THINK FAST

Competitive Programming

From Problem 2 Solution in O(1)

Combinatorial Game Theory

Conjunctive Compound Games

Mostafa Saad Ibrahim
PhD Student @ Simon Fraser University

Other Compound games

- Selective compound (covered last session)
  - Move **at least 1** knight
  - Move **at least 1** knight but **NOT all**
  - Make use of grundy
- Conjunctive compound
  - Move **all** knights (short rule)
  - **Minimax** style game to count the game # steps
  - E.g. introduce **Remoteness function**
- Continued conjunctive compound
  - Move in **all available** knights (long rule)
  - **Maximin** / **Suspense function** (count the game # steps)

Conjunctive compound

- Game (short rule)
  - Conjunctive = Move **all** knights
  - **Short** rule: Once any pile is empty=> we know winner
  - Players now are **very sensitive** to the sub-games that takes very **little steps** to be completed!
  - That is, player has interest in winning **quickly** on winning sub-games and postponing defeat **as long as** possible on losing ones (different winning concerns)
  - This makes **grundy useless**
    - In knight chess, *Grundy(1, 2) = Grundy(7, 6) = 2*
    - Although knight(1, 2) can be completed in 1 step!
  - We need function that **computes game steps**!

Conjunctive compound

- **Remoteness function r**
  - the # of games steps if a player who can **force a win** will try to **win as soon as possible** and the losing player will try to **lose as slowly as possible**.
  - Losing-position has **even r** and W-positions have **odd r**.
    - Intuition: if after odd step game ends = first win
  - r(terminal position) = 0
  - For each sub-move, compute r for its sub-game
  - If any sub-game is even: r(x) = 1 + **smallest even** r
    - Win using min # steps
  - otherwise r(x) = 1 + **largest odd** r
    - Lose using max # steps

Conjunctive compound

- For knights game: r table
  - knight(1, 2) needs 1 step to 1 to win
  - knight(7, 6) needs 5 steps to win
  - knight(4, 4) = 4 = loses
  - Recall:
  - **odd => win**
  - **even => lose**

0	0	1	1	2	2	3	3
0	0	$\boxed{1}$	1	2	2	3	3
1	1	1	1	3	3	3	3
1	1	1	3	3	3	3	5
2	2	3	3	$\boxed{4}$	4	5	5
2	2	3	3	4	4	5	5
3	3	3	3	5	5	5	5
3	3	3	5	5	5	$\boxed{5}$	6

Conjunctive compound

```
int remotnessMem[120][120];
int suspenseMem[120][120];

bool valid(int v) {
  return v >= 0 && v < 8;
}

const int DIR = 4;
int dr[DIR] = { 1, -1, -2, -2 };
int dc[DIR] = { -2, -2, 1, -1 };
const int OO = 1000000;
```

Conjunctive compound

```
int calcRemoteness(int r, int c) {
  int &ret = remotnessMem[r][c];
  if (ret != -1)
    return ret;

  int leastEven = OO, largestOdd = -OO;

  for (int d = 0; d < 4; ++d) {
    if (valid(r + dr[d]) && valid(c + dc[d])) {
      int remotness = calcRemoteness(r + dr[d], c + dc[d]);
      if (remotness % 2 == 0)
        leastEven = min(leastEven, remotness);
      else
        largestOdd = max(largestOdd, remotness);
    }
  }
  ret = 0;
  if (leastEven != OO)
    ret = 1 + leastEven;
  else if (largestOdd != -OO)
    ret = 1 + largestOdd;
  return ret;
}
```

Conjunctive compound

- Given k sub-games, who win?
  - Compute r function for each position
  - r(game) = min(r(g1), r(g2)...., r(gn))
  - if the overall game r is even => lose
- So r(g1, g2) = min(r(g1), r(g2))
- Do you notice the overall corresponds to grundy computations?
  - sg(game) = **xor**(sg(g1), sg(g2)...., sg(gn))

Conjunctive compound

```
void chessRemotenessMain() {
  clr(remotnessMem, -1);

  for (int i = 0; i < 8; ++i) {
    for (int j = 0; j < 8; ++j) {
      cout << calcRemoteness(i, j) << " ";
    }
    cout << "\n";
  }

  int minVal = 1000000, knights;

  cin>>knights;
  for (int d = 0; d < knights; ++d) {
    int x, y;
    cin>>x>>y;
    minVal = min(minVal, calcRemoteness(x, y));
  }
  if(minVal % 2 != 0)    cout<<"First win";
  else                   cout<<"Second win";
}
```

Continued conjunctive compound

- Game (conjunctive compound long rule)
  - Not so intuitive. Think about it.
  - Move in **all available** knights
    - If some piles are empty, we keep playing (normal)
  - Losing subgame is ok if some subgames are not nished.
  - Winning strategy: finish losing subgames as soon as possible and play winning subgames as long as possible
  - Overall, it is the reverse of shor rule case
  - The function to compute # of steps named **Suspense function**

Continued conjunctive compound

- **Suspense function s**
  - the # of games steps if the loser player will try to **lose as soon as possible** and the winner player will try to **win as long as possible**.
  - Losing-position has **even s** and W-positions have **odd s**.
  - s(terminal position) = 0
  - For each sub-move, compute r for its sub-game
  - If any sub-game is event: s(x) = 1 + **largest even** s
    - Win using max # steps
  - otherwise s(x) = 1 + **smallest odd** s
    - Lose using min # steps

Continued conjunctive compound

- Suspense values (very close to r)

0	0	1	1	2	2	3	3
0	0	1	1	2	2	3	3
1	1	1	3	3	3	3	3
1	1	3	3	3	3	5	5
2	2	3	3	2	4	5	5
2	2	3	3	4	4	5	5
3	3	3	5	5	5	5	5
3	3	3	5	5	5	5	6

Continued conjunctive compound

- Given k sub-games, who win?
  - Compute s function for each position
  - s(game) = **max**(s(g1), s(g2)...., s(gn))
  - if the overall game s is even => lose
- So s(g1, g2) = max(s(g1), s(g2))

Continued conjunctive compound

```
int calcSuspense(int r, int c) {
  int &ret = suspenseMem[r][c];
  if (ret != -1)
    return ret;

  int largestEven = -OO, leastOdd = OO;

  for (int d = 0; d < 4; ++d) {
    if (valid(r + dr[d]) && valid(c + dc[d])) {
      int remotness = calcSuspense(r + dr[d], c + dc[d]);
      if (remotness % 2 == 0)
        largestEven = max(largestEven, remotness);
      else
        leastOdd = min(leastOdd, remotness);
    }
  }
  ret = 0;
  if (largestEven != -OO)
    ret = 1 + largestEven;
  else if (leastOdd != OO)
    ret = 1 + leastOdd;
  return ret;
}
```

Continued conjunctive compound

```
void chessSuspenseMain() {
  clr(remotnessMem, -1);

  for (int i = 0; i < 8; ++i) {
    for (int j = 0; j < 8; ++j) {
      cout << calcRemoteness(i, j) << " ";
    }
    cout << "\n";
  }


  int maxVal = -1000000, knights;

  cin>>knights;
  for (int d = 0; d < knights; ++d) {
    int x, y;
    cin>>x>>y;
    maxVal = max(maxVal, calcRemoteness(x, y));
  }
  if(maxVal % 2 != 0)   cout<<"First win";
  else                  cout<<"Second win";
}
```

Misere for conjunctive cases

- In both previous cases, all what to do, interchange words even and odd
  - Then to win in conjunctive compound
    - r(x) = 1 + smallest odd r
  - Also, if **final game is even, you win**
    - Remember it with base case misere(0) => 1st win
  - E.g. RemotenessMisere code
    - = Suspense code with 2 simple changes
- In other words, in normal and misere plays, we are coding classical dp functions

Misere for conjunctive compound

```
int calcRemotenessMisere(int r, int c) {
   int &ret = remotnessMem[r][c];
   if (ret != -1)
     return ret;

   int leastOdd = OO, largestEven = -OO;

   for (int d = 0; d < 4; ++d) {
     if (valid(r + dr[d]) && valid(c + dc[d])) {
       int remotness = calcRemotenessMisere(r + dr[d], c + dc[d]);
       if (remotness % 2 != 0)  // 1st change in calcSuspense
         leastOdd = min(leastOdd, remotness);
       else
         largestEven = max(largestEven, remotness);
     }
   }
   ret = 0;
   if (leastOdd != OO)     // 2nd change in calcSuspense
     ret = 1 + leastOdd;
   else if (largestEven != -OO)
     ret = 1 + largestEven;
   return ret;
}
```

Finally

- The only trick in this game is to notice its direct **correspondence to minimax, maximin** games (E.g. Grundy won't be used)
- Misere case under such reccurance is a normal thing too, as we don't need any theory
- Code is normal and you can write from mind once you know the **optimal strategy** for each player
- Remoteness/Suspnse are not new theory :)

Summary: normal play

Disjunctive compound	• Long rule, Like Nim, based on xor values for the piles • Grundy is computed for sub-games, then xor sub-games grundies
Diminished **disjunctive** compound	• Short rule version • W-numbers = Modified Grundy* = {SL | SW | Grundy-{SW, SL}}
Selective compound	• Move in at least 1 sub-game • If ALL grundies = 0 ⇒ Losing position
Selective compound - variant	• Move in at least 1 sub-game **but not all** • If ALL grundies are equal value ⇒ Losing position
Shortened **selective** compound	• Short rule. Move in at least 1 sub-game • If ALL grundies = 0 ⇒ Losing position
Continued **conjunctive** compound	• Long rule • Winning strategy: Classical **Maximin** (max steps to win / min to lose) • Compute # of game steps (suspense function)
Conjunctive compound	• Similar, short rule (**Minimax** = (min steps to win / max to lose)) • Compute # of game steps (remoteness function)

Summary: misere play

Disjunctive compound	• Misere name defined. Normal nim + special base case handling • Grundy is **NOT** defined - so not solvable in complex games • As base case grundy = 1, and other winning position = 0
Diminished **disjunctive** compound	• Surprisingly modified grundy (w-numbers) are defined. • Short rule + SW position trick is a reason behind
Selective compound	• If ALL grundies = 0 ⇒ Losing position • Only 1 grundy = 0 ⇒ Wining position (exception for above case) • Otherwise ⇒ Winning position
Selective compound - variant	•
Shortened **selective** compound	• Same as normal play..no special handling!
Continued **conjunctive** compound	• A classical **Maximin game**, just normal **recurrence** handling
Conjunctive compound	• A classical **Minimax game**, just normal **recurrence** handling

Final notes

- 3 critical parts in game theory solving
  a. **Win/Lose positions classical rules**
  b. **Base Case analysis**
     - From **a, b**: Some games are the normal nim playing, except special handling for the bottom cases
     - Once bottom cases are handled, remaining from bottom to top is normal Win/Lose rules
     - Examples: Nim misere, Diminished Disjunctive
  c. **Identifying the winning strategy**
     - From Win Strategy => grundy, w-number, remoteness, suspense functions, adhock recurrence handling...etc)
     - Xor/Mex/Grundy are the critical theories parts

Reading

- For formal proves, please read
  - See 1
  - See 2

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً