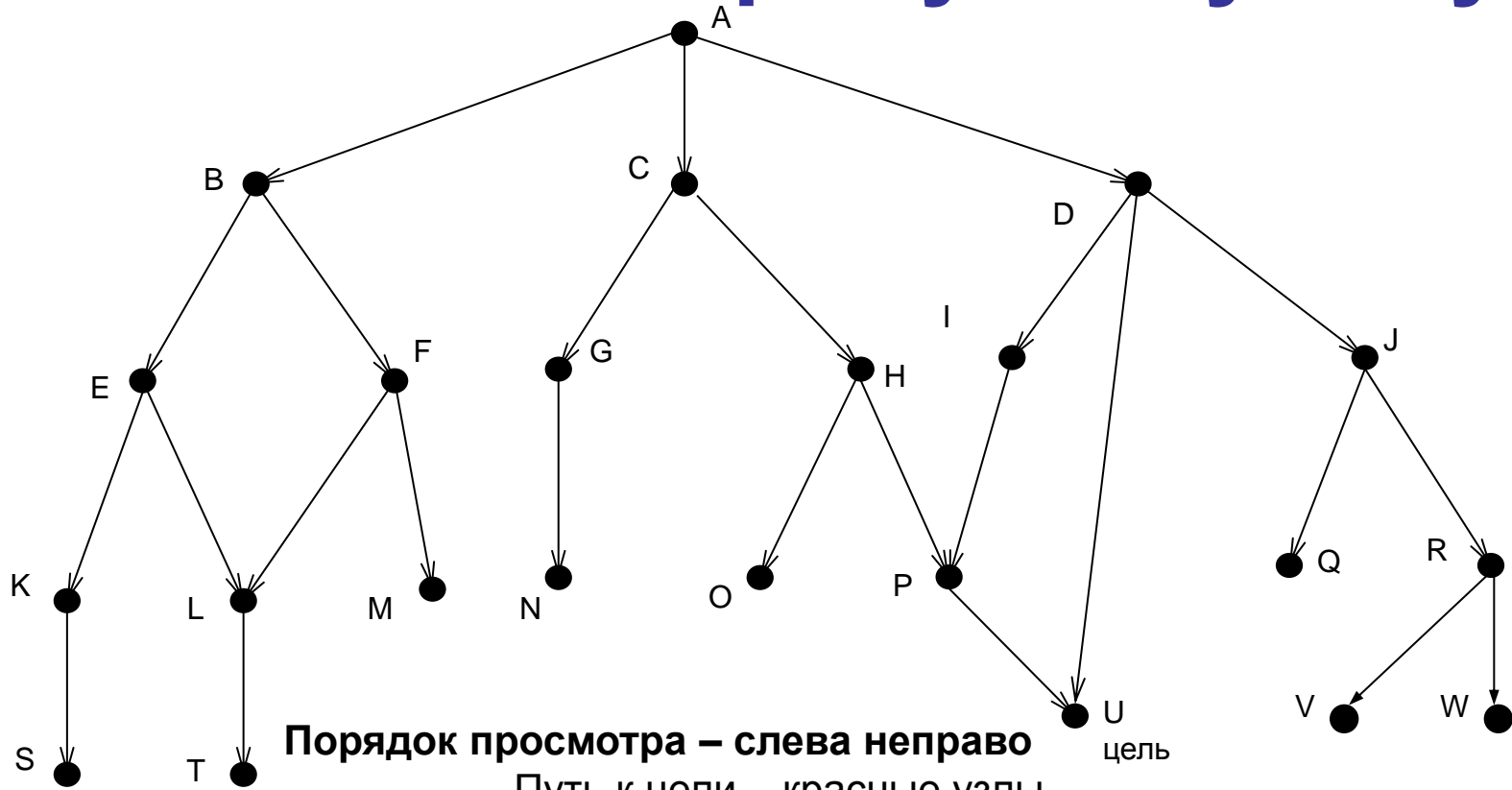
- **open** – список уже сгенерированных, но еще нерассмотренных состояний, для того чтобы можно было вернуться к любому из них;
- **closed** - список уже просмотренных состояний, для того чтобы исключить их повторный просмотр;
- **list** - список узлов текущего просматриваемого пути (возвращается как результат поиска при достижении целевого состояния).

Каждое новое состояние, сгенерированное в процессе поиска, проверяется на вхождение в эти списки (проверка на зацикливание).

Алгоритм поиска с возвратами запускается из начального состояния и следует по пути до тех пор, пока не будет достигнуто целевое состояние или концевая вершина графа состояний. В последнем случае алгоритм возвращается в какую-либо из пройденных вершин и просматривает ее вершины-братья, спускаясь по одной из ветвей.

Замечание. Выбор последовательности генерации и просмотра узлов определяется конкретным алгоритмом поиска пути к цели.

Поиск в ширину и глубину



Поиск в ширину (гарантирует минимальную длину пути)

Очередность просмотра вершин: **A**, B, C, **D**, E, F, G, H, I, **U**;

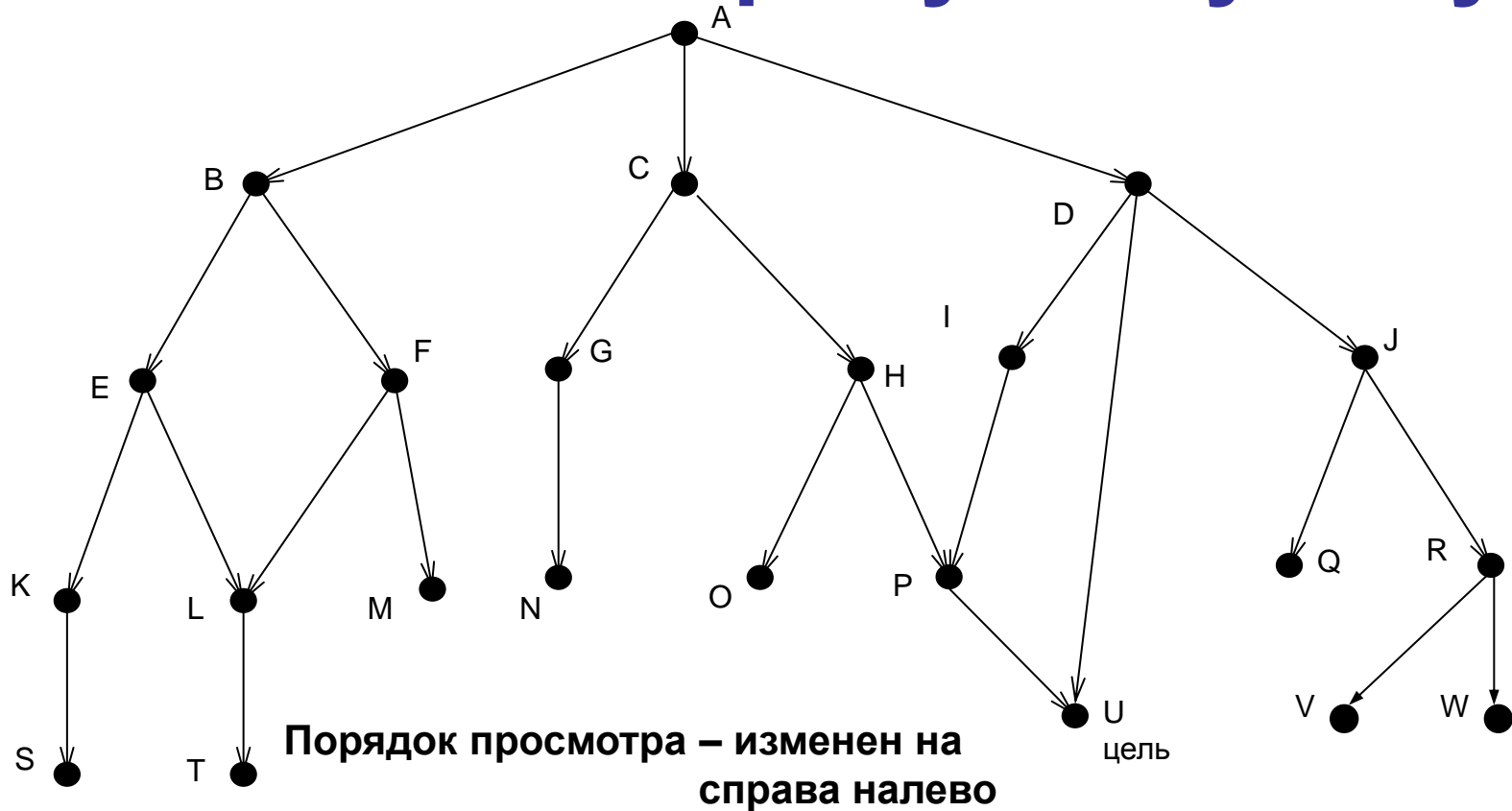
Поиск в глубину (не гарантирует минимальную длину пути)

Очередность просмотра вершин: **A**, B, E, K, S, L, T, F, M, **C**, G, N, **H**, O, **P**, **U**

Итерационное заглубление (не гарантирует минимальную длину пути)

Очередность просмотра вершин: **A**, B, C, **D**, E, K, L, F, M, G, N, H, O, P, I, **U**

Поиск в ширину и глубину



Поиск в ширину. Очередность просмотра вершин: **A, D, C, B, J, U**;

Поиск в глубину. Очередность просмотра вершин: **A, D, J, R, W, V, Q, U**

Итерационное заглубление. Очередность просмотра вершин: **A, D, C, B, J, R, Q, U**

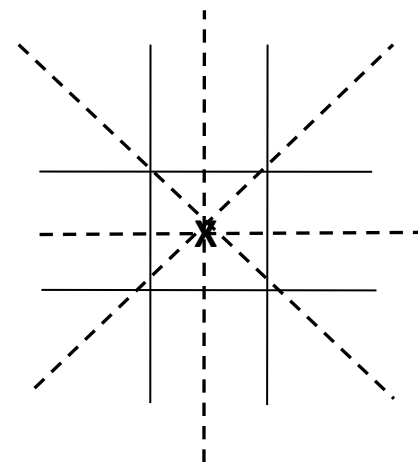
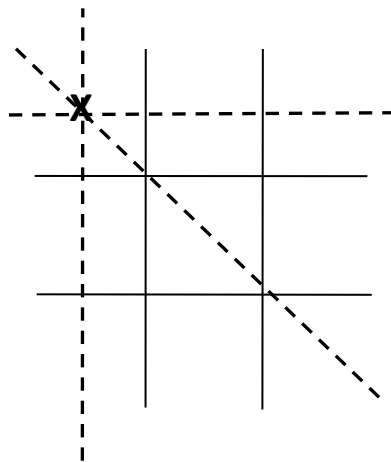
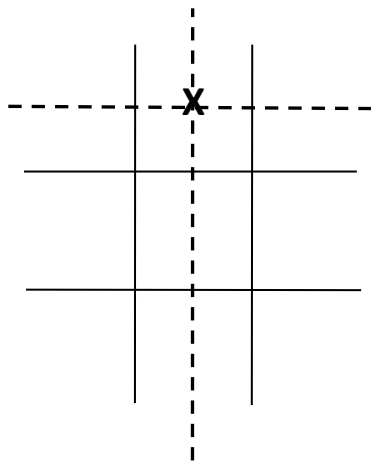
От порядка просмотра узлов зависит число просмотренных узлов и путь к цели

Эвристические алгоритмы

Эвристику приходится использовать если:

- задача не имеет точного решения в связи с неопределенностью в постановке задачи или в исходных данных (медицинская диагностика, система технического зрения);
- задача имеет точное решение, но стоимость поиска неприемлема из-за размеров пространства состояний (игра в шахматы).

Эвристические алгоритмы включают в себя вычисление оценки узлов дерева возможных действий с использованием этой оценки для выбора следующего шага в процессе поиска решения.



Примеры.

Оценка качества первого хода в игре «Крестики-нолики» - число контролируемых линий
 Оценка состояния в «Мире блоков» – число блоков, стоящих на нужном месте

Экстремальный и жадный алгоритмы

Экстремальный алгоритм.

Из возможных действий в текущем состоянии выбирается то, которое приведет к состоянию с экстремальной (максимальной / минимальной) оценкой. Остальные действия из дальнейшего рассмотрения исключаются. Если оценка всех возможных состояний ниже, чем текущее состояние, то алгоритм останавливается.

Основная проблема – остановка при достижении **локального** экстремума функции оценки состояний.

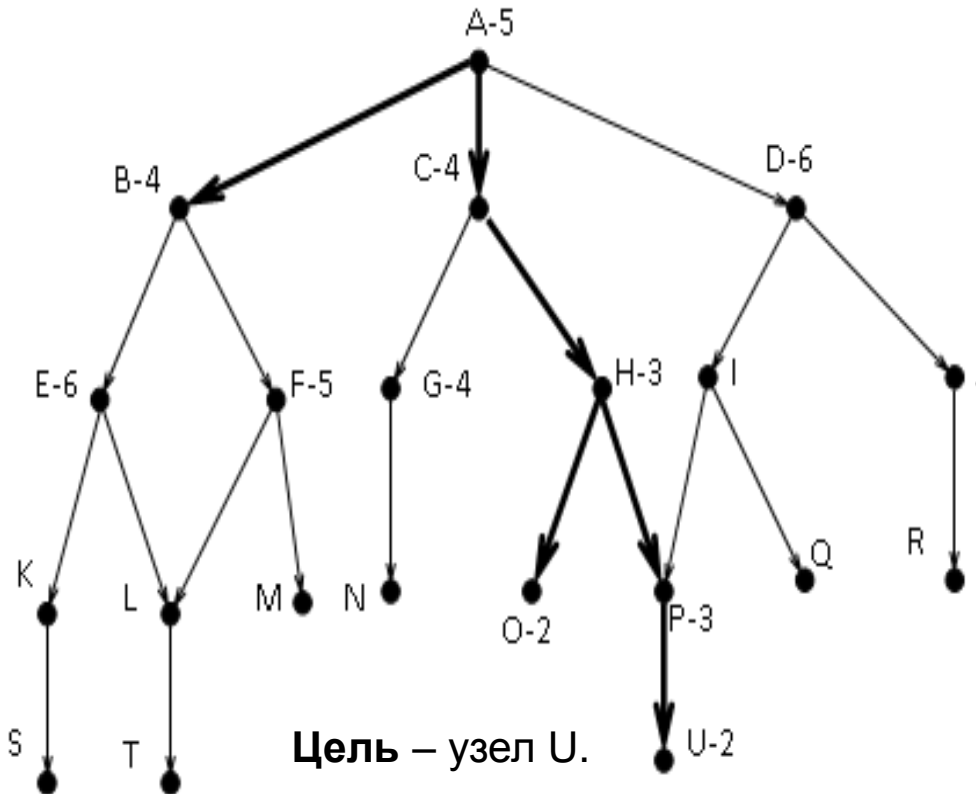
Жадный алгоритм.

Использует список нерассмотренных состояний, присваивая каждому его члену некоторую оценку. При выборе следующего шага алгоритм сортирует список нерассмотренных состояний по уменьшению или увеличению оценки, формируя приоритетную очередь, и выбирает действие, приводящее к состоянию из начала очереди.

Обходит локальный экстремум функции оценки состояний, находит кратчайший путь.

Недостаток – нет оценки **цепочек** возможных **последующих** состояний.

Примеры алгоритмов



Оценка состояния – чем **меньше**, тем лучше
Возможно наоборот – чем **больше**, тем лучше

Экстремальный алгоритм

остановится в узле В или О (зависит от способа выбора узла при совпадении оценок узлов В и С)

Жадный алгоритм выйдет из тупиков В и О.

Жадный алгоритм.

В начале:

open: A-5; *closed*: пусто.

Для А потомки (В,С,Д):

open: **B-4**, C-4, D-6; *closed*: A.

Для В потомки (Е, F):

open: **C-4**, F-5, D-6, E-6; *closed*: B, A.

Для С потомки (G, H):

open: **H-3**, G-4, F-5, D-6, E-6;
closed: C, B, A.

Для Н потомки (О, P):

open: **O-2**, P-3, G-4, F-5, D-6, E-6;
closed: H, C, B, A.

Для О потомков нет, цель не достигнута

Выбирается узел Р из очереди:

open: **P-3**, G-4, F-5, D-6, E-6;
closed: O, H, C, B, A.

Для Р потомок (U):

open: **U-2**, G-4, F-5, D-6, E-6;
closed: P, O, H, C, B, A.

Для U – **цель**.

Минимаксный алгоритм

Применяется в играх с двумя участниками, каждый из которых стремится к победе и имеет одинаковую информацию о текущем состоянии игры. Основан на предположении, что на каждом шаге противник выбирает наилучший из возможных для него ходов.

MAX – игрок, стремящийся выиграть (максимизировать **свое** преимущество), MIN – его противник, стремящийся соответственно минимизировать преимущество соперника.

Для выбора наилучшего из всех возможных на данный момент ходов строится граф, состоящий из **всех** возможных состояний от текущего до **заданной** глубины.

Далее производится оценка **конечных** узлов, которые передаются корневой вершине.

В качестве наилучшего хода выбирается тот ход, оценка которого максимальна.

Уровни ходов на графе называются соответственно уровни MAX и MIN.

Правила передачи оценки конечных узлов по уровням:

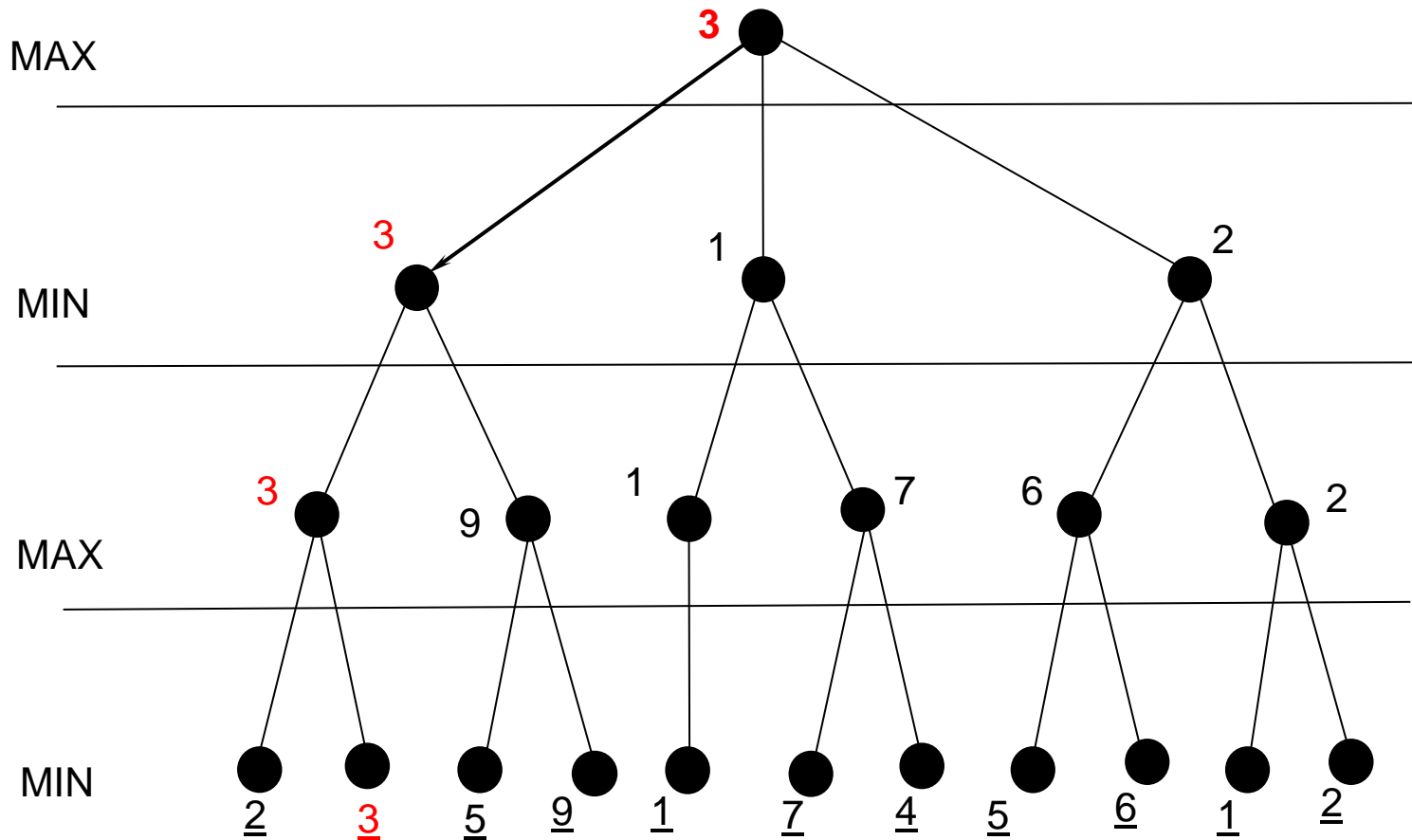
- если родительский узел относится к уровню MAX, то ему присваивается **максимальное** из значений его потомков;
- если родительский узел относится к уровню MIN, то ему присваивается **минимальное** из значений его потомков.

В результате игрок выбирает ход, наилучший из возможных при условии, что противник применяет аналогичную стратегию.

Недостаток – очень быстрый рост графа в ширину при увеличении глубины просмотра.

Замечание. Противник может и не обладать интеллектом, например, окружающий мир.

Минимаксный алгоритм



Оцениваются только узлы нижнего уровня (подчеркнуты). Порядок просмотра и оценки узлов нижнего уровня не имеет значения, т.к. просматриваются все. В качестве наилучшего будет выбран ход с оценкой 3.

Алгоритм альфа-бета усечения

Основа – минимаксный алгоритм.

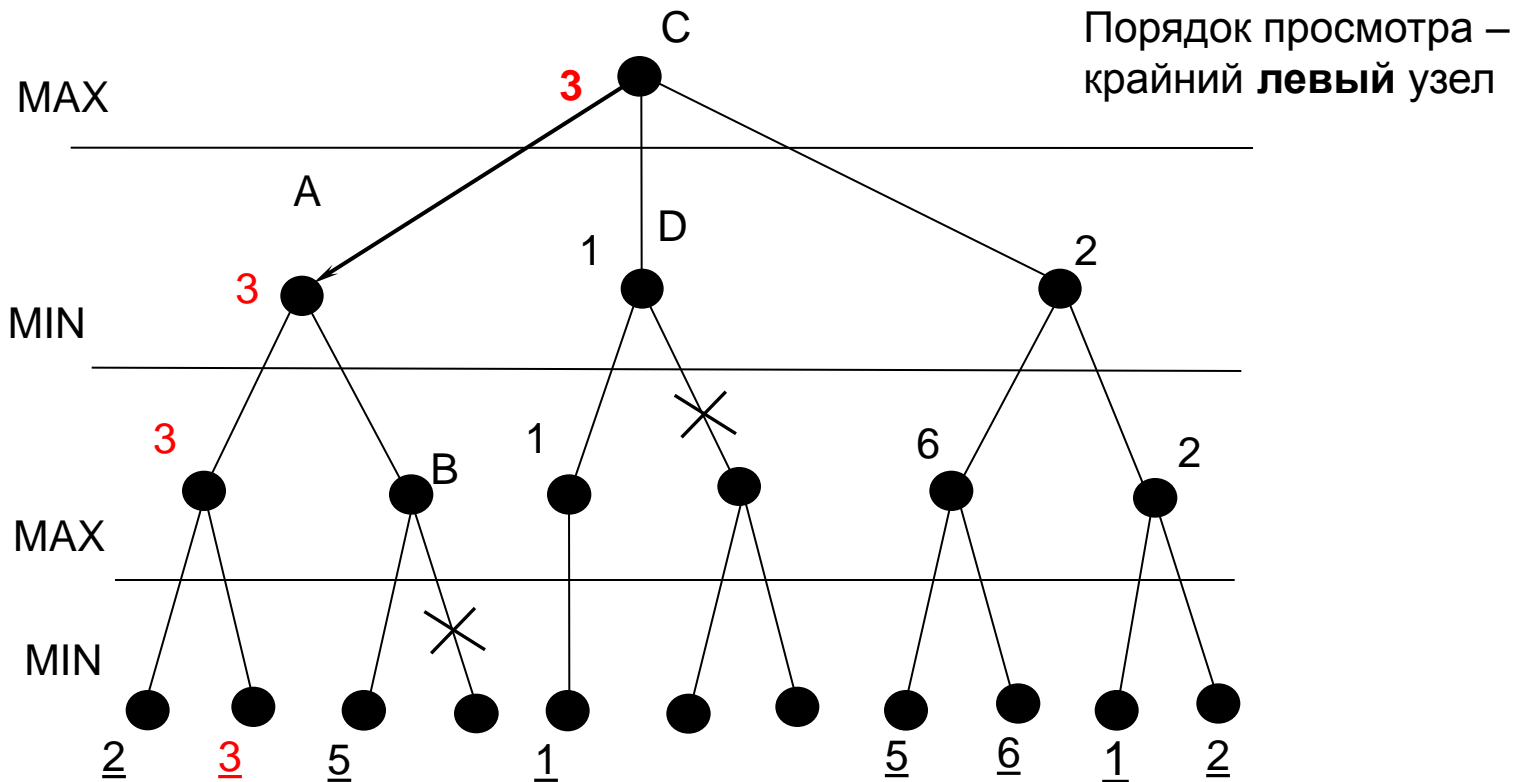
Идея – возможность прекратить построение части графа досрочно, используя результаты анализа уже построенной части.

Выполнить поиск в глубину до заданного уровня и вычислить оценки для состояний этого уровня. Если это узлы уровня MIN, то **максимальная** из этих оценок передается предыдущему родительскому состоянию, т.е. на уровень MAX, и далее на уровень MIN, как **уровень отсечения бета**. Далее опуститься к потомкам этого уровня, игнорируя те состояния, оценка которых больше или равны данному значению бета. Аналогичные процедуры выполняются на уровне MAX для вычисления **уровня отсечения альфа**. Алгоритм прекращает просмотр состояний на промежуточном уровне, если для узла уровня MIN значение бета меньше или равно значению альфа любого из его предков MAX, а для узла уровня MAX значение альфа больше или равно значению бета любого из его предков MIN.

Таким образом, значение альфа, связанное с узлами MAX, никогда не уменьшается, а значение бета, связанное с узлами MIN, никогда не увеличивается. В результате из рассмотрения могут быть исключены как отдельные узлы, так и целые поддеревья, что значительно сужает область поиска, причем поиск производится за один проход.

Недостаток – число просмотров зависит от выбора порядка просмотра в глубину.

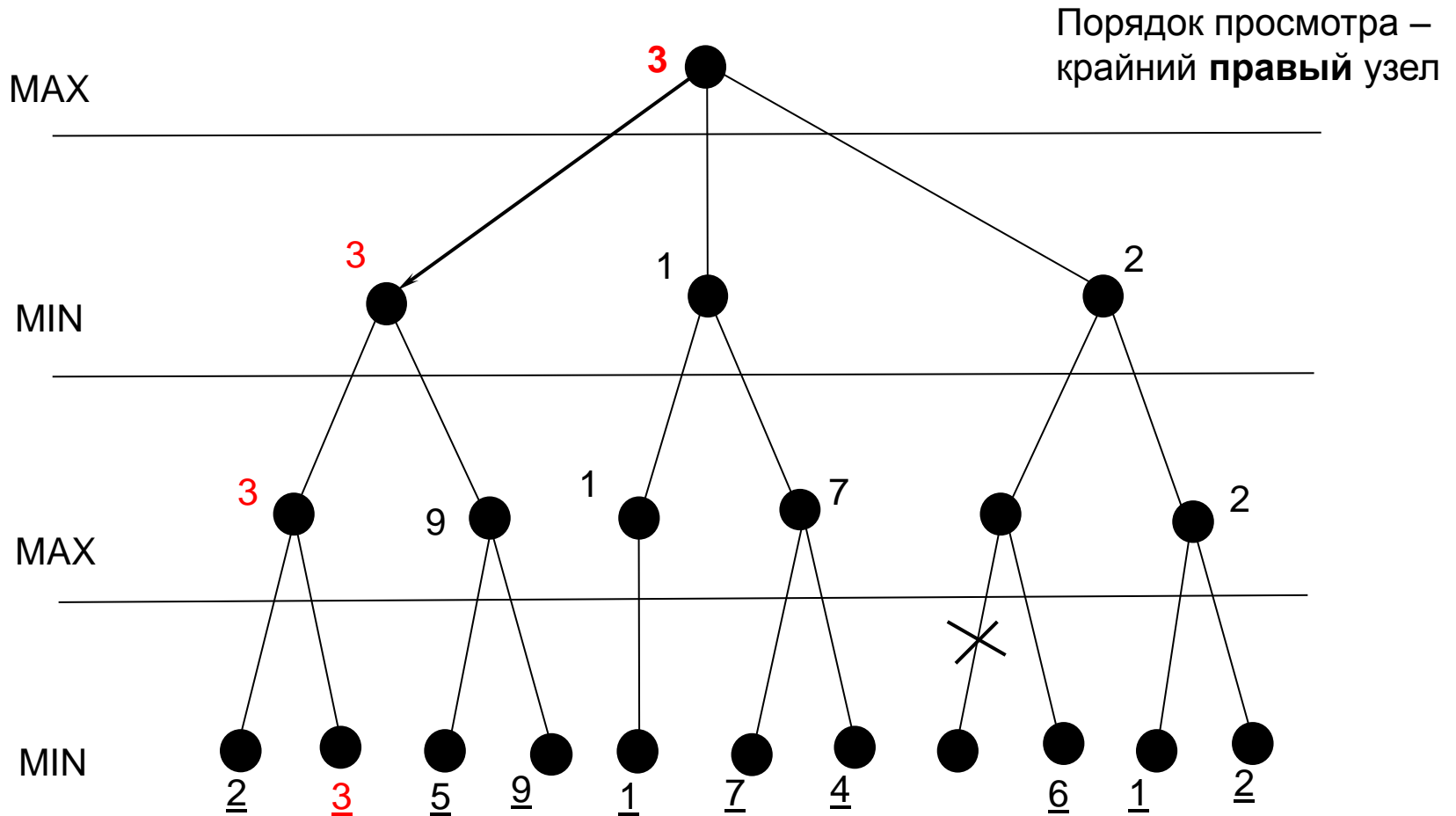
Алгоритм альфа-бета усеечения



Для узла A $\beta = 3$, т.е. оценка узла A будет **не больше** чем 3. Тогда узел B является β -усеченным, т.к. $5 > 3$. Для узла C $\alpha = 3$, т.е. оценка узла C будет **не меньше** чем 3. Тогда узел D является α -усеченным, т.к. $1 < 3$. Отсеченные ветви показаны X.

Результат оценки корневой вершины с применением алгоритма альфа-бета усеечения совпадает с результатом оценки минимаксным алгоритмом, но число сформированных и просмотренных вершин меньше.

Алгоритм альфа-бета усечения



При таком направлении просмотра отсечен один конечный узел.
Отсечение ветвей не происходит.