Trương Hải Bằng, VNU HCM 2018

Thuật toán Mini-Max, Alpha-Beta

Nội dung

- Trò chơi
- Trò chơi đối kháng
- Thuật toán MINIMAX
- Cắt tỉa α–β

Trò chơi

- Trò chơi một trong những đặc tính được xem là "thông minh" của con người
- Trò chơi là phiên bản "F1" của AI
- Có được những thành tựu đáng kể: AlphaGo *9/3/2016 (Google DeepMind)*

Chiến lược minimax

- Giải thuật tìm kiếm Heuristic với các hàm heuristic chỉ thích hợp cho các bài toán không có tính đối kháng. Như các trò chơi chỉ có một người chơi: Puzzle, tìm lối ra mê cung, bài toán n quân hậu,…
- Các trò chơi có tính đối kháng cao, thường là các trò chơi 2 người chơi như: tic tac toa, caro, cờ quốc tế,… giải thuật trên không có tác dụng vì: Đối phương không bao giờ đi theo con đường cho ta có thể đi đến Goal

Thủ tục Min-Max:

Áp dụng trong các trò chơi đối kháng 2 phía. Để ước lượng nước đi tốt dựa trên hàm ước lượng, chúng ta dùng thủ tục Min-Max như sau:

Giả sử một trong hai người chơi:

- Gọi một người là Max: tìm cách làm cực đại hàm ước lượng qua việc xác định giá trị hàm ước lượng ở mỗi nước đi có khả năng rồi chọn nước đi tương ứng với giá trị lớn nhất.
- Nhưng khi đó đối thủ của Max là Min thì lại tìm cách làm cực tiểu giá trị hàm ước lượng này.

Thủ tục min-max

Như vậy ở mỗi mức của cây biểu diễn trò chơi:

- Nếu 1 đỉnh tương ứng với 1 nước đi của Max thì giá trị của đỉnh này sẽ lấy giá trị cực đại của các đỉnh tiếp sau đó.
- Nếu 1 đỉnh tương ứng với 1 nước đi của Min thì giá trị của đỉnh này sẽ lấy giá trị cực tiểu của các đỉnh tiếp sau đó.

Thủ tục min-max

Ví dụ: Trò chơi Tic-Tac-Toe:

Max = X (đi trước)

Min = O

Nguyên tắc: Nếu có 3 con thẳng hàng thì thắng.

Hàm ước lượng:

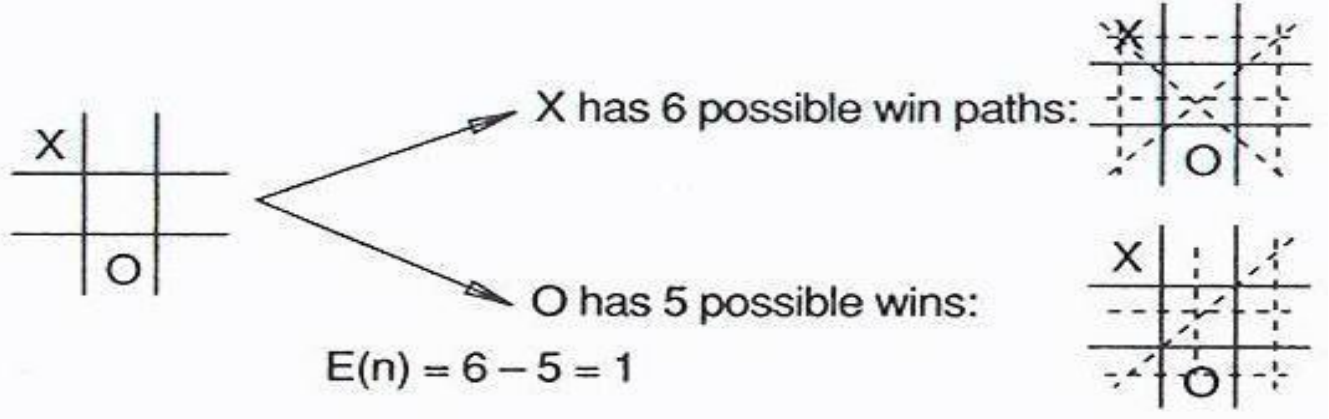
f(x) = (Số dòng, số cột, số đường chéo còn mở đối với Max)-(Số dòng, số cột, số đường chéo còn mở đối với Min)

Heuristic trong trò chơi tic-tac-toe

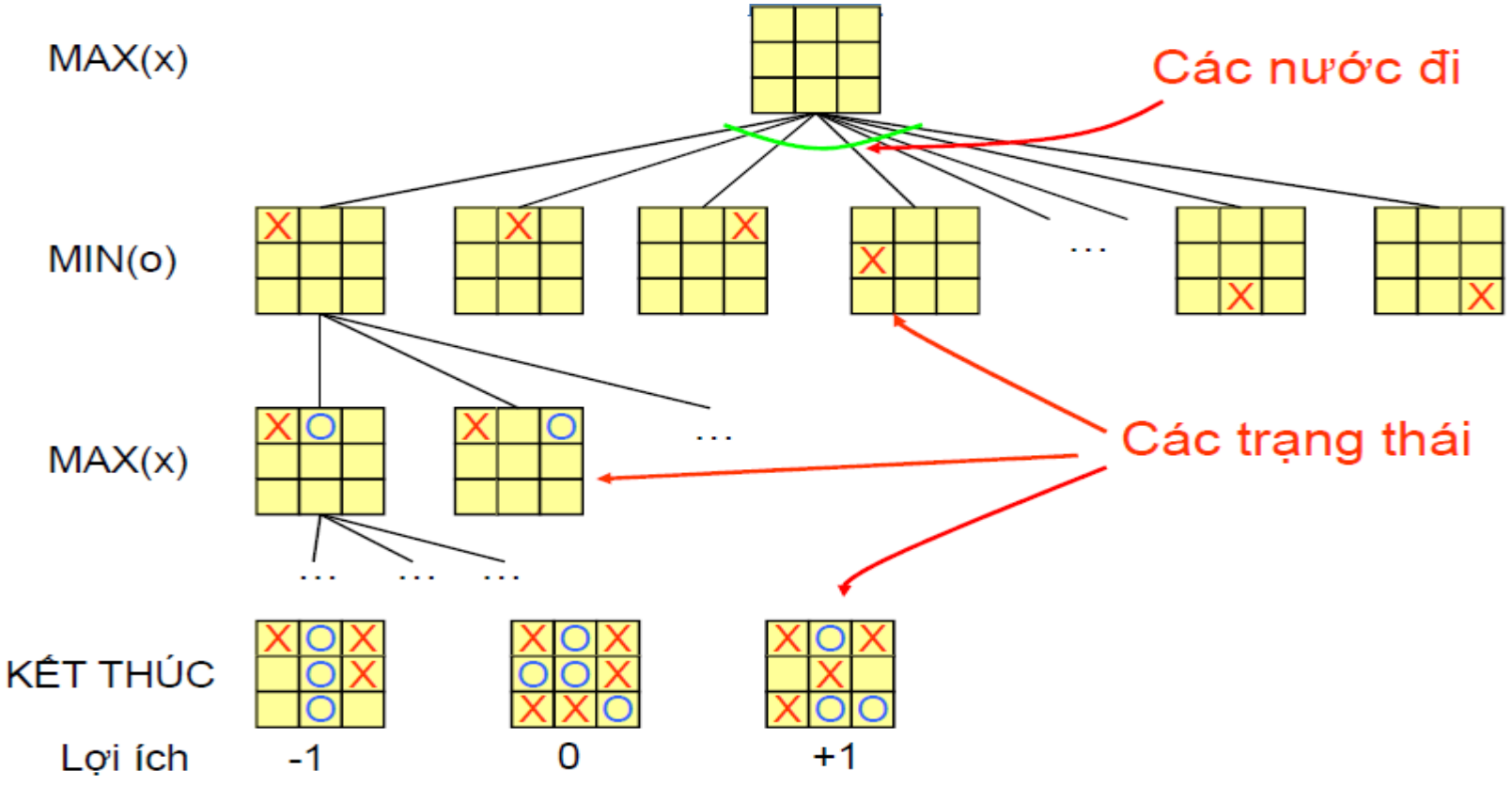
***Hàm Heuristic*: E(n) = M(n) – O(n)**

M(n) là *tổng số đường thẳng có thể thắng của MAX*

O(n) là *tổng số đường thẳng có thể thắng của MIN (đối thủ)*



TicTacToe



Chiến lược

- Đặc điểm
  - Hai bên luân phiên đi
  - Hai bên biết thông tin đầy đủ về nhau
  - Mỗi bên tìm kiếm nước đi tốt nhất
  - Nước đi tốt nhất là nước đi dẫn đến chiến thắng
  - Biểu diễn KGTT bằng: cây trò chơi.
- Thuật toán tiêu biểu: MINIMAX

Thuật toán MINIMAX

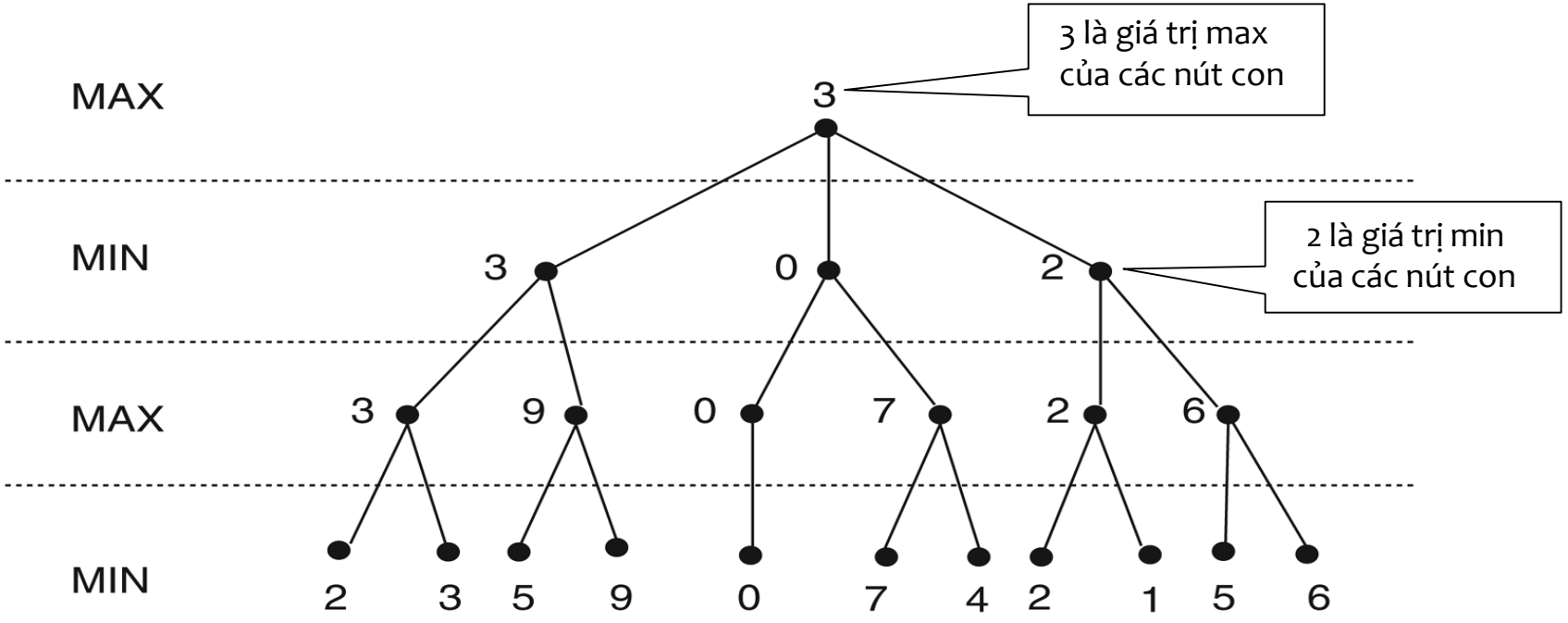
```
int minimax(node n, int d)// mức sâu d
   if leaf(n) or depth == 0 return evaluate(n)
   if n is a max node
      v = L
      for each child of n
         v' = minimax (child,d-1)
         if v' > v, v = v'
      return v
   if n is a min node
      v = W
      for each child of n
         v' = minimax (child,d-1)
         if v' < v, v = v'
      return v
```

Đánh giá thuật toán MINIMAX

- Đầy đủ? Có (nếu cây tìm kiếm là hữu hạn)
- Tối ưu? Có (với một đối thủ đối ưu)
- Độ phức tạp thời gian: $O(b^d)$;
- Độ phức tạp không gian: $O(b^d)$ *(b: số nhánh, d: chiều sâu)*
- Trò chời cờ vua: $b \approx 35$, $d \approx 100$ với một ván thông

  Thường –> không tìm được lời giải tối ưu

Minimax với độ sâu cố định



*Các nút lá được gán các giá trị lợi ích (**heuristic**) nào đó, các giá trị tại các nút trong là các giá trị nhận được dựa trên giải thuật Minimax (min hay max cua các nút con)*

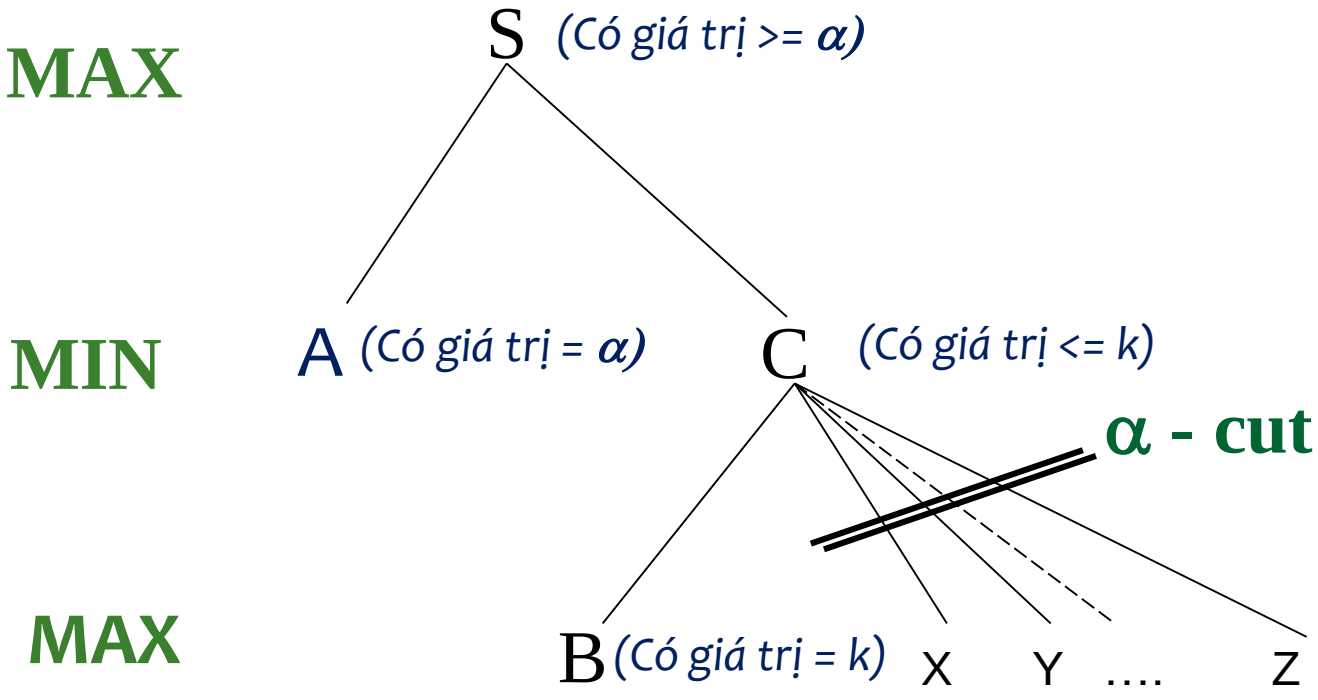
Cắt tỉa α–β (alpha-beta)

- làm thế nào để hạn chế khối lượng KGTK ngoài việc hạn chế số mức d đi vì số trạng thái vẫn còn quá lớn
- Cờ vua: nhân tố nhánh b=35; d=3 có 35*35*35=42.785 trạng thái
- Giảm bớt các trạng thái cần khảo sát mà vẫn không ảnh hưởng gì đến việc giải quyết bài toán
- ***Cắt tỉa các nhánh không cần khảo sát*** **(Cắt tỉa α–β)**

Chiến lược cắt tỉa α–β

- Tìm kiếm theo kiểu *depth-first*
- Nút MAX có 1 giá trị α (luôn tăng)
- Nút MIN có 1 giá trị β (luôn giảm)
- Tìm kiếm có thể kết thúc dưới bất kỳ
  - Nút MIN nào có $\beta \leq \alpha$ của bất kỳ nút cha MAX nào
  - Nút MAX nào có $\alpha \geq \beta$ của bất kỳ nút cha MIN nào
- Cắt tỉa α–β thể hiện *mối quan hệ giữa các nút ở mức* ***n*** *và* ***n+2***, mà tại đó toàn bộ cây có gốc tại mức ***n+1*** có thể cắt bỏ

Cắt α (vị trí MAX)

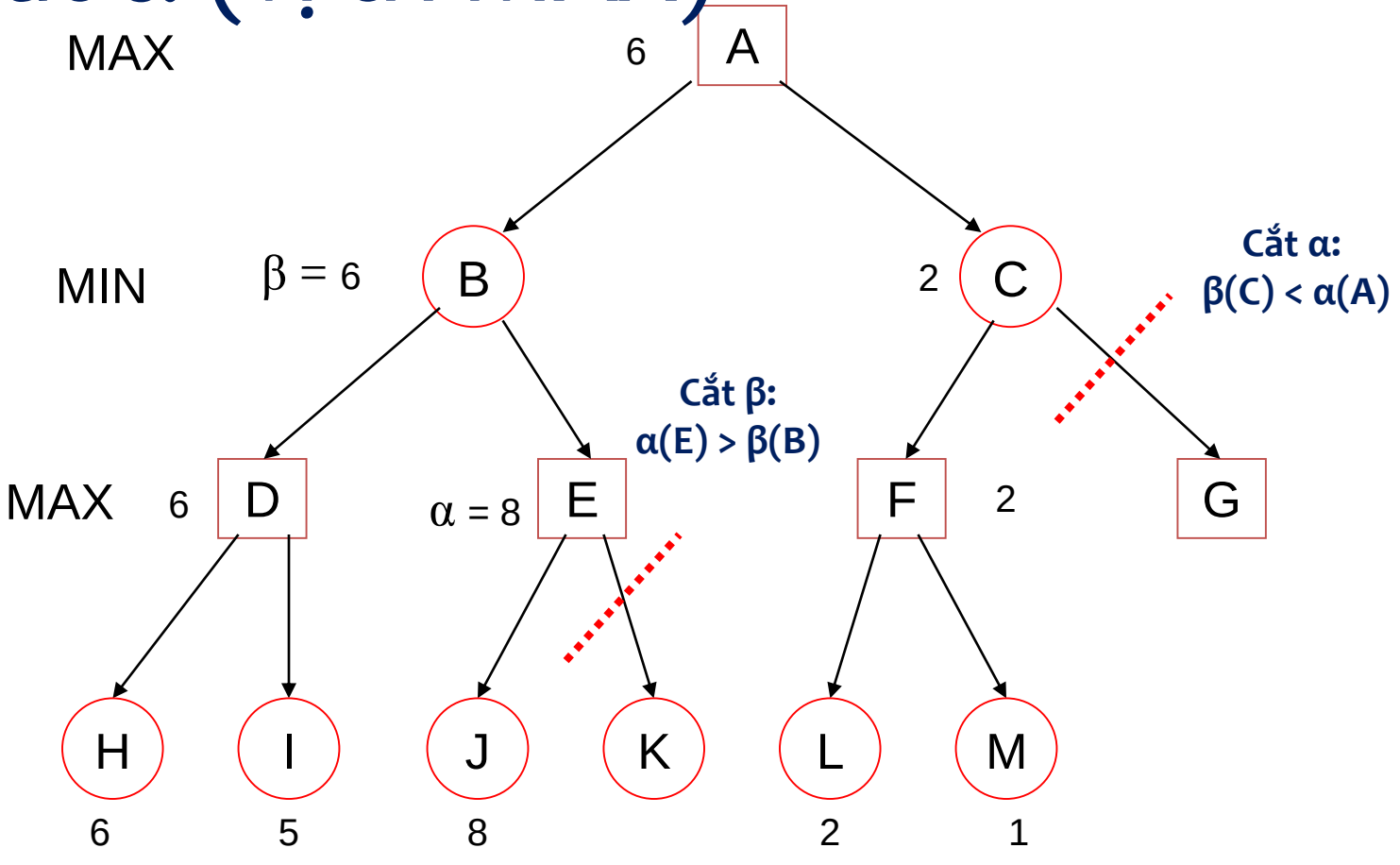


Điều kiện 1: Chỉ cần biết giá trị tại A và B

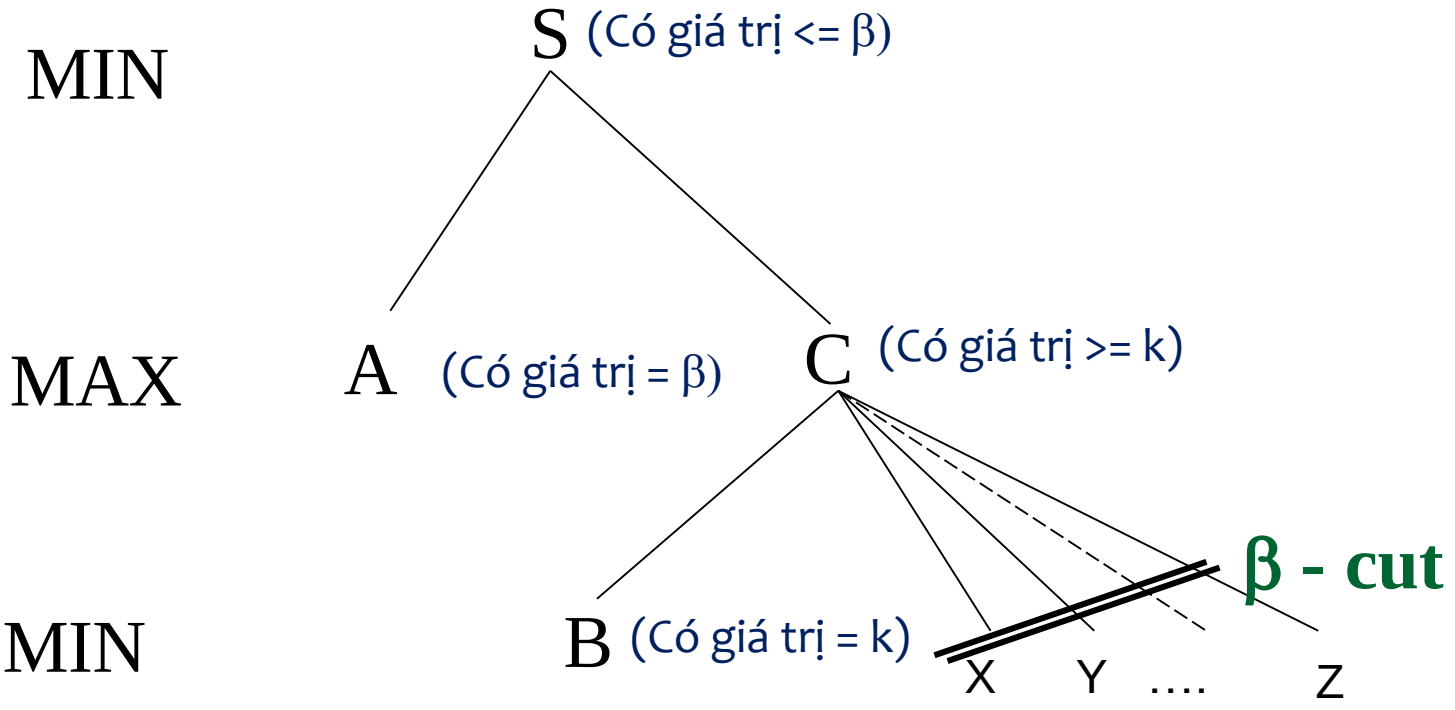
Điều kiện 2: Giá trị A > giá trị B

Điều kiện 3: X, Y, .. , Z ở vị trí Max - Bỏ những cây con có gốc là X,Y,… , Z

Cắt α (vị trí MAX)



Cắt β (vị trí Min)

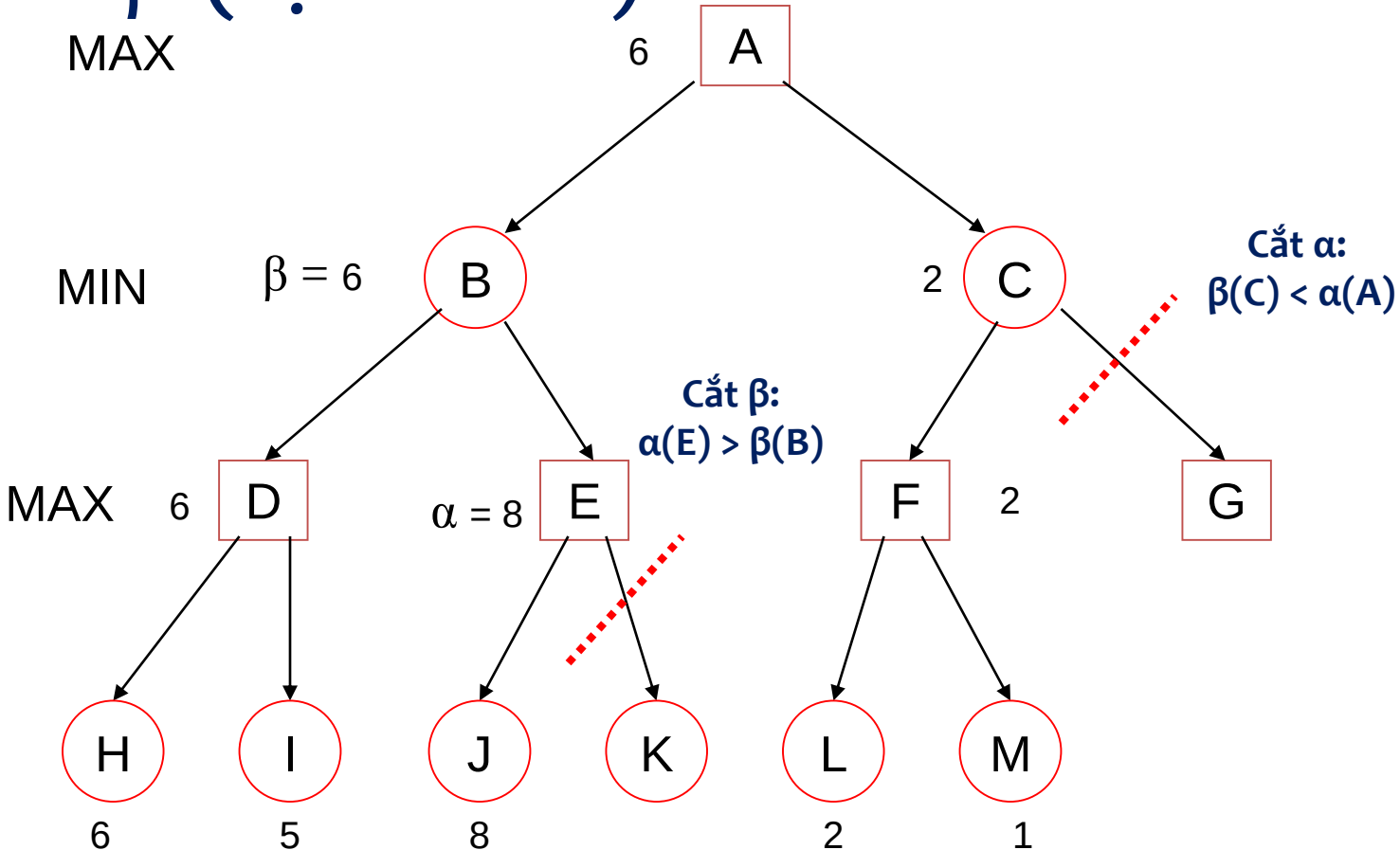


Điều kiện 1: Chỉ cần biết giá trị tại A và B

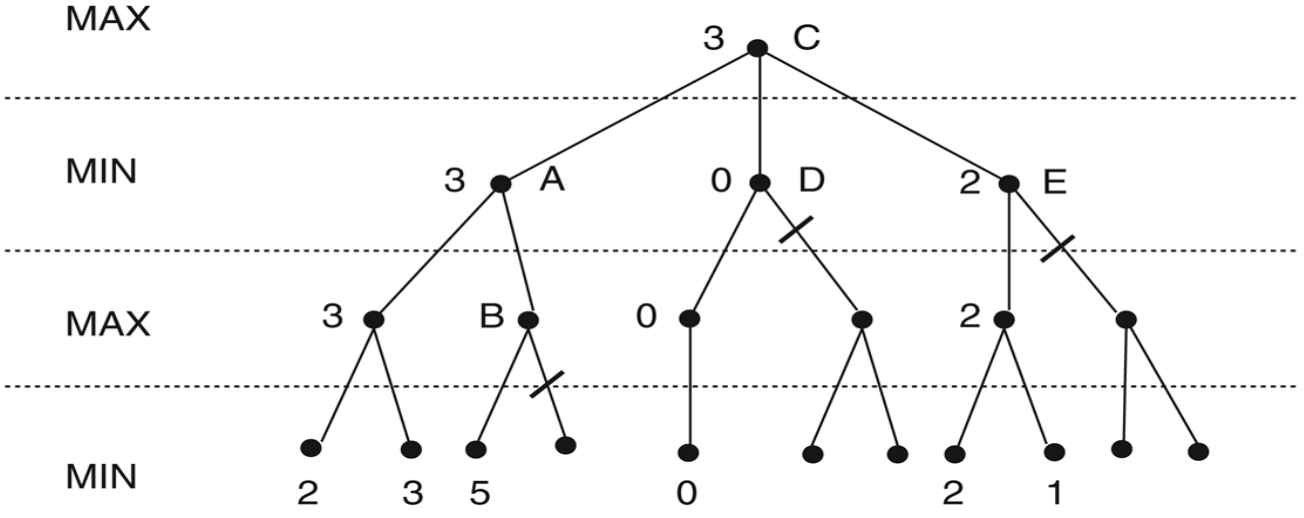
Điều kiện 2: Giá trị A < giá trị B

Điều kiện 3: X, Y, .. , Z ở vị trí Min - Bỏ những cây con có gốc là X,Y,… , Z

Cắt β (vị trí Min)



Ví dụ: Cắt tỉa α–β



A has $\beta = 3$ (A will be no larger than 3)
B is β pruned, since 5 > 3
C has $\alpha = 3$ (C will be no smaller than 3)
D is α pruned, since 0 < 3
E is α pruned, since 2 < 3
C is 3

Thuật toán α–β: *Pseudo code*

```
function minimax(int node, depth,BOOL isMaximizingPlayer, int alpha, int beta)
 {   if node is a leaf node
    {    return value of the node
        if isMaximizingPlayer :
        bestVal = -INFINITY(-∞)
        for each child node :
        {         value = minimax(node, depth+1, false, alpha, beta)
            bestVal = max( bestVal, value)
            alpha = max( alpha, bestVal)
            if beta <= alpha:
                break
        }return bestVal
    }
    else
    {    bestVal = +INFINITY(+∞)
         for each child node :
            value = minimax(node, depth+1, true, alpha, beta)
            bestVal = min( bestVal, value)
            beta = min( beta, bestVal)
            if beta <= alpha:
                break
     } return bestVal
}
```

Thuật toán α–β: *Code CPP*

```
int minimax(int depth, int nodeIndex, bool maximizingPlayer, int values[], int alpha, int beta)
{    // Terminating condition. i.e leaf node is reached
    if (depth == 3)
        return values[nodeIndex];
     if (maximizingPlayer)
    {       int best = MIN;
            // Recur for left and right children
            for (int i=0; i<2; i++)
            {
                        int val = minimax(depth+1, nodeIndex*2+i, false, values, alpha, beta);
                        best = max(best, val);
                        alpha = max(alpha, best);
                        // Alpha Beta Pruning
                        if (beta <= alpha)
                        break;
            }
            return best;
    }
    else
    {       int best = MAX;
         // Recur for left and right children
            for (int i=0; i<2; i++)
            {           int val = minimax(depth+1, nodeIndex*2+i, true, values, alpha, beta);
                        best = min(best, val);
                        beta = min(beta, best);
                        // Alpha Beta Pruning
                        if (beta <= alpha)
                        break;
            }
            return best;
    }
}
```

Thuật toán α–β: *code cpp*

```
// Driver Code
int main()
{
    int values[8] = { 3, 5, 6, 9, 1, 2, 0, -1 };
    cout << "The optimal value is : "
         << minimax(0, 0, true, values, MIN, MAX);;
    return 0;
}
```

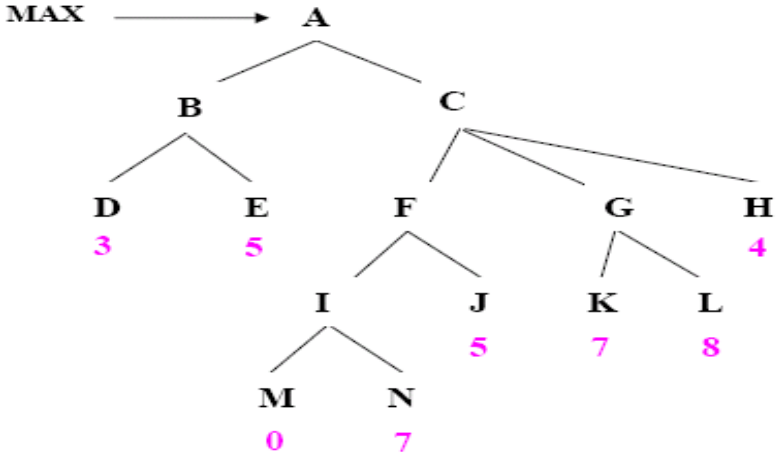
Links reference

- Giải thuật MiniMax (GS Patrick Henry Winston, MIT)
  - http://www.ai.mit.edu/courses/6.034f/gamepair.html
- Giải thuật MiniMax với cắt tỉa α–β (GS Patrick Henry Winston, MIT)
  - http://www.ai.mit.edu/courses/6.034f/searchpair.html
  - https://www.cs.cornell.edu/courses/cs312/2002sp/lectures/rec21.htm
  - http://homepage.ufp.pt/jtorres/ensino/ia/alfabeta.html
  - http://inst.eecs.berkeley.edu/~cs61b/fa14/ta-materials/apps/ab_tree_practice/
  - http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html

Bài tập: bài 1 (minimax)

Liệt kê danh sách các nút được duyệt theo tìm kiếm DFS.

- Thực hiện giải thuật Minimax trên cây.



- Sẽ có gì khác biệt nếu như ta dùng giải thuật cắt tỉa alpha – beta để định trị nút gốc cho cây?