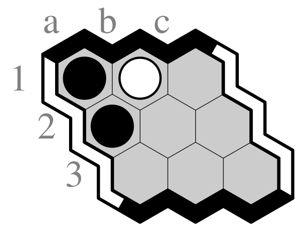
cmput 396 mcts example revised 2018-11-20

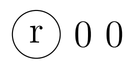
```
class Node:
  def __init__(self, m, p): # move is from parent to node
    self.move, self.parent, self.children = m, p, []
    self.wins, self.visits  = 0, 0

  def expand_node(self, state):
    if not terminal(state):
      for each non-isomorphic legal move m of state:
        nc = Node(m, self) # new child node
        self.children.append(nc)

  def update(self, r):
    self.visits += 1
    if r==win: self.wins += 1

  def is_leaf(self):
    return len(self.children)==0

  def has_parent(self):
    return self.parent is not None

def mcts(state):
  root_node  = Node(None, None)
  while time remains:
    n, s = root_node, copy.deepcopy(state)
    while not n.is_leaf():    # select leaf
      n = tree_policy_child(n)
      s.addmove(n.move)
    n.expand_node(s)          # expand
    n = tree_policy_child(n)
    while not terminal(s):    # simulate
      s = simulation_policy_child(s)
    result = evaluate(s)
    while n.has_parent():     # propagate
      n.update(result)
      n = n.parent

return best_move(tree)
```

mcts example: this hex position, white to play

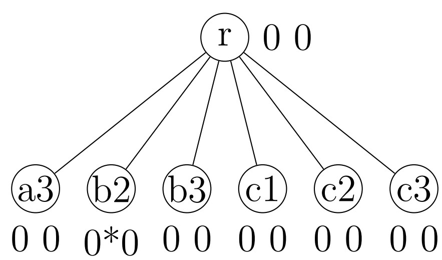


iteration 1

- **select leaf**

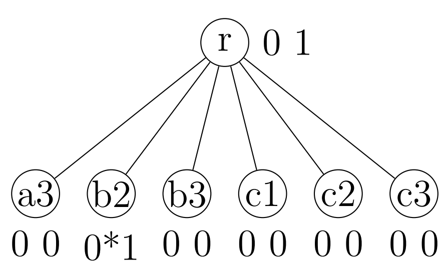


- **expand-leaf, pick-best-child (say b2)**



- **simulate from state r-b2 (say b[c1] w[c3] b[a3] ! black win)**

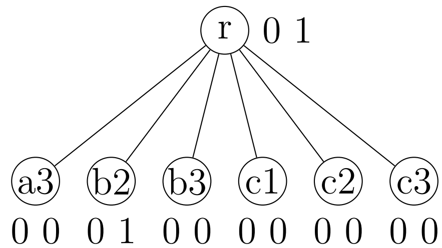
- **back-propagate**



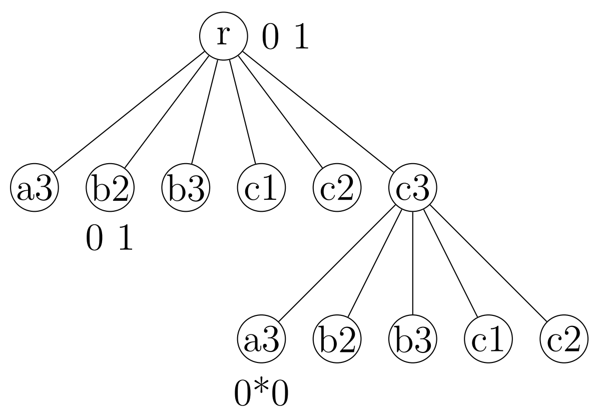
win,visit counts are for root player (white)

iteration 2

- **select leaf (repeat pick-best-child)**

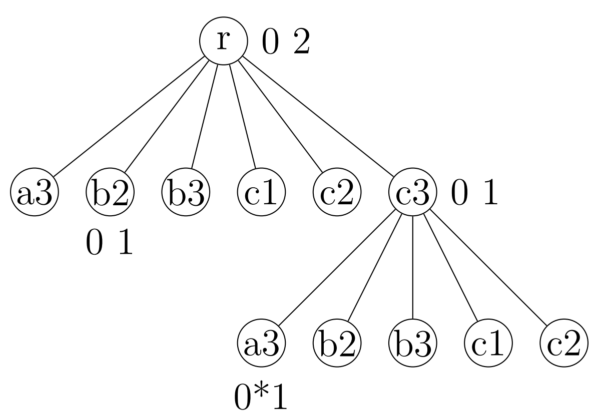


- **expand-leaf, pick-best-child (say a3)**



- **simulate from r-c3-a3 (say black win)**

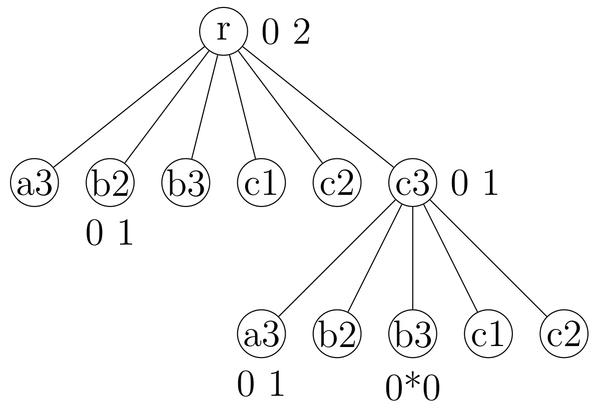
- **back-propagate**



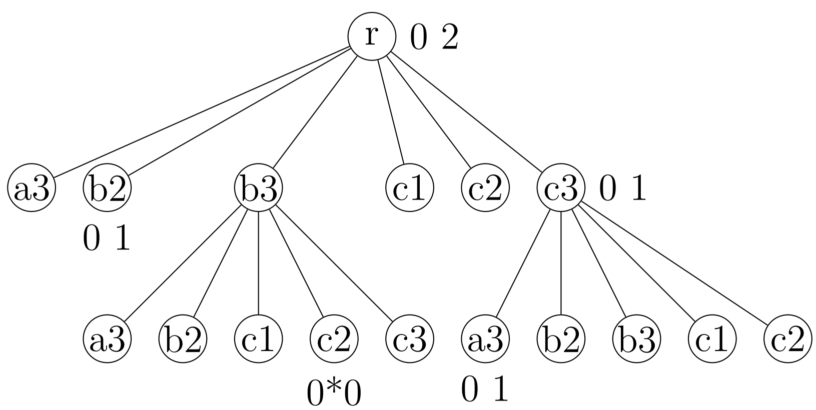
unlabelled nodes are all 0 0

iteration 3

- **select leaf (repeat pick-best-child)**

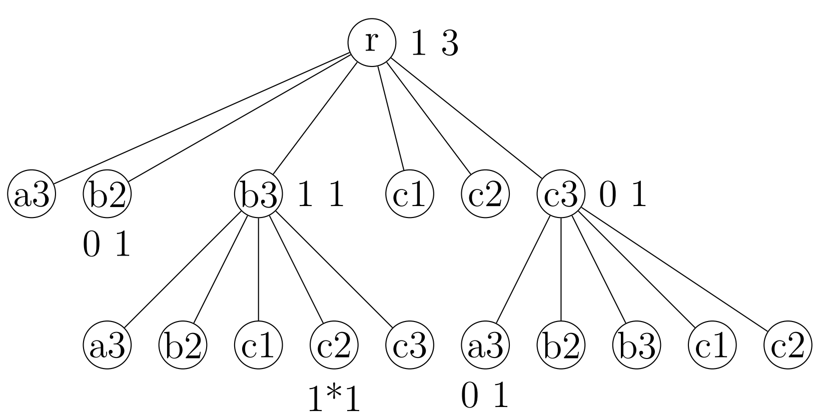


- **expand leaf, pick-best-child (say c2)**



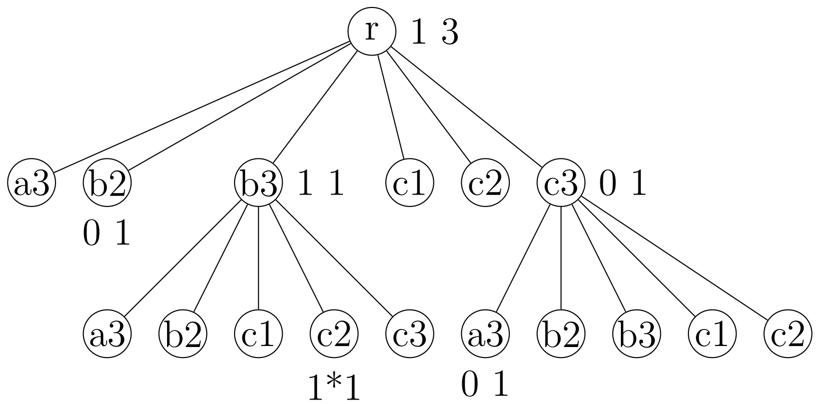
- **simulate from r-b3-c2 (say white win)**

- **back-propagate**

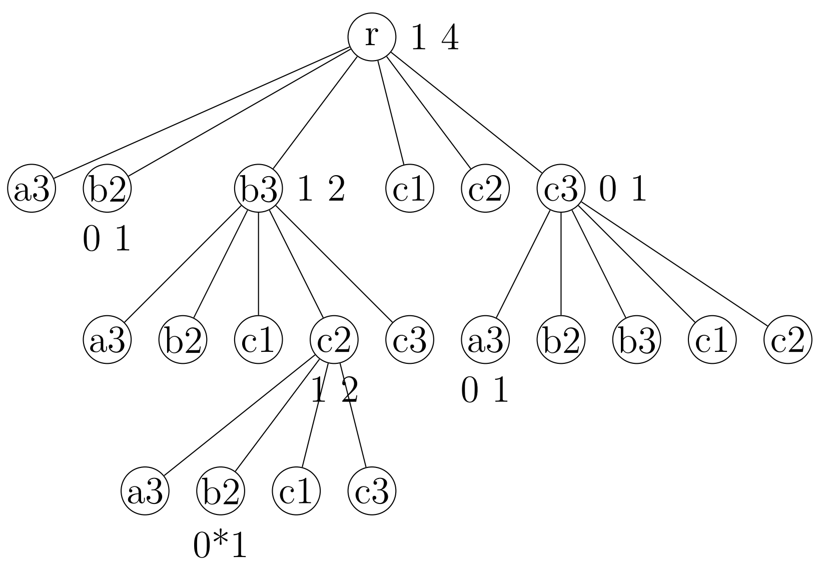


iteration 4

- select leaf (repeat pick-best-child)



- expand-leaf (r-b3-c2), pick-best-child (say b2)
- simulate from r-b3-c2-b2 (say black win)
- back-propagate



How should we compute the win rate of a node with no visits?

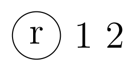
We prefer 0 0 (wins/visits) to 0 1, because nothing could be worse than losing all simulations. And we prefer 1 1 to 0 0, because nothing could be better than winning all simulations.

One way to implement this is to initialize all new nodes with T wins and 2T visits for some integer T.

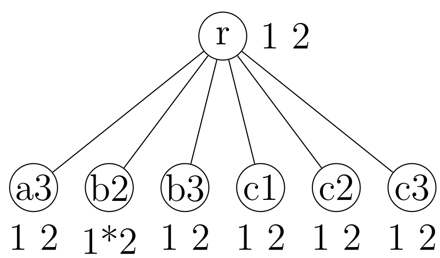
Let's repeat this example using this initialization.

iteration 1

- **select leaf**

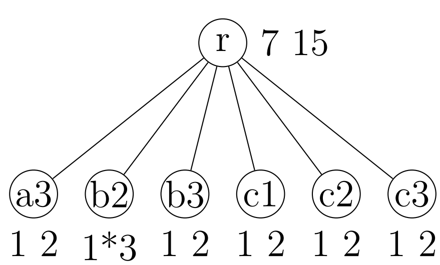


- **expand-leaf, pick-best-child (say b2)**



- **simulate from state r-b2 (say b[c1] w[c3] b[a3] ! black win)**

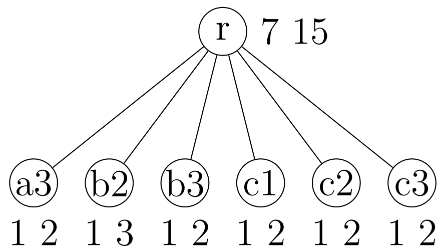
- **back-propagate**



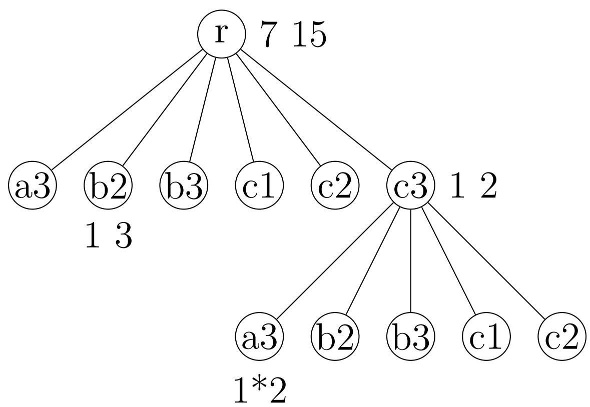
win,visit counts are for root player (white)

iteration 2

- **select leaf (repeat pick-best-child)**

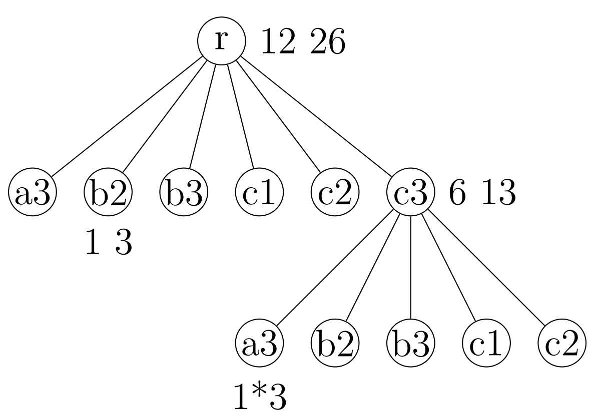


- **expand-leaf, pick-best-child (say a3)**



- **simulate from r-c3-a3 (say black win)**

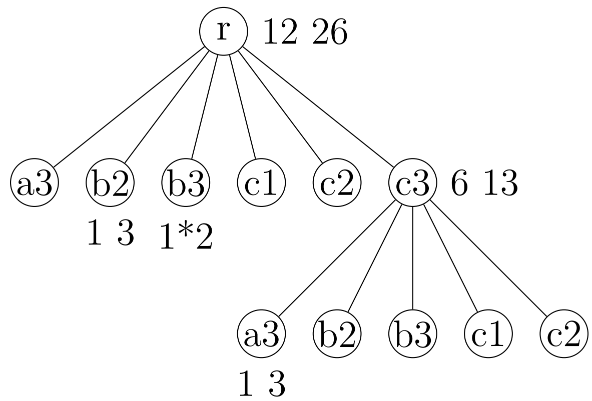
- **back-propagate**



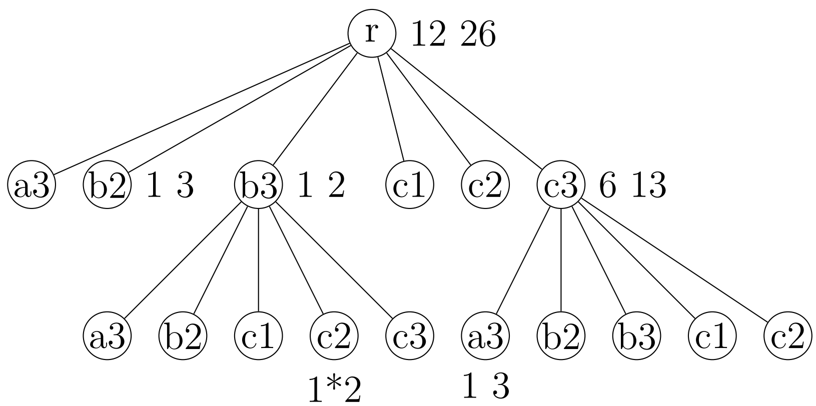
unlabelled nodes are all 1 2

iteration 3

- **select leaf (repeat pick-best-child)**

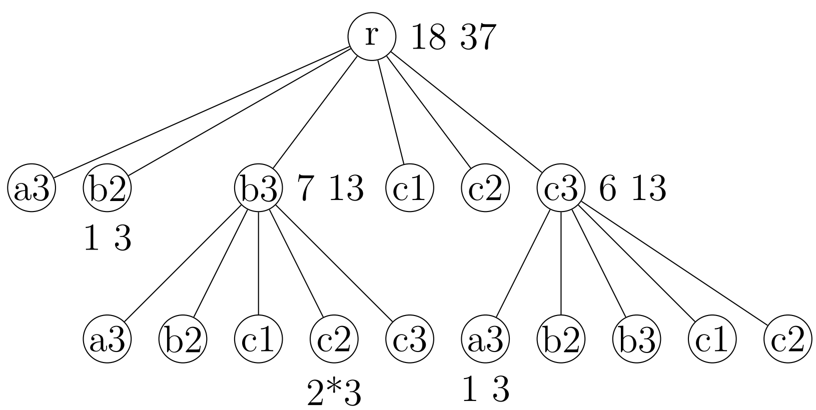


- **expand leaf, pick-best-child (say c2)**



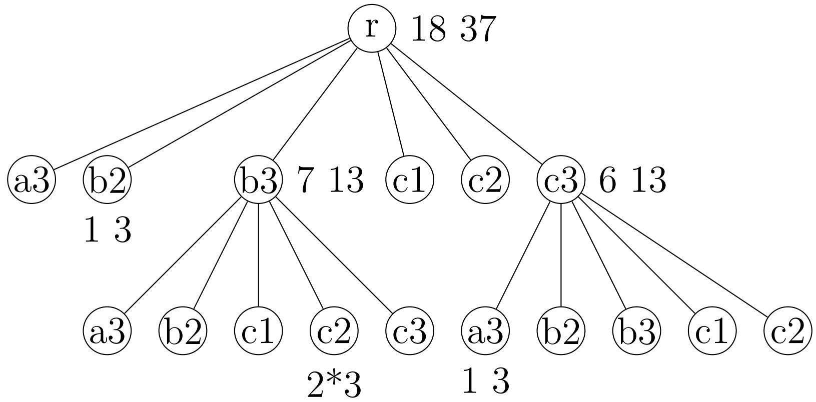
- **simulate from r-b3-c2 (say white win)**

- **back-propagate**

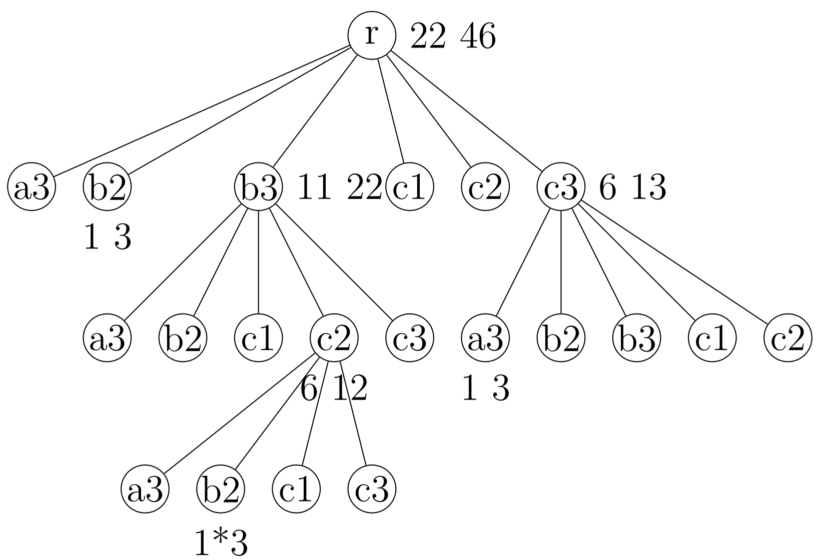


iteration 4

- **select leaf (repeat pick-best-child)**



- **expand-leaf (r-b3-c2), pick-best-child (say b2)**
- **simulate from r-b3-c2-b2 (say black win)**
- **back-propagate**



questions to think about

- trace another iteration of this example
- how close is the current tree to finding the best move? or to finding the correct win rate?
- how would you improve the performance of mcts if you were writing a hex player? or a go player?