DAA Assignment-05

Nischay Nagar
IIT2019198

Abhishek Bithu
IIT2019199

Raj Chandra
IIT2019200

***Abstract*—In this report we designed a Dynamic Programming algorithm to find the propablity of a knight to remain in the cheessboard after k number of steps.**

I. Introduction

It can be observed that at every step the Knight has 8 choices to choose from. Suppose, the Knight has to take k steps and after taking the Kth step the knight reaches (x,y). There are 8 different positions from where the Knight can reach to (x,y) in one step, and they are: (x+1,y+2), (x+2,y+1), (x+2,y-1), (x+1,y-2), (x-1,y-2), (x-2,y-1), (x-2,y+1), (x-1,y+2). If the probablities of reaching these 8 positions after k-1 steps is already known then, the final probablity after k steps will simply be equal to the (probability of reaching each of these 8 positions after K-1 steps)/8.

Here we are dividing by 8 because each of these 8 positions has 8 choices and position (x,y) is one of the choices.

For the positions that lie outside the board, we will either take their probabilities as 0 or simply neglect it.

Since we need to keep track of the probabilities at each position for every number of steps, we need Dynamic Programming to solve this problem. We are going to take an array dp[x][y][steps] which will store the probability of reaching (x,y) after (steps) number of moves.

Base case: if the number of steps is 0, then the probability that the Knight will remain inside the board is 1

II. Algorithm Design

Following is Dynamic Programming algorithm

- Define direction vectors for the knight.
- Take an array dp[N, N, steps + 1] which will store the probability of reaching (x,y) after (steps) number of moves.
- Base case: if the number of steps is 0, then the probability that the Knight will remain inside the board is 1
- Take the position (x, y) after s number of steps.
- Take prob = 0.0 then check for each position reachable from (x, y) using the direction vectors and store it in a new position (nx, ny).
- Check if this new position (nx, ny) is inside of the chessboard, if yes then add dp1[nx][ny][s - 1] / 8.0 to prob.
- Store the prob in dp[x][y][s].
- Keep repeating for the given number of steps.
- Required probability is stored as dp[start_x][start _y][k], where,start_x and start_y are the starting position of knight and k is the number of moves

III. Algorithm Analysis

Time complexity:

O(NxNxK), where N is the size of the board and K is the number of steps

Space complexity:

O(NxNxK), where N is the size of the board and K is the number of steps

IV. Conclusion

We can observe that by Dynamic Programming Approach we can easily solve the given problem and get better time-complexity.

V. References

1.https://www.geeksforgeeks.org/dynamic-programming/

APPENDIX

Code for implementation of this paper is given below:

```cpp
#include <bits/stdc++.h>
using namespace std;

int N = 0;

int dx[] = {1, 2, 2, 1, -1, -2, -2, -1};
int dy[] = {2, 1, -1, -2, -2, -1, 1, 2};

bool inside(int x, int y)
{
    return (x >= 0 and x < N and y >= 0 and y < N);
}

double findProb(int start_x, int start_y, int steps)
{
    double dp[N][N][steps + 1];

    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            dp[i][j][0] = 1;

    for (int s = 1; s <= steps; ++s)
    {

        for (int x = 0; x < N; ++x)
        {
            for (int y = 0; y < N; ++y)
            {
                double prob = 0.0;

                for (int i = 0; i < 8; ++i)
                {
                    int nx = x + dx[i];
                    int ny = y + dy[i];

                    if (inside(nx, ny))
                        prob += dp[nx][ny][s - 1] / 8.0;
                }

                dp[x][y][s] = prob;
            }
        }
    }

    return dp[start_x][start_y][steps];
}

int main()
{
    int K;
    int x, y;
    cout << "Enter the size of chessboard ";
    cin >> N;
    cout << "Enter the number of steps ";
    cin >> K;
    cout << "Enter the space-separated position of knight ";
    cin >> x >> y;
    cout << findProb(x, y, K) << endl;

    return 0;
}
```

Listing 1. Code for this paper