**BLG 435E – Artificial Intelligence**

**2019-2020 Fall, Assignment 2 Report**

**Submission Deadline: 03.12.19, 23:59**

Kadir Emre Oto (150140032)

**Problem 1 – First-Order Logic Representation (10 Points)**

**Problem 2 – FOL, Inference via Resolution (30 Points)**

**Problem 3 – Optimal Decision in Games (60 Points)**

**Compile and Run Commands:**

g++ minimax.cpp -o minimax -O2 -std=c++11

g++ minimax\_alpha\_beta.cpp -o minimax\_alpha\_beta -O2 -std=c++11

./minimax <input-file>

./minimax\_alpha\_beta <input-file>

**My state representation:**

In order to reduce the memory taken and easily memoize the solution (aka dynamic programming), we can hold the state in a number (in this case long long) using bitmask (also this value will be our hash value). So, we need:

* 1 bit for player turn (minimize or maximize) (M)
* 12 bits for vertical edges (V)
* 12 bits for horizontal edges (H)
* 4 bits for score of Player-1 (A)
* 4 bits for score of Player-2 (B)

Hence, we can use a ***long long integer*** to represent the state (because we need 33 bits at least) -> {? x 31} {M x 1} {V x 12} {H x 12} {A x 4} {B x 4}

|  |  |  |  |
| --- | --- | --- | --- |
| **Bit Number** | **Meaning** | **Bit Number** | **Meaning** |
| 33..63 | insignificant | 16 | 9th horizontal edge |
| 32 | min or max | 15 | 8th horizontal edge |
| 31 | 12th vertical edge | 14 | 7th horizontal edge |
| 30 | 11th vertical edge | 13 | 6th horizontal edge |
| 29 | 10th vertical edge | 12 | 5th horizontal edge |
| 28 | 9th vertical edge | 11 | 4th horizontal edge |
| 27 | 8th vertical edge | 10 | 3rd horizontal edge |
| 26 | 7th vertical edge | 9 | 2nd horizontal edge |
| 25 | 6th vertical edge | 8 | 1st horizontal edge |
| 24 | 5th vertical edge | 7 | 1st bit of Player 1’s score |
| 23 | 4th vertical edge | 6 | 2nd bit of Player 1’s score |
| 22 | 3rd vertical edge | 5 | 3rd bit of Player 1’s score |
| 21 | 2nd vertical edge | 4 | 4th bit of Player 1’s score |
| 20 | 1st vertical edge | 3 | 1st bit of Player 2’s score |
| 19 | 12th horizontal edge | 2 | 2nd bit of Player 2’s score |
| 18 | 11th horizontal edge | 1 | 3rd bit of Player 2’s score |
| 17 | 10th horizontal edge | 0 | 4th bit of Player 2’s score |

**Table 1:** bit-mask representation

**Implementation**

I used the standard minimax algorithm in order to find optimal game decisions. However, some states can be explored many times, I used very common algorithm technique, dynamic programming, to reduce run time and time complexity. If any state is discovered one time, we do not need to branch that point, so we can basically save the answers of states and return the saved results if the algorithm encounters the same state after. The other problem is hashing the state to save results, than I used bitmask to represent states with 33 bits (Table 1).

The time complexity = (O ())

In worst case, I expect that the operation count should not exceed with given constraints.

**Results**

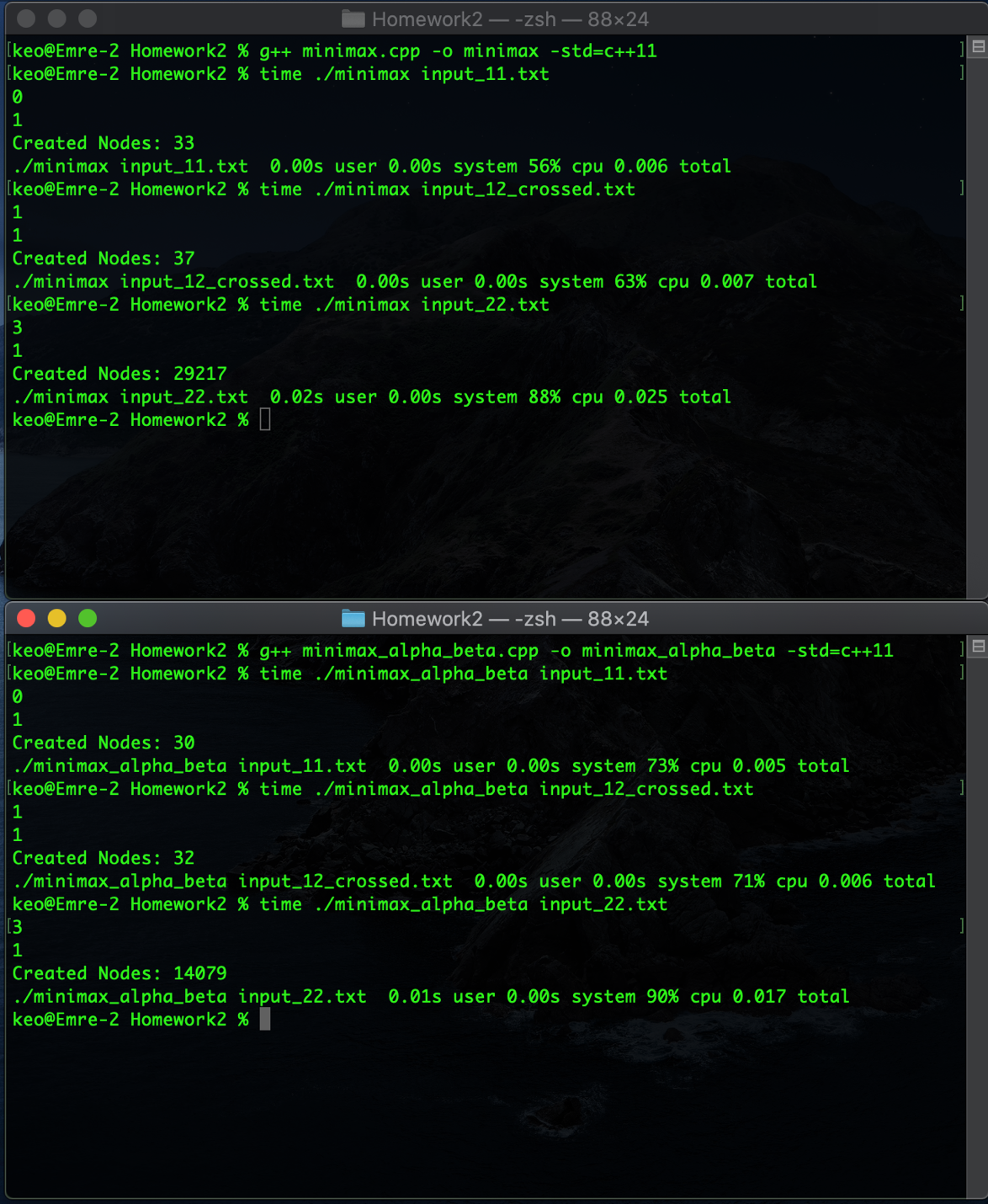


Figure 1: Running time and created node counts