Theory of Computer Game Hw2 資工所碩一 R05922068 彭宇劭

source code: OTP.h / Tree.h / TreeNode.h / search.cpp / board.h

執行方式：

make all / make judge / make test

檔案說明：

- OTP.h　大部分的code都寫在此，負責做主要的simulation操作
- Tree.h　維護一棵UCT產生的樹，其中由nowPtr在不同結點移動操作
- TreeNode.h　定義每個node所包含的資訊，下面會對數的結構做詳細說明

文字說明：

- 當接收到genmove的訊號時，會開始UCT搜尋
- 可以先利用目前的盤面呼叫B.get_valid_move獲得合法步並儲存在ML array裡面，當目前沒有任何合法步時直接return 64(pass)，只有一個合法步時，也直接走該步

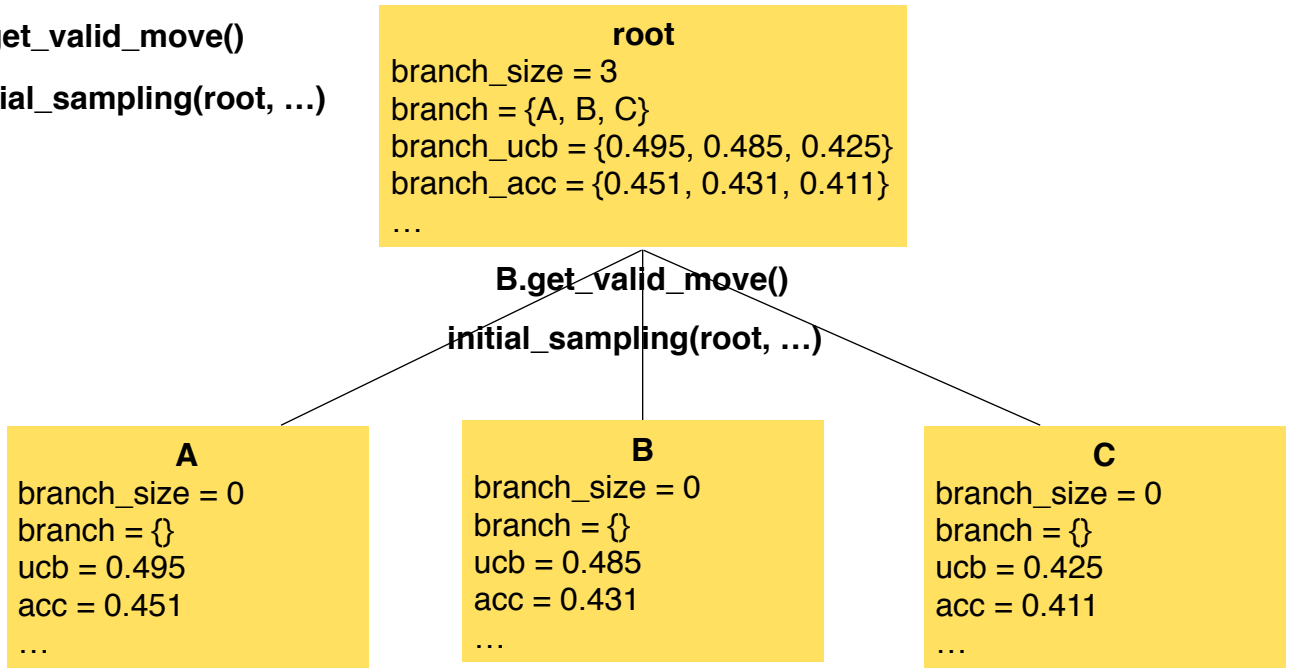
```
int ML[64],*MLED(B.get_valid_move(ML)), t_ML[64];
if (MLED==ML)
    return 64;
else if (*MLED==*(ML+1))
    return *(ML);
```

- 若有兩步以上就進入UCT search，先種下一棵樹，並初始化root，存入目前的盤面以及自己是什麼顏色的棋子。
- 初始完樹根之後開始要做 simulation，並適時的expansion，這裡主要分為五步驟
  - 取得目前盤面的valid move
  - 呼叫initial_sampling function做expansion，這邊會同時計算mean std，並利用同一層每個node之前的mean std關係做progressive pruning
  - 呼叫re_simulation根據UCB分數繼續做simulation數次（re_simu_size）
  - 將所增加的勝敗次數累計，並向root的方向一步步backtrace更新勝敗次數
  - 再來從root開始向下走，選擇最高的node前進，產生出一條PV，若遇到還沒有被expansion的node，就對該點做上述的initial_sampling和re_simulation，做完之後再backtrace回去並更新，最多expansion 變數depth次（除了可以控制expansion的數量，根據expansion的深度遞減，搜到越深的地方simulation的次數越少）
- 經過一連串的simulation後，回到root，選擇Accuracy最高的點移動。

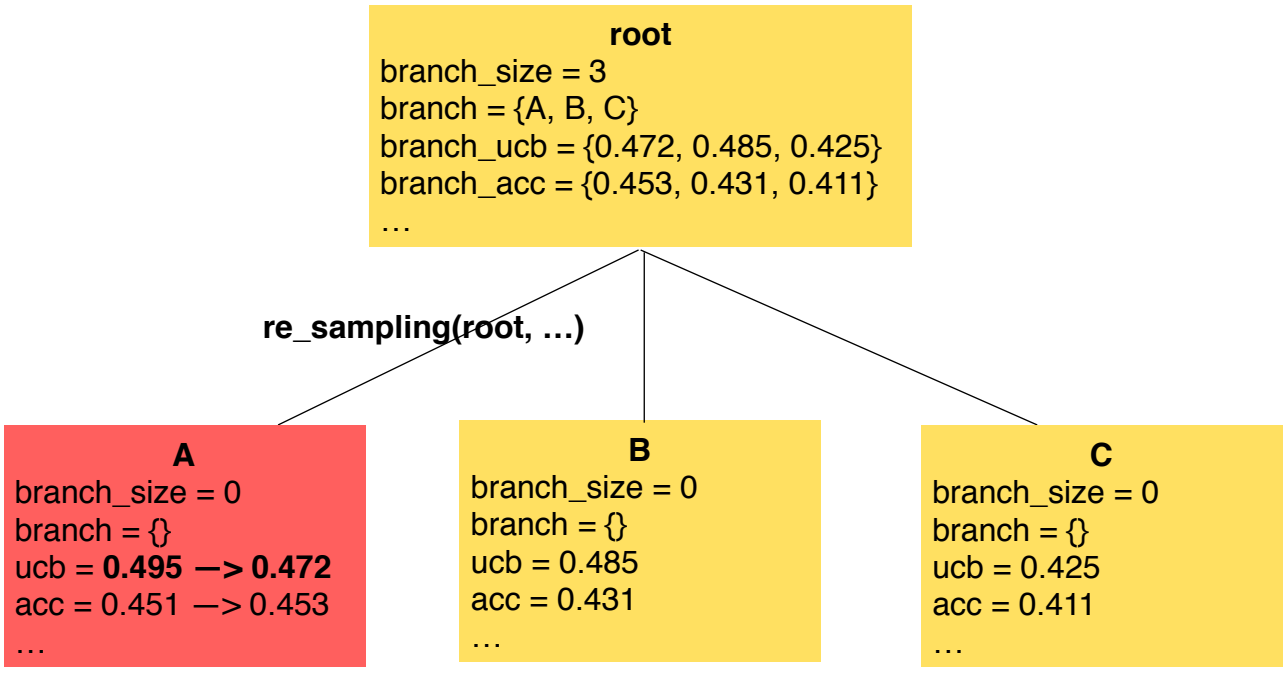
圖像說明：

B.get_valid_move()

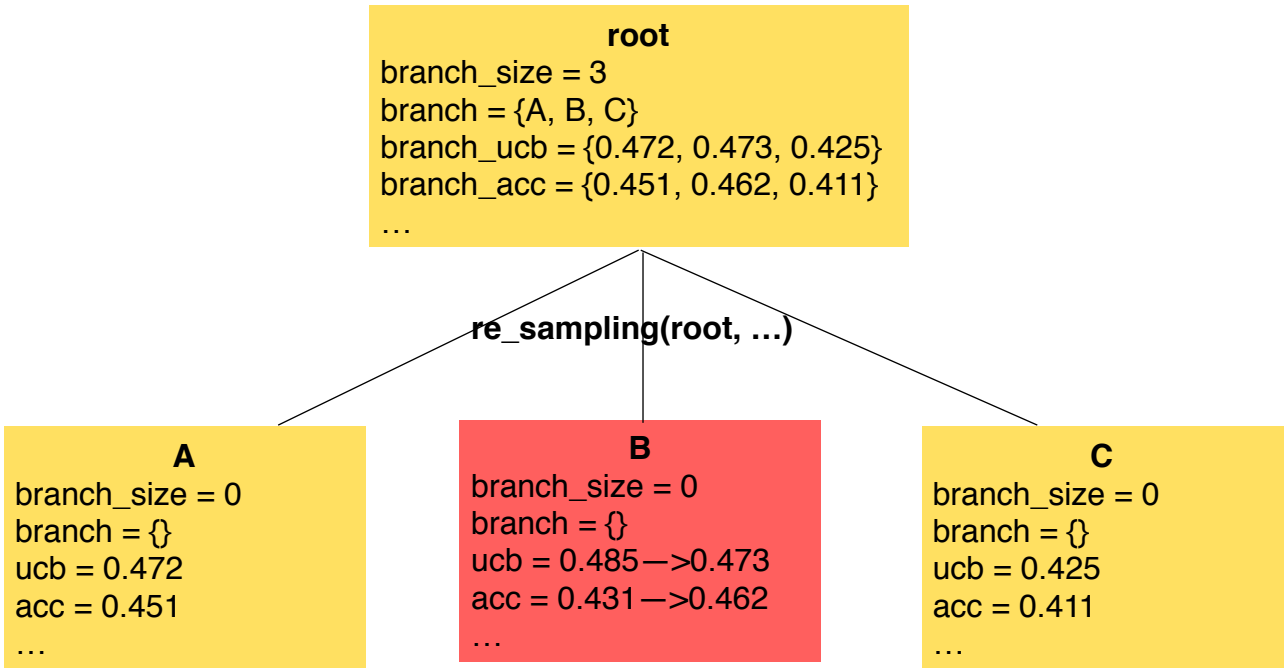
initial_sampling(root, …)



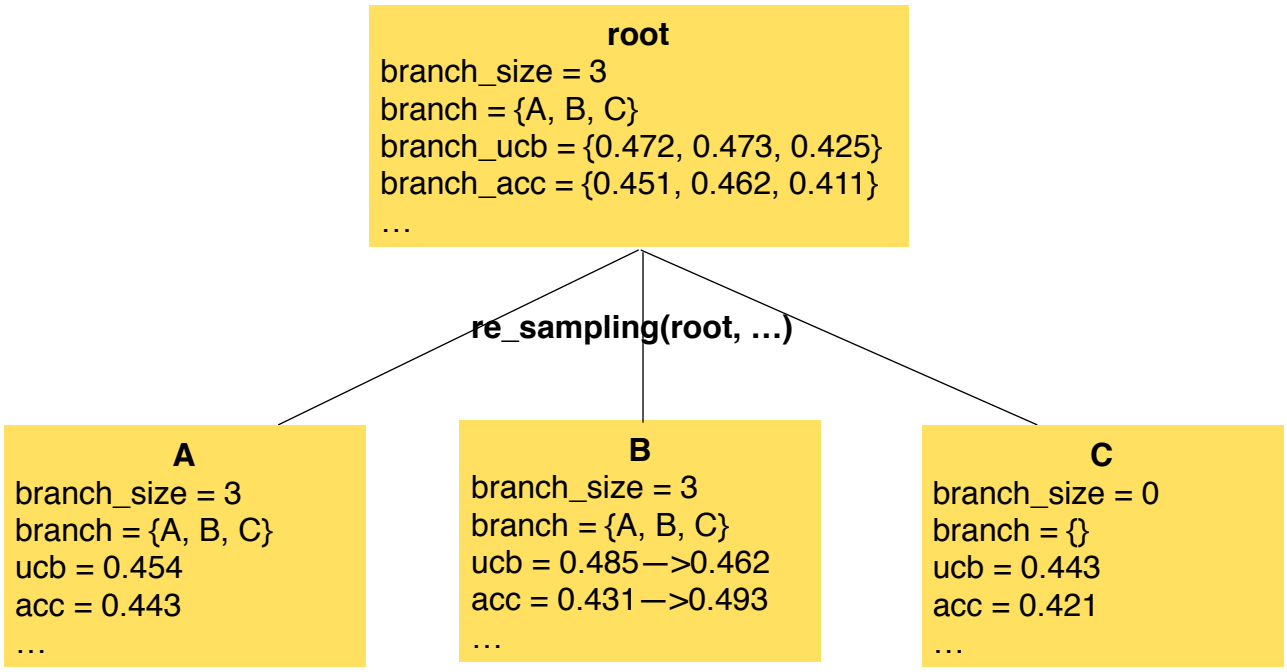
re_sampling(root, …) 1st



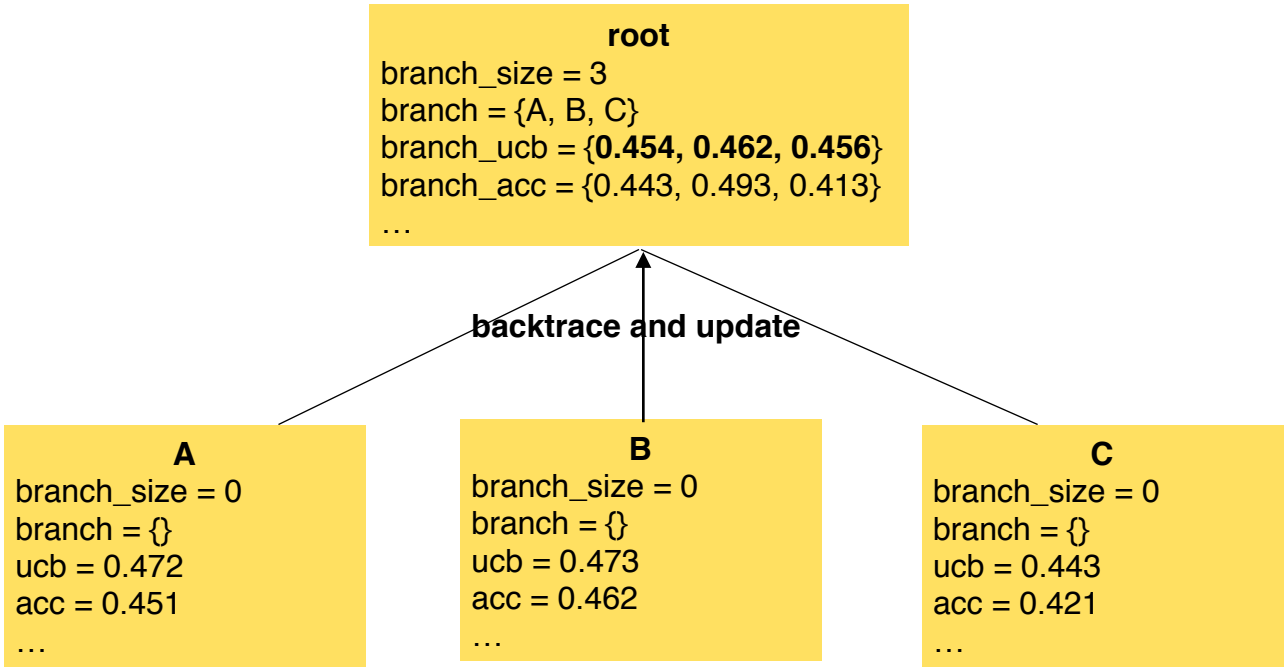
re_sampling(root, …) 2st



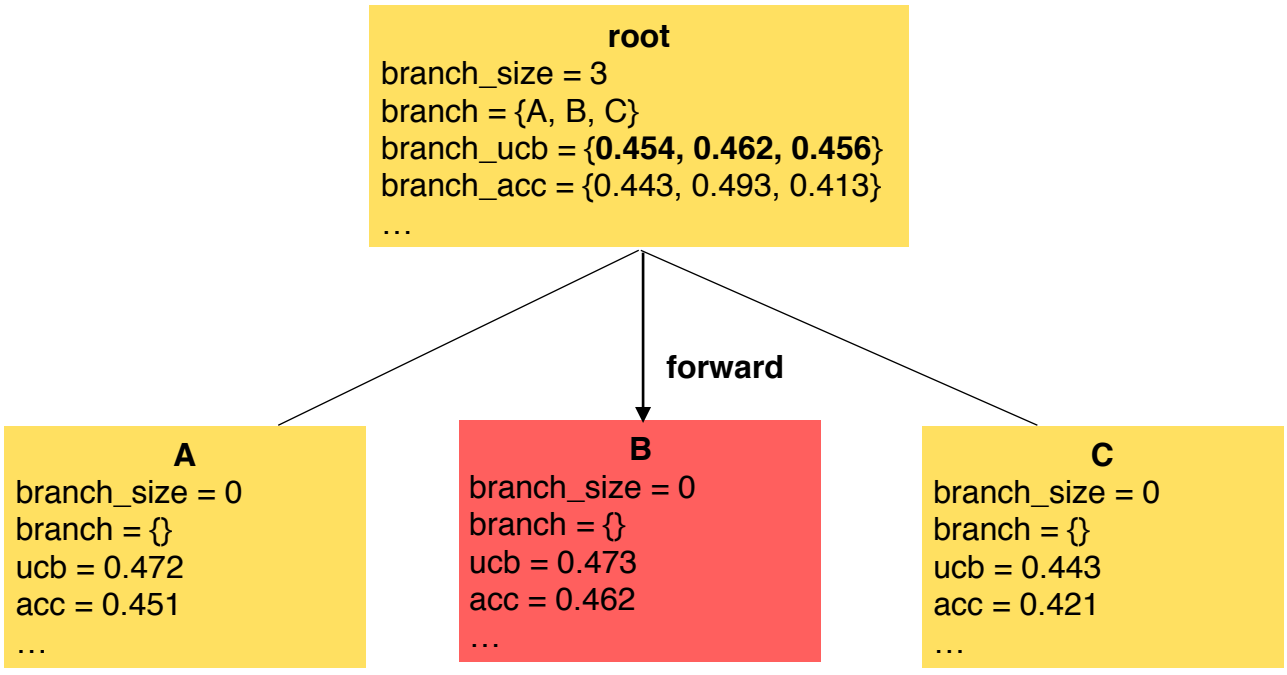
…
do 20 rounds
calculate mean and standard on each node
and check progressive pruning



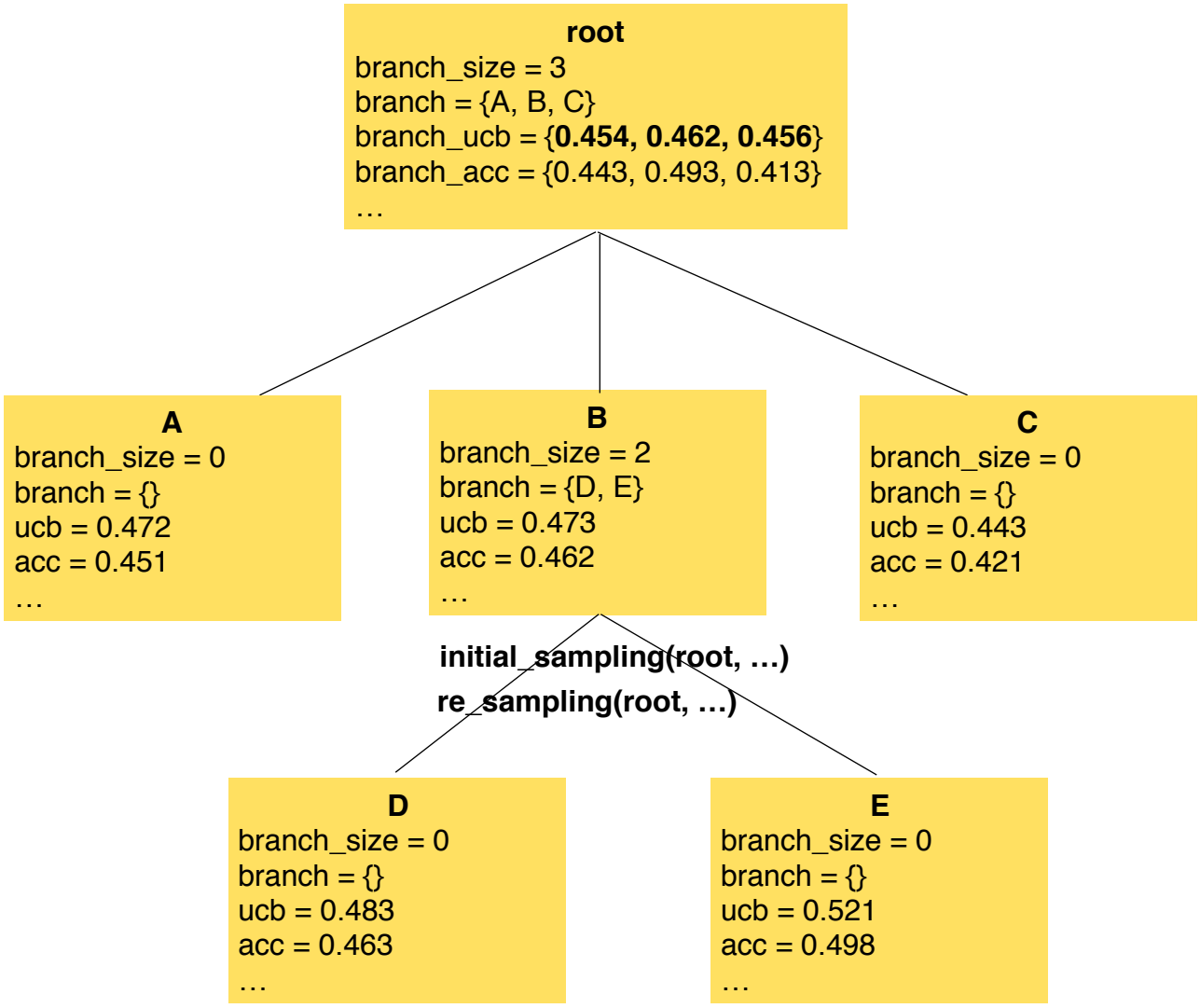
Backtrace and update UCB score



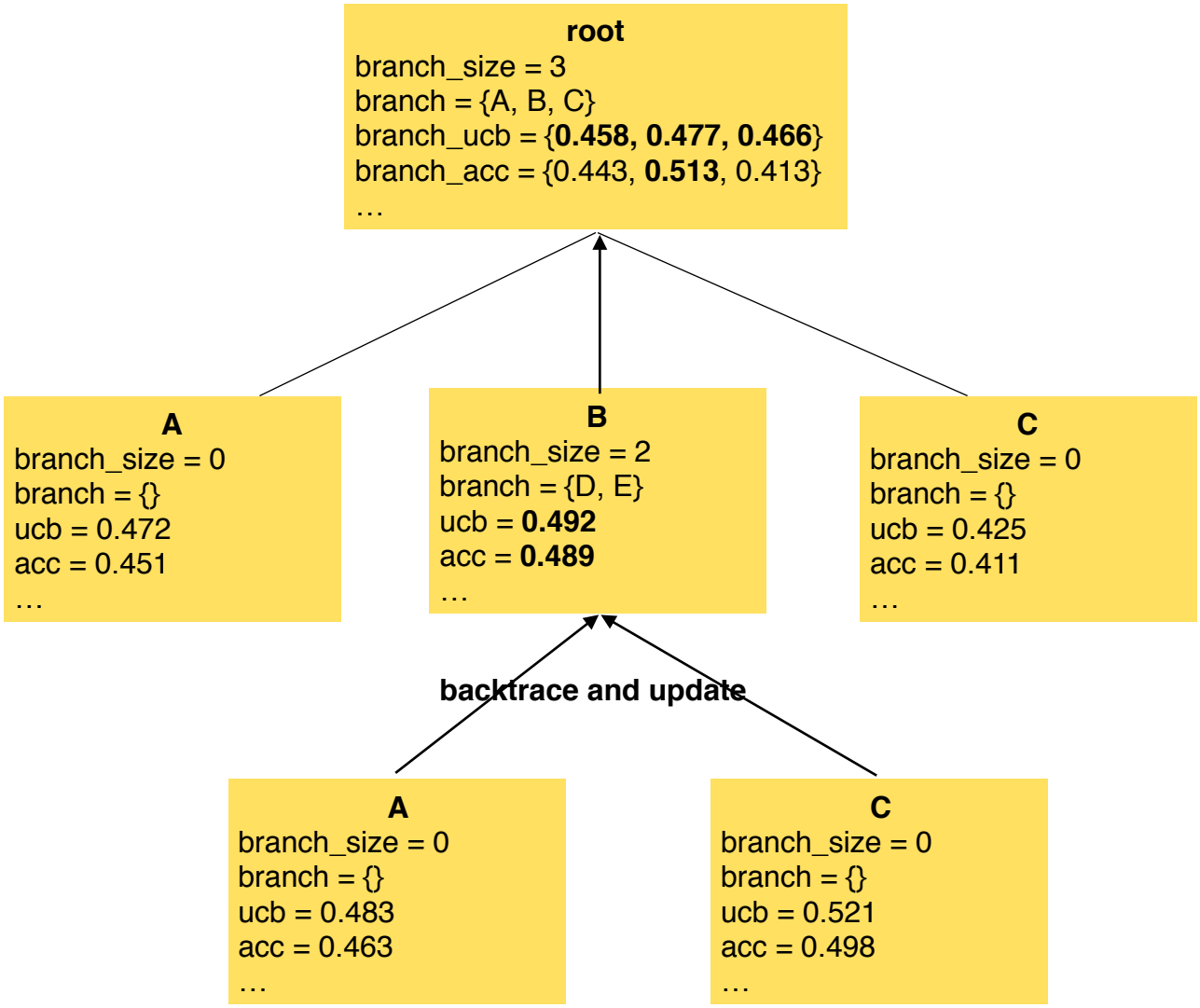
Start from the root
If the node is already have branch, choose the one with the best ucb score



Expansion
Do initial_simulation and re_simulation as mention above



Backtrace and update ucb score of node on PV path



不斷重複上述步驟，但要注意因為是min-max，所以層和層之間所選取的是不一樣的，一層取大一層取小，這邊利用一個變數min_max計算目前層數，再拿來mod 2就可以知道是min層還是max層，藉此不斷迭代，慢慢長出較佳的UCT樹，最後選擇root下正確率最高的做移動。

TreeNode 資料結構設計

- TreeNode *branch[64]
  記錄此節點下的所有branch，在forward時使用
- TreeNode *parent
  記錄此節點的parent，在backtrace時使用
- unsigned char map[8][8]
  記錄此節點的盤面
- int id
  此節點為同層的第幾個（由0開始）
- int win, loss, total
  此節點simulation的勝敗及總和次數統計
- int branch_size
  此節點有多少個branch
- double branch_ucb[64]
  此節點每個branch的ucb後面那一項根號(logN/Ni)的值，只存後面那一項是為了min-max，有時候前面像要取勝率有時候要取敗率
- double branch_acc[64]
  此節點每個branch的勝率，1-acc可以很快速地轉換為敗率
- double branch_mean_m_std[64]
  此節點每個branch的mean value
- double branch_mean_p_std[64]
  此節點每個branch的standard deviation
- int have_branch
  記錄此節點有沒有branch（可以知道有沒有被expansion過）
- int my_tile
  記錄此節點在盤面上是O還是X
- int enable
  記錄此節點有沒有被pruning掉，如果有值為0

特殊變數

- UCB後面那一項的係數c=1.126
  指導教授的生日，此數字為實驗室的幸運數字
- depth = 12
  在做UCT時會做expansion的上限次數，經過多次測試，太小樹會長不大，太大太久
- re_simu_size = 20
  當一個node下的branch都經過initial_simulation後，需要根據當前UCB較大的值做個別化的simulation，這邊設為20是讓他可以重複做20回合，做完一回合就更新一次，而每字做幾個simulation則是redo_size來決定
- redo_size = 150-min_max*30
  這個大小會根據目前的層數來遞減，因為越下面層所需花的力氣不用那麼多，太多會造成浪費
- initial_sampling_size = 4000 - depth*800
  這是initial simulation size，同上述，大小會隨著層數遞減。