Report for mastermind
(CID:01333269)

Introduction:

Game of the mastermind was a well know deductive game played by two players. Classically the game was developed by 6 different colors and 4 different balls. Mastermind is one of the most famous optimizing problems. In the program, we are required to design a general solution for the game which means we could solve the problem with the high-speed computer and the algorithm.

Feedback function:

- The definition of black-hits and white-hits:

  Black-hits: if the guessed number is contained in the sequence and it is in the right position, it would be a black-hits;

  White-hits: if the guessed number is contained in the sequence but it is not in the right position, it would be a white-hits;

- The methods to get the value of black-hits and white-hits:

  According to the definition for the black-hits and white-hits there could be calculated by two different ways ,the first method was comparing the true sequence and the attempts first to get the black hits, then , due to the number could not be hit twice ,the both positions of black-hits can be removed, then storing the rest numbers in two sequences into other vectors, at last compared with two sequences to get the white-hits.

  Compared with the first method the second method could be much more efficient, the formula on the documents could be used to calculate the white-hits, the method to get the black-hits was same with the method which has be mentioned and using the minimal hits

of each numbers in the possibilities. And the value of white-hits could be calculated in w=sum-black-hits.

Here is some code to get the black-hits and white-hits:

The strategy to solve problem:

At first, one thing must be known is the possibility of each situation and the length of the sequence. Total situations could be calculated by number^length.

- Creating all attempts:

  The solution to create all situations could use the remainder and the set should be the vector to store the vectors.

  The length of the set is equal to the total numbers of situations and the length of each elements is equal to the length of sequence.

```
void creatallvector(std::vector<int>attempt,long p)
{
    for(long l=0;l<p;l++)
    {
        newint.clear();
        change(l, num, length, newint);
        all.push_back(newint);
    }
}
void change(long n,long num,int m,std::vector<int> &k)
{
    for (int i=m-1; i>=0; i--)
    {
        int p=n/(pow(num, i));
        k.push_back(p);
        n=n-p*pow(num, i);
    }
}
```

The code has been shown that change function is used to get the number form length=4, number=6, 1295 change to 5555, and put this vector into the total set.

- Select attempts:

  Let A to be the true sequence which was generated by the computer randomly. Then B1 is the first attempt or the previous attempt, after we call the function feedback, there would be an answer for black hits and white hits. Next step, the feedback for black hits and white hits could be used to reduce all the situations.

  Since, if A has value b for black hits and w for white hits which means B1 also would have the same feedback for A, as the result the next step would be to go through all situations then call the feedback function select all situations which fit the black hits and white hits in the previous attempt. Then it could be the true sequence.

  It is reasonable to try these steps because the following step could reduce all situations to a very small number of set, which must include the exact sequence.

```
void create_attempt(std::vector<int>& attempt){
    if(times==0){
        long p=pow(num,length);
        creatallvector(attempt, p);
        for(int u=0;u<length;u++){
            attempt.push_back(randn(num));
        }
        times++;
    }
    else{
        int n=randn(all.size());
        attempt.clear();
        for(int i=0;i<length;i++){
            attempt.push_back(all[n][i]);
            times++;
        }
    }
}
```

The code above is used to create the attempts which the first attempt could be create by random number, and after all that attempts could be select in the new set.

```
void learn(std::vector<int>& attempt, int black_hits, int white_hits){
    for(int k=0;k<all.size();k++){
        int b;
        int w;
        myfeedback(all[k], attempt, b, w);
        if((b==black_hits)&&(w==white_hits)){
            newall.push_back(all[k]);
        }
    }
    all.clear();
    for(int i=0;i<newall.size();i++){
        all.push_back(newall[i]);
    }
    newall.clear();

}
```

The code above is used to refresh the new set of all possibilities, which the old set would be cleared, and new set would be established.

Testing results:

Hence, the attempt chosen by the computer is according to the testing results randomly. The average number of 10000 tests would be 4.6 and the max guess number would be 7. Unfortunately, there was one time that

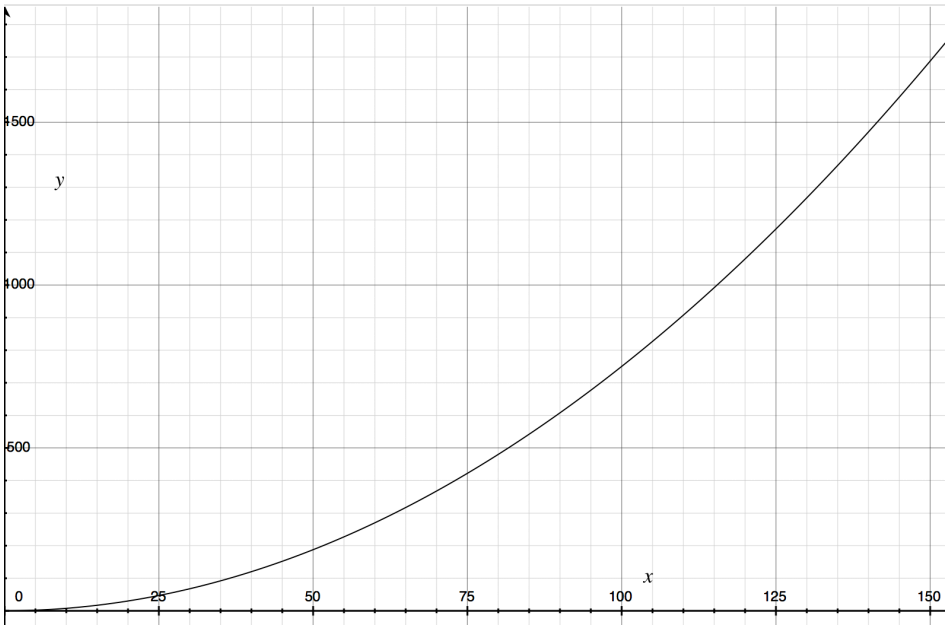
it appears the max attempts 8. According to the possibility of 8 attempts is much less than 1/10000, personally I ignored the results. And the average running time for length 4 and sequence 6 is 12s for each 10000 cycles, on the Linux os, in the computing lab.

Comparisons:

After finishing the coding, there are some algorithm on the website, there would be many algorithms could have the better average running time here is an image shows each algorithm:

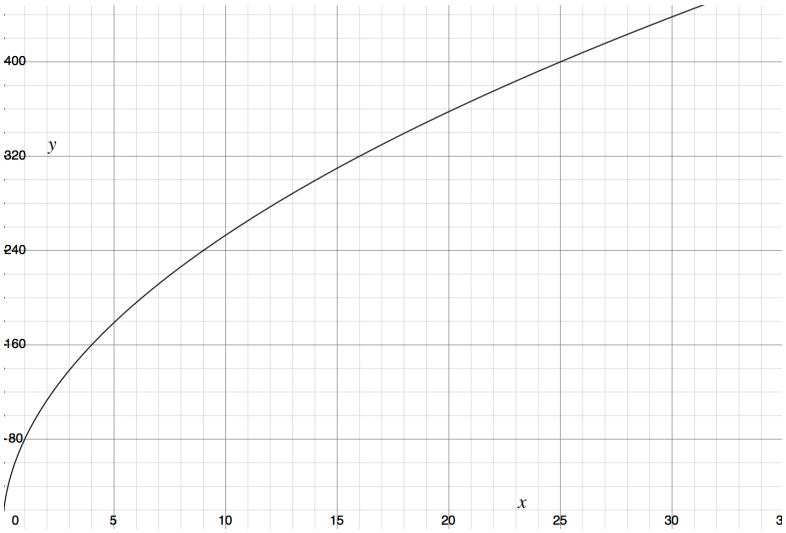
Strategy	*Total Moves*	*Average*	*Max Moves*	*6+ Moves*	*Author*	*Year*
Simple	7471	5.76466	9	853	Shapiro	1983
Worst case	5801	4.47608	5	0	Knuth	1977
Entropy	5722	4.41512	6	12	Neuwirth	1982
Expected Size	5696	4.39506	6	3	Irving	1979
Most parts	5668	4.37346	6	7	Kooi	2005
Optimal (Max: 5)	5626	4.34105	5	0	Lai	1993
Optimal (Max: 6)	5625	4.34028	6	1	Lai	1993

There would be another algorithm which is widely used: first randomly select a vector test if it would fit the feedback in all previous attempts, if and only if the vector fits all the attempts then it will become the next attempts. According to the test , the conclusion could be easily concluded that at first the running time of program is very short and in the last few attempts the CPU and memory usage would be very large , so the image of running time against the number^length:



As the result by using this algorithm computer would be easily get the true answer, however, dealing with the large sequence this algorithm cannot perform very well.

Alternatively, the algorithm I have used has the opposite results, which could be easily used to solve the large sequence, due to the program would first build a very large set which includes all situations so initially it would work a little bit slowly, nevertheless, it would be very fast at the last few attempts. so, the image of running time against the number^length:



Reference:
Donald E. Knuth (1976).the computer as mastermind.J.recreationasl mathematics, vol.9(1),1976-77,Available from: http://www.cs.uni.edu/~wallingf/teaching/cs3530/resources/knuth-mastermind.pdf、
Serkan gur optical mastermind solutions