ソフトウェア２
第４回
（2013/1/10）

鶴岡 慶雅

連絡用ページ

- URL

  http://www.logos.t.u-tokyo.ac.jp/~tsuruoka/lecture/software2/

  ユーザ名 :　soft2

  パスワード :　ee2012

- 資料
  - 講義スライド（ppt, pdf）
  - サンプルプログラム

今日の内容

- C言語入門
  - 関数の再帰呼び出し
  - assert
- オセロゲームの思考エンジン
  - AI入門（探索）
  - ミニマックス法
  - 深さ優先探索

関数の再帰呼び出し

- 再帰関数
  - 自分自身を(引数を変えて)呼び出す関数

- 例
  - 階乗を求める関数

```
int factorial(int n)
{
  if (n == 0) return 1;

  return n * factorial(n - 1);
}
```

関数の再帰呼び出し

- 再帰呼び出しを用いることでアルゴリズムをわかりやすく実装できることがある
- 例
  - ユークリッドの互除法（Euclid's algorithm）
    - 最大公約数を計算する効率的なアルゴリズム

while 文を用いた実装

```
int euclid(int m, int n)
{
  int remainder;

  while ((remainder = m % n) != 0) {
    m = n;
    n = remainder;
  }

  return n;
}
```

再帰関数を用いた実装

```
int euclid(int m, int n)
{
  int remainder = m % n;

  if (remainder == 0)
    return n;
  else
    return euclid(n, remainder);
}
```

デバッグ

- プログラムが意図した通りに動かない
  ⇒　デバッグ（debugging）

- デバッグ手法
  - printf デバッグ
    - プログラム中の重要な変数等をひたすら表示
  - 変数の値の動的なチェック
    - assert文
  - デバッガを使う
    - gdb

assert

- assert 文で開発者の意図をコードに埋め込む
  - 引数、変数等の値の動的なチェック
  - 意図どおりの値でない場合、プログラムが停止

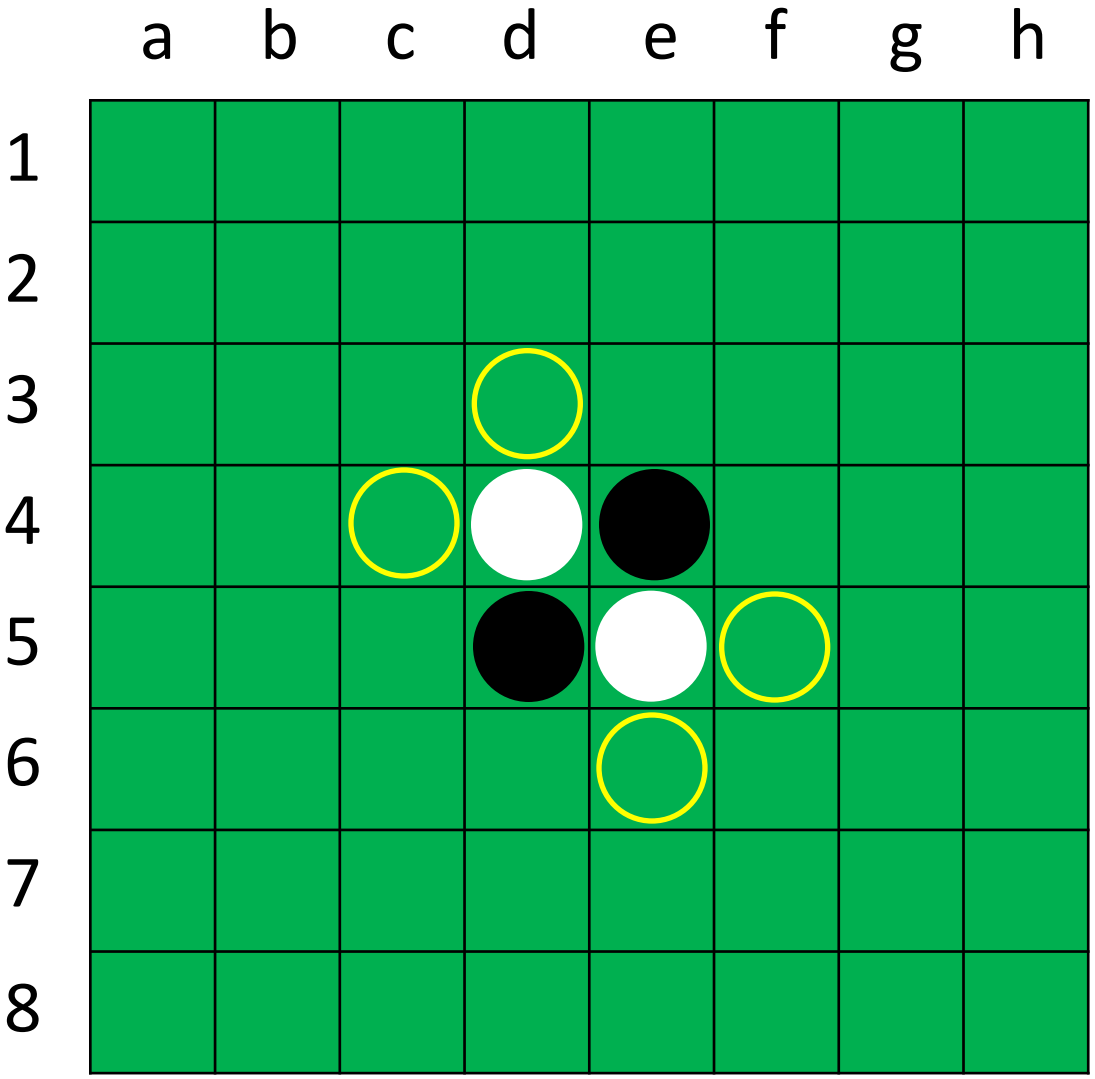
```
int euclid(int m, int n)
{
  assert(m > 0 && n > 0);

  int remainder = m % n;

  if (remainder == 0)
    return n;
  else
    return euclid(n, remainder);
}
```

m も n も絶対 0 より大きいハズ！
（そうでない場合、どこかで間違ってる）

コンパイル時に –DNDEBUG オプションを付けると assert文は全て無視されるので、プログラムの性能に影響を与えない

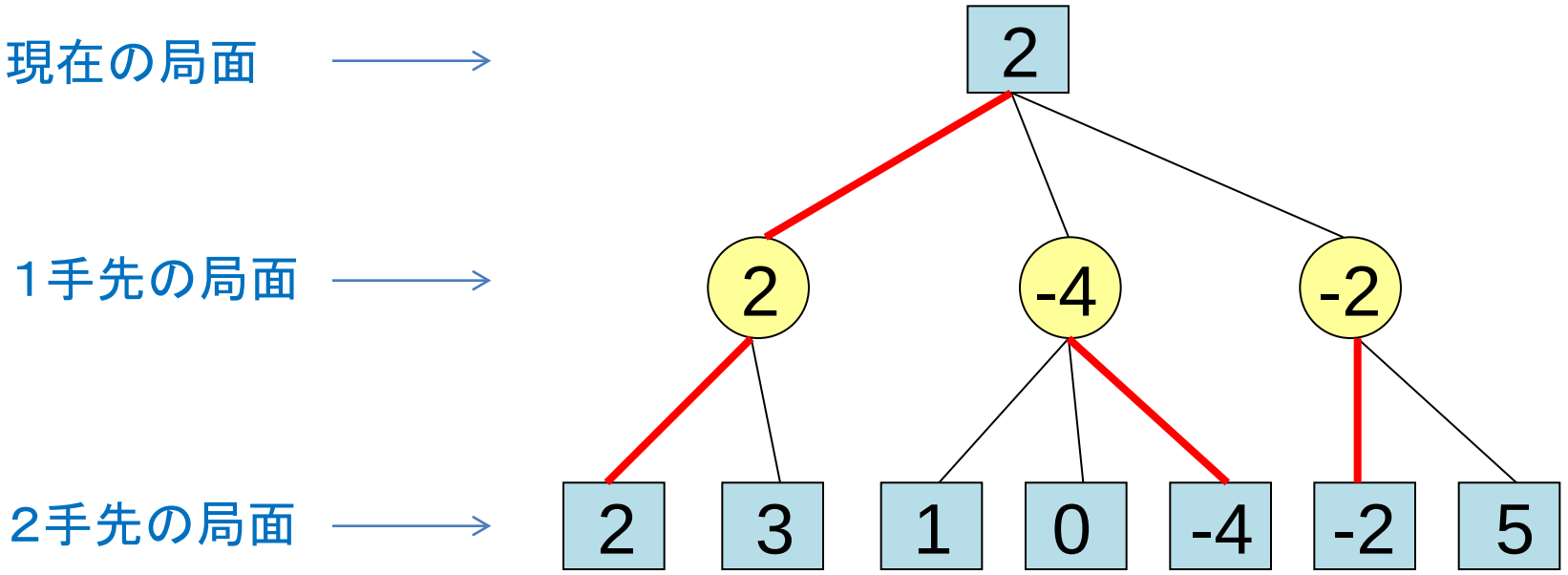
assert.h をインクルードすること

オセロゲーム（Reversi）



- 黒、白が交互に置く
- 相手のディスクを自分のディスクで挟む（縦・横・斜め）と自分のディスクになる
- 初期局面での合法手は4つ
- 合法手が存在しない場合はパス
- 最終的にディスクの数が多い方が勝ち

ミニマックス法
(Minimax algorithm)



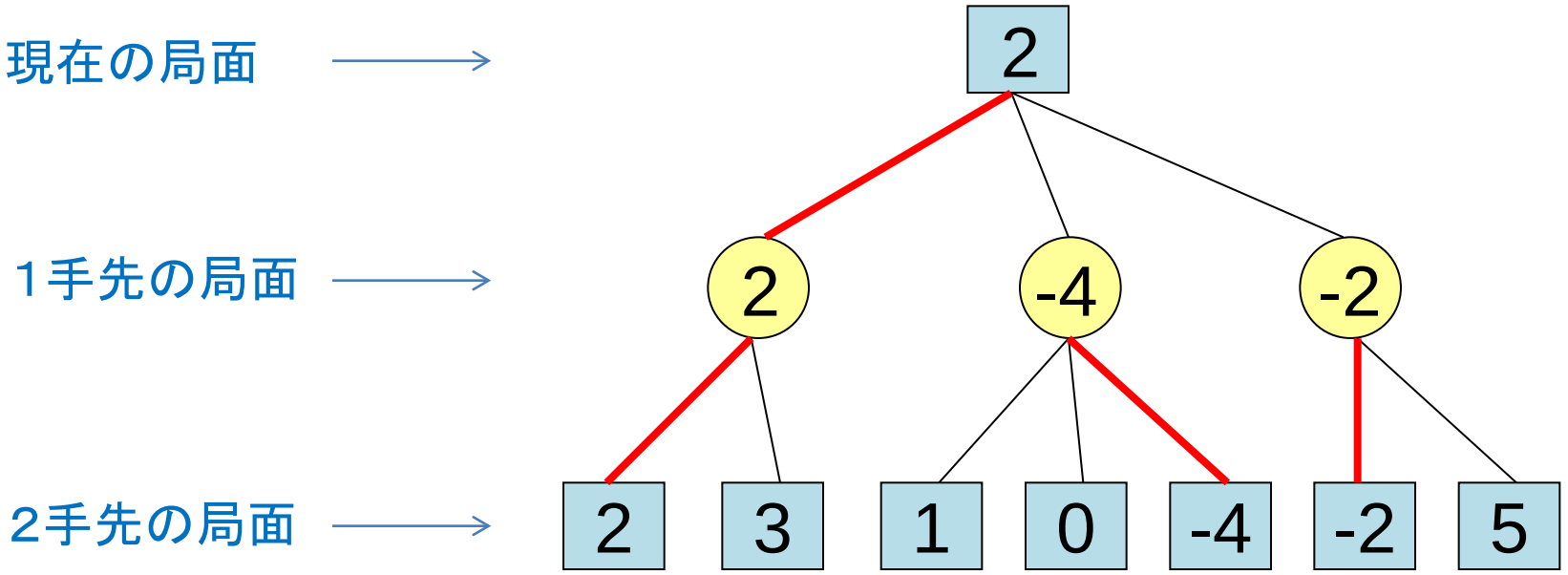
- **評価関数**によって末端局面の有利・不利の度合いを数値化
- お互いが自分にとって最も都合の良い手を選ぶと仮定して、スコア、最善手を逆算

評価関数（evaluation function）

- 局面の有利不利を数値化する関数
  - 互角ならゼロ
  - 先手（黒番）にとって有利であればプラス
  - 後手（白番）にとって有利であればマイナス

- オセロの場合
  - 盤上のディスクの数
    - （黒のディスクの数 ）ー（白のディスクの数）
  - 角のディスクにはボーナス
  - 角の隣のディスクはちょっと減点
  - ：

深さ優先探索（depth-first search）



- 関数の再帰呼び出しで簡単に実装できる
- 木構造をつくる必要がないので省メモリ

ミニマックス法の実装

- Maxノード

```
int max_node(depth, max_depth)
    if depth == max_depth:
        return eval_func()
    best = -infinity
    for all legal moves:
        update_board(move)
        v = min_node(depth+1, max_depth)
        if v > best:
            best = v
        undo_board()
    return best
```

- Minノード

```
int min_node(depth, max_depth)
    if depth == max_depth:
        return eval_func()
    best = infinity
    for all legal moves:
        update_board(move)
        v = max_node(depth+1, max_depth)
        if v < best:
            best = v
        undo_board()
    return best
```

- 実際には最良スコアだけでなく最良手も返す必要あり
- 木の末端よりも手前で勝負がついていることも

Negamax法

- 対称性を利用することでひとつの関数で書ける

```
int negamax(depth, max_depth)
    if depth == max_depth:
        return eval_func() * turn
    best = -infinity
    for all legal moves:
        update_board(move)
        v = -negamax(depth+1, max_depth)
        if v > best :
            best = v
        undo_board()
    return best
```

turn の値は、今の局面が
先手番であれば +1
後手番であれば -1

頭にマイナスがついていることに注意

サンプルプログラム reversi.c

- コンパイル

```
$ gcc -O2 -DNDEBUG reversi.c
```

assert を全て無視してコンパイル
（プログラムの実行速度が速くなる）
開発作業中は、
gcc -g reversi.c

- 実行
  - コンピュータ同士の対戦

```
$ ./a.out
```

  - 人間が先手（black）

```
$ ./a.out 1
```

  - 人間が後手（white）

```
$ ./a.out -1
```

課題（締め切り1/16）

1. 人間に合法手がない場合、パスができるように改良せよ
   – プログラムを添付すること（ファイル名は “reversi1.c”）

2. 対局終了時に勝利者を判定するように拡張せよ
   – プログラムを添付すること（ファイル名は “reversi2.c”）

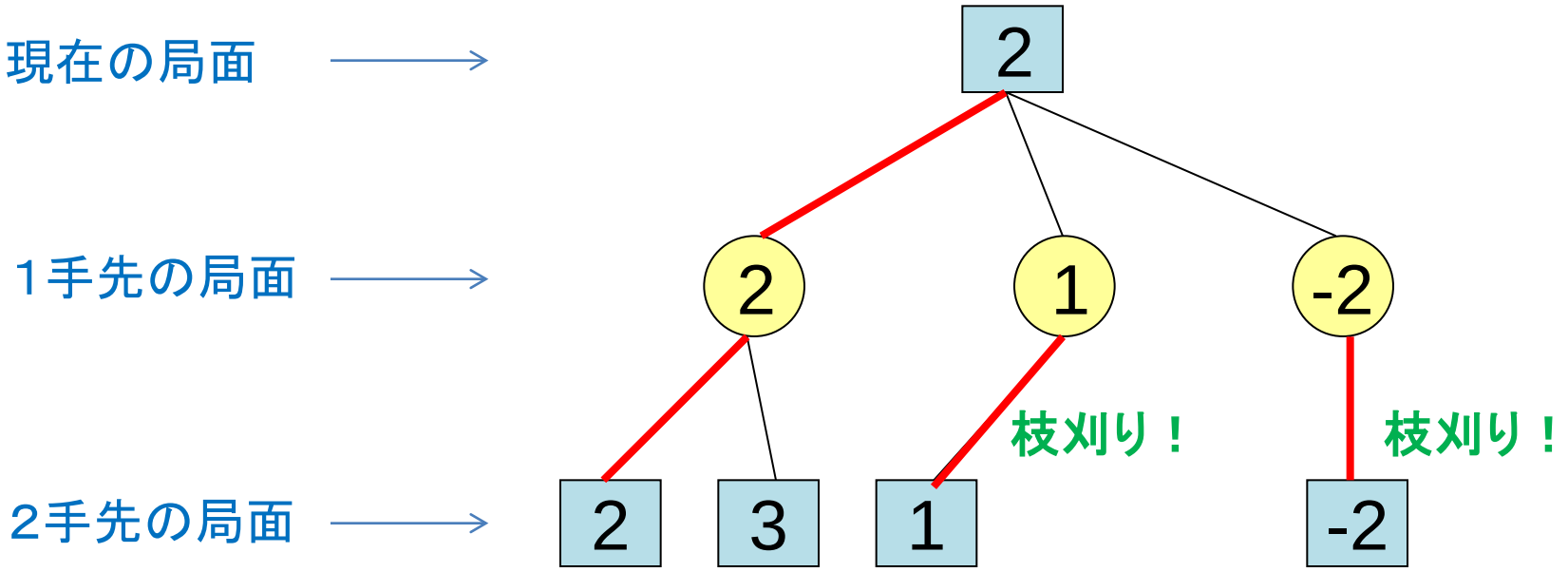
3. ミニマックス法を実装せよ
   – プログラムを添付すること（ファイル名は “reversi3.c”）

4. ［発展課題］ 思考エンジンをさらに強化せよ
   – プログラムを添付すること（ファイル名は “reversi4.c”）
   – αβ枝刈りの導入、評価関数の自動チューニング、高速化、etc
   – 用いた手法とその効果を簡単に説明すること

課題の提出方法

- 宛先
  - software2@logos.t.u-tokyo.ac.jp
- Subject
  - 形式: SOFT-MM-DD-NNNNNNX
    - MM: 月
    - DD: 日（授業が行われた日）
    - NNNNNNX: 学籍番号
- 本文
  - 冒頭に学籍番号、氏名を明記

（参考）枝刈り



- 探索の結論を変えずに探索ノード数を大幅に減らせる