NUMBER MIND

Crack a Hack

Course: Algorithmic Problem Solving
Course code: 17ECSE309

Akshay Gudiyawar
01FE15BEC021

1. Introduction

This problem is variant of a popular logic strategy game called – “Master mind”.

Mastermind or Master Mind is a code-breaking game for two players. The modern game with pegs was invented in 1970 by Mordecai Meirowitz, an Israeli postmaster and telecommunications expert.

Gameplay in brief:
The two players decide in advance how many games they will play, which must be an even number. One player becomes the codemaker, the other the codebreaker. The codemaker chooses a pattern of four code pegs. Duplicates and blanks are allowed depending on player choice, so the player could even choose four code pegs of the same colour or four blanks. In the instance that blanks are not elected to be a part of the game, the codebreaker may not use blanks in order to establish the final code. The chosen pattern is placed in the four holes covered by the shield, visible to the codemaker but not to the codebreaker

The codebreaker tries to guess the pattern, in both order and color, within twelve (or ten, or eight) turns. Each guess is made by placing a row of code pegs on the decoding board. Once placed, the codemaker provides feedback by placing from zero to four key pegs in the small holes of the row with the guess. A colored or black key peg is placed for each code peg from the guess which is correct in both color and position. A white key peg indicates the existence of a correct color code peg placed in the wrong position

Once feedback is provided, another guess is made; guesses and feedback continue to alternate until either the codebreaker guesses correctly, or twelve (or ten, or eight) incorrect guesses are made.
The codemaker gets one point for each guess a codebreaker makes. An extra point is earned by the codemaker if the codebreaker doesn't guess the pattern exactly in the last guess. (An alternative is to score based on the number of colored key pegs placed.) The winner is the one who has the most points after the agreed-upon number of games are played.

<u>This is generically solved by →</u>
Five-guess algorithm:
This progressively reduces the number of possible patterns

1. Create the set S of sample space
2. Start with initial guess
3. Play the guess and calculate a response
4. If the response has zero error, the game is won, the algorithm terminates.
5. Otherwise, change the mutation (digit guesses) to previous best one.
6. Apply minimax technique to find a next guess
7. Repeat through step 3

2. Algorithm

This problem can be solved with simulated annealing [2]. The concept is as follows:

- Start with a random guess and compute its error (distance() function)
- Modify single elements randomly and check whether the error is reduced, keep the best mutation
- If mutations fail to improve for a certain time (here 20 rounds), change an element randomly even if the error gets worse

Last point is important because, otherwise the program gets stuck at local optimum.

distance() compares the current sequence against all guesses.
Here, error is the sum of all differences between the actual matching digits and the expected number.
For example, comparing 6666666666 against the first guess 5616185650518293 finds 3 matching digits, but only 2 are expected, so the error is → 1

The variable MaxRoundsWithoutImprovement = 20 is optimized for minimizing execution time.

Hackerrank Problem link:
https://www.hackerrank.com/contests/projecteuler/challenges/euler185

3. Code

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

typedef std::vector<unsigned char> Sequence;
vector<Sequence>     sequences;

vector<unsigned int> hits; // this stores how many of their digits match the secret number

unsigned int myrand(unsigned int modulo) // psuedo-random no generator ... for guessing the
numbers
{
  static unsigned int seed = 0;
  seed = 1103515245 * seed + 12345;
  return seed % modulo;
```

```
}

void shuffle(unsigned char& digit) // shuffle by a new random digit
{
  auto old = digit;
  do
    digit = myrand(10);
  while (digit == old);
}

void add(const std::string& guess, unsigned int matches)
{
  Sequence s;
  for (auto c : guess)
    s.push_back(c - '0');             // ASCII to int conversion
  sequences.push_back(s);

  hits.push_back(matches);
}

unsigned int distance(const Sequence& current) // to compute how many of the random
guessed digits match the current number
{
  unsigned int errors = 0;

  for (unsigned int i = 0; i < sequences.size(); i++)
  {
    // count number of matching digits
    unsigned int same = 0;
    for (unsigned int j = 0; j < current.size(); j++)
      if (current[j] == sequences[i][j])
        same++;

    if (same > hits[i])
      errors += same - hits[i];
    else
      errors += hits[i] - same;
  }

  return errors;
}

int main()
{
```

```
unsigned int numGuesses;
cin >> numGuesses;
while (numGuesses--)
{
  string guess;
  unsigned int correct;
  cin >> guess >> correct;
  add(guess.c_str(), correct);
}

// initially a random guess
const auto NumDigits = sequences.front().size();
Sequence current(NumDigits, 0);
for (auto& x : current)
  shuffle(x);

// shuffle a random digit when stuck in a local optimum
const auto MaxRoundsWithoutImprovement = 20;
auto quietRounds = 0;

auto errors   = distance(current);
auto previous = errors;
while (errors != 0)
{
  // replace every digit by a different random number, keep those that minimize the error
  for (auto& digit : current)
  {
    // replace by a new random digit
    auto previousDigit = digit;
    do
      shuffle(digit);
    while (digit == previousDigit);

    // calculate error
    auto modified = distance(current);
    if (modified <= errors)
    {

      errors = modified;
    }
    else
      // if new guess is bad, keep the previous one
      digit = previousDigit;
    }

    if (errors == previous)
    {
      // avoid getting stucked at local optimum
      quietRounds++;
      if (quietRounds == MaxRoundsWithoutImprovement)
      {
        // change a random number
        shuffle(current[myrand(current.size())]);
        errors = distance(current);

        // reset counter
        quietRounds = 0;
      }
    }
    else
    {
      quietRounds = 0;
      previous = errors;
    }
  }

  for (auto c : current)
    cout << int(c);                     // print the answer
  cout << std::endl;

  return 0;
}
```

4. References

1. https://en.wikipedia.org/wiki/Linear_congruential_generator
2. https://en.wikipedia.org/wiki/Simulated_annealing
3. https://en.wikipedia.org/wiki/Mastermind_(board_game)

4. http://www.cs.uni.edu/~wallingf/teaching/cs3530/resources/knuth-mastermind.pdf
5. https://lirias.kuleuven.be/bitstream/123456789/164803/1/kbi_0806.pdf