06 - Adversarial Search

Adversarial Game

- Zero-Sum games:
  - Đối kháng hoàn toàn
  - Chỉ 1 trong 2 người thắng, không hòa
- **Gía trị trạng thái**: kết quả tốt nhất mình có thể đạt được nếu mình xuất phát từ trạng thái đó
- Adversarial game tree:
  - Đối thủ (min player) luôn muốn mình thua → min giá trị trạng thái
  - Bản thân (max player) luôn muốn thắng → max giá trị trạng thái

Adversarial Search (Minimax)

- Gía trị minimax: là giá trị tối đa có thể nhận được khi đang đối đầu với đối thủ không nhượng bộ

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor))
    return v
```

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

max player

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successor))
    return v
```

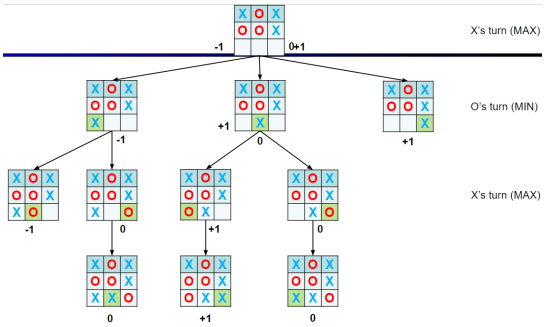
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

min player

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

⇒ Đệ quy tương hỗ, chạy tương tự DFS

- Time: $O(b^m)$
- Space: $O(bm)$



⇒ Tốn tài nguyên tính toán, tìm kiếm do bắt buộc vét cạn cây

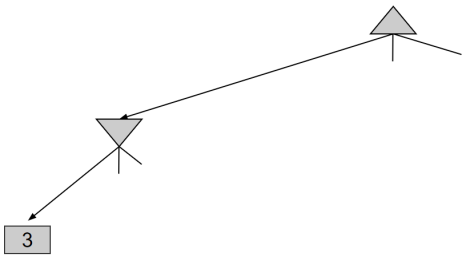
- Gỉa sử đến 1 lúc nào đó đối thủ không đánh theo ý mình → vẫn sẽ thắng, vì thuật toán đặt trường hợp đối thủ không nhượng bộ

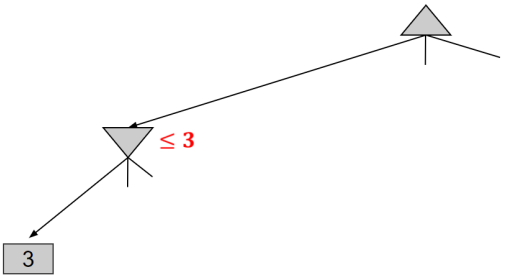
Nhận xét Minimax

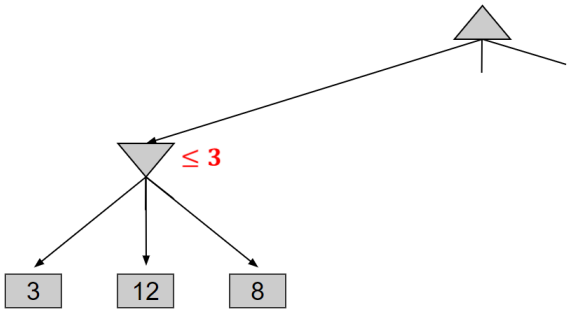
- Bởi vì minimax đặt tình huống đối thủ không nhượng bộ, nên nó sẽ đi theo phương án an toàn nhất → max player khi nghĩ đối thủ tìm cách tiêu diệt mình và không biết đối thủ sẽ làm gì, để giảm thiểu số điểm mất đi, max player sẽ tự sát
- 100 lần chơi thì cả 100 lần max player sẽ tự sát

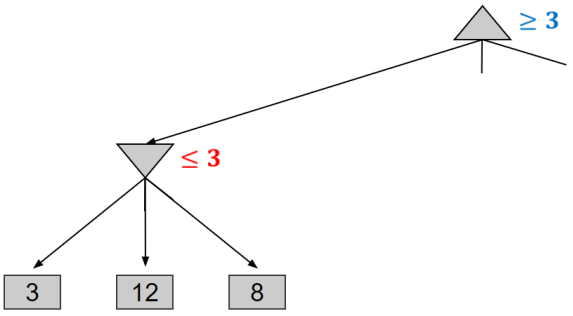
Alpha Beta Pruning

- Ví dụ 1:

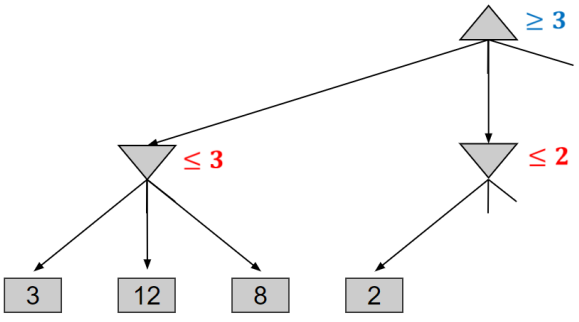




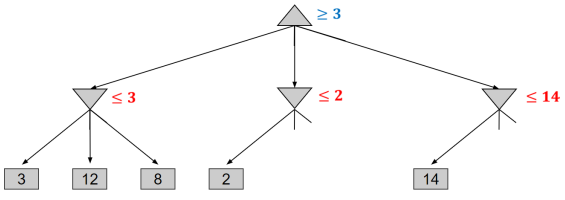




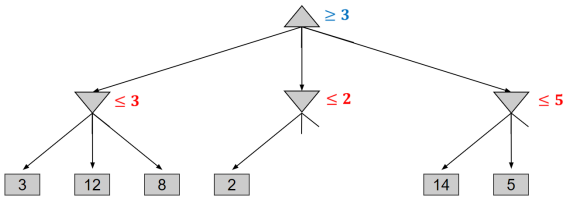
nhánh max ≥3 → tìm xem có nhánh con nào > 3 hay không

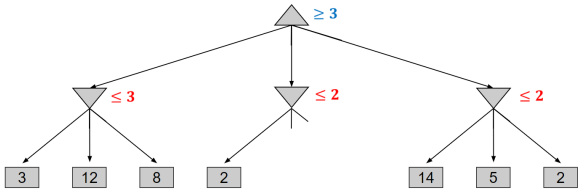


Vì nhánh min2 ≤ 2 mà max đã luôn ≥ 3 nên toàn bộ con của min2 sẽ bị bỏ



Vì min3 ≤ 14 lớn hơn max → tiếp tục xét con của min 3





- Thuật toán:
  - node gốc là max → $v = -\inf$; node gốc là min → $v = \inf$
  - max player → cập nhập α; min player → cập nhập β
  - v truyền từ dưới lên; α, β truyền từ trên xuống

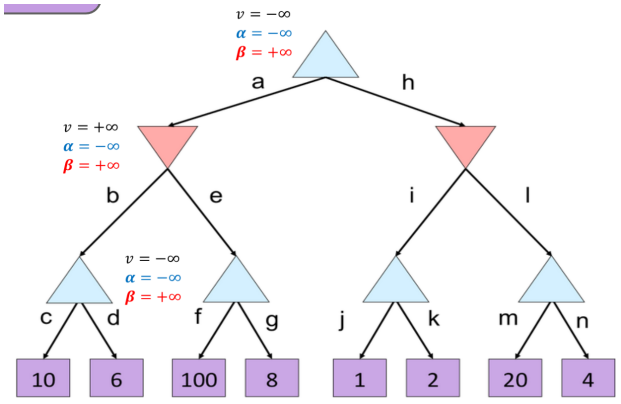
α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

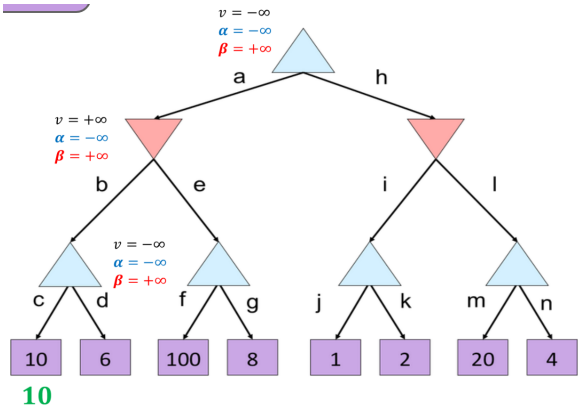
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

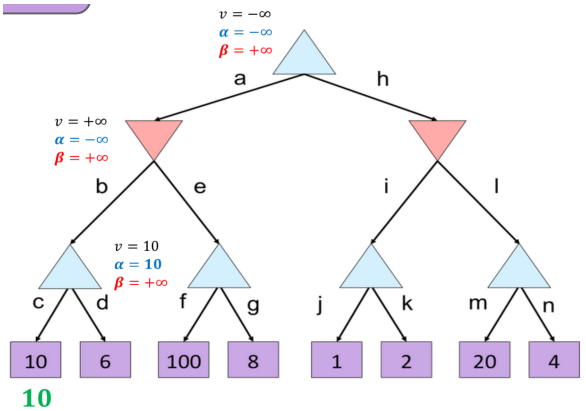
```
def value(state, α, β):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state, α, β)
    if the next agent is MIN: return min-value(state, α, β)
```

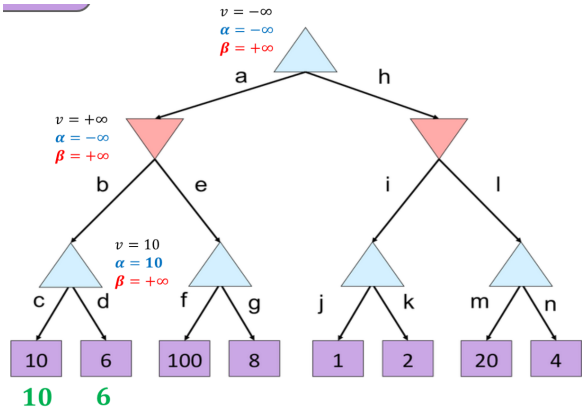
- Ví dụ chạy thuật toán:



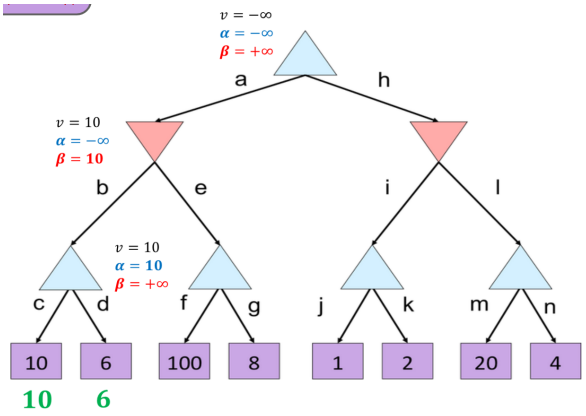
ko phải node kết thúc nên đi xuống tiếp



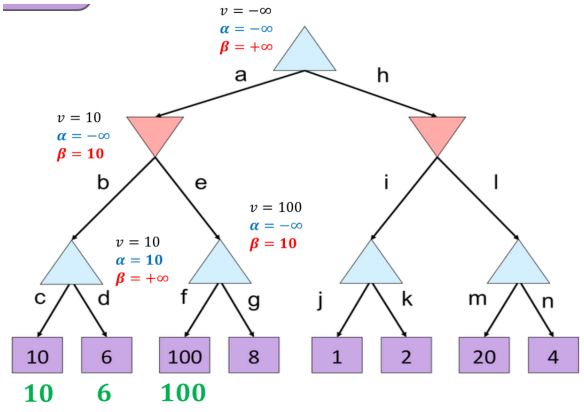


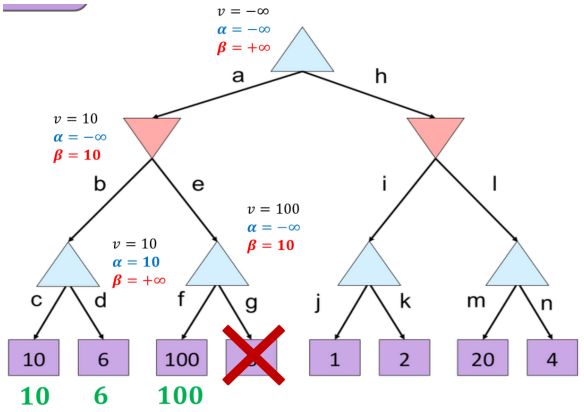


xem nhánh **d** có tốt hơn 10 hay ko

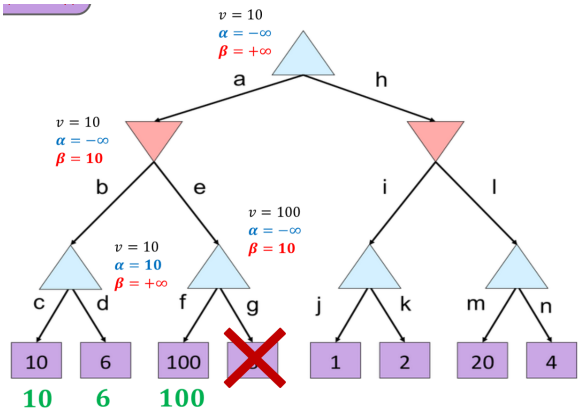


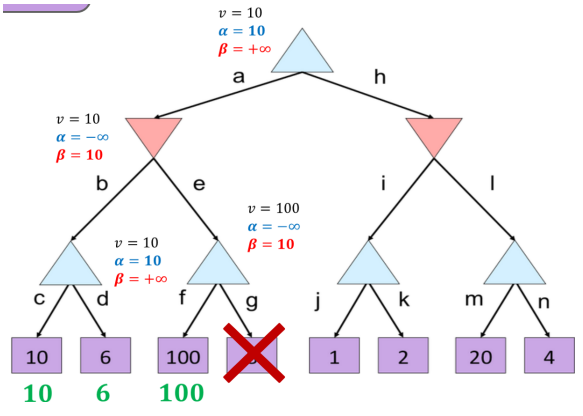
min1 xuống nhánh e xem có nhánh nào < **beta** không

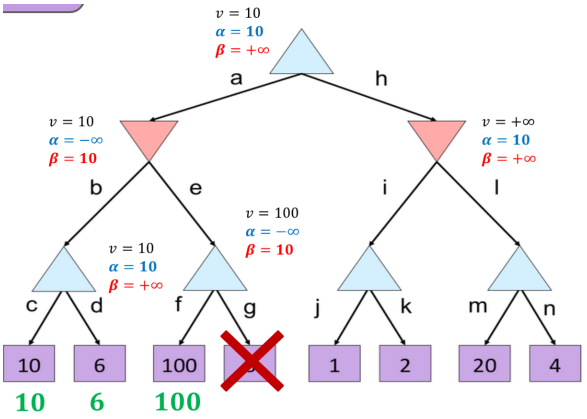




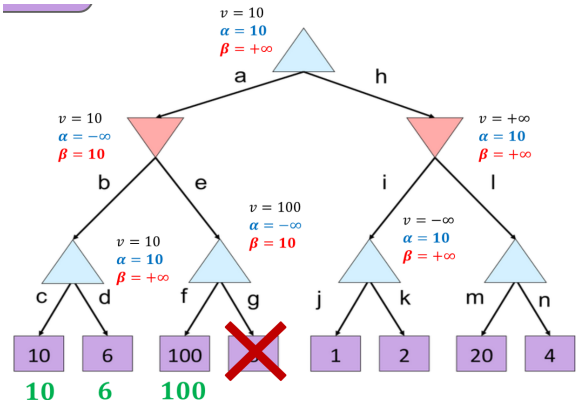
(nhánh e) v = 100 > **beta** → vì max2 sẽ chọn số > 100 mà min1 sẽ kbh chọn > **beta**→ loại nhánh **g**



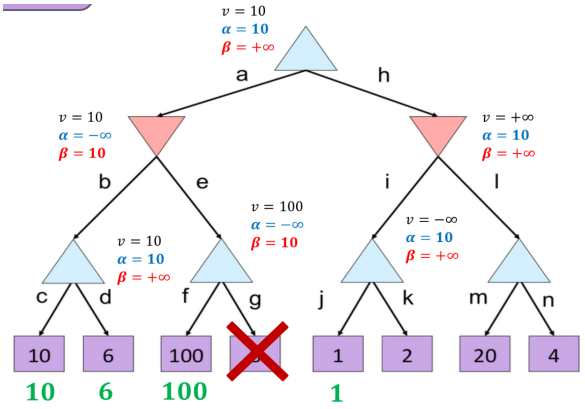


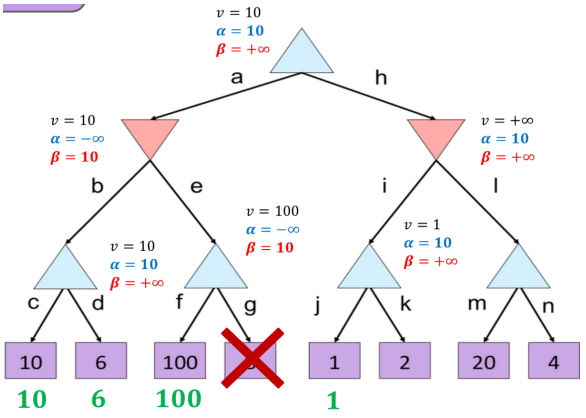


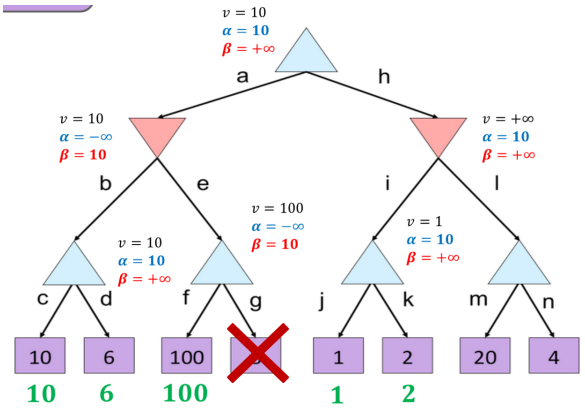
min1 hạ **alpha** từ max root xuống → lấy giá trị nào bé hơn **alpha**



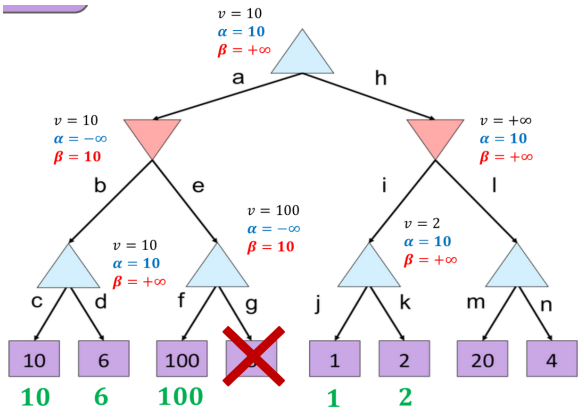
hạ alpha từ min1 → lấy giá trị nào lớn hơn **alpha**



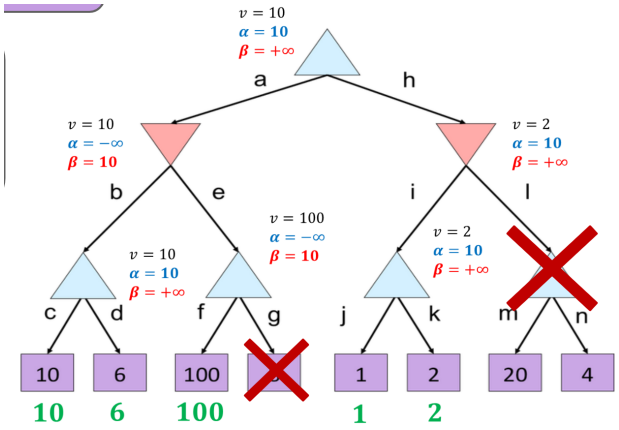




v=1 < beta nên tiếp tục chạy tới khi nào hết node con



v=2 > v=1 nên nhận max3 v=2



v=2 < **alpha (nhánh h)** → max root sẽ kbh chọn **h**

Nhận xét

- Thứ tự node con ảnh hưởng tới cắt tỉa cây
- Cắt tỉa không làm thay đổi kết quả trò chơi: giá trị node gốc không thay đổi nhưng *giá trị node trung gian thay đổi*
- Độ phức tạp thời gian: $O(b^{\frac{m}{2}})$ ⇒ Vẫn không chạy nổi

Resources Limit

- Trong các trò chơi thực tế, không thể nào tìm xuống tầng lá

Solution: Depth-limited Search

- Cho phép nhìn trước số nước đi cụ thể tùy thuộc vào phần cứng máy
- Sử dụng 1 **hàm lượng giá** để ước lượng giá trị của những trạng thái không là terminate state → tính minimax dựa vào giá trị hàm lượng giá này
- Đối thủ có thể có 1 hàm ước lượng khác → phải chờ đối thủ thực hiện hành động → mới quyết định tiếp tục nhìn trước **x** bước dựa trên hành động đối phương
- Khi đối thủ thực hiện hành động khác với dự tính ⇒ chưa chắc mình sẽ giành chiến thắng, vì đây chỉ là **giá trị ước lượng** chứ không phải giá trị minimax
- Tính tối ưu giảm: không chắc chắn sẽ tìm được đường đi chiến thắng
  - Số lượng bước đi được nhìn trước ảnh hưởng đến kết quả (2 bước không tìm được đường sống → tự sát, 10 bước → tìm được đường chiến thắng)

Evaluation Functions

- Dùng các đặc trưng phi tuyến tính (số lượng food, khoảng cách pacman và ghost, ...) khiến hàm mạnh hơn các đặc trưng tuyến tính
- Điều chỉnh trọng số