

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
TAIKOMOSIOS INFORMATIKOS KATEDRA

DISKREČIOSIOS STRUKTŪROS (P170B008)
KURSINIS DARBAS
Užduoties Nr. B01

Atliko: IFF-7 / 9 gr. studentas

Justinas Garipovas

Priėmė: lekt. *Audrius Nečiūnas*

KAUNAS

1. Užduotis (nr. 10 iš C lygio)

Šachmatų lentoje liko „sužeistas“ karalius ir žirgai. Reikia, kad visi jie sueitų į vieną laukelį. Kadangi karalius „sužeistas“, jį ten turi „nunešti“ žirgas. Sueiti reikia kuo greičiau (kad reikėtų kuo mažiau ėjimų). Jei žirgų skaičius būtų lygus nuliui, „sueiti“ reikėtų į jo stovimą laukelį ir tai būtų padaryta per nulį ėjimų (iškart).

2. Užduoties analizė

Atlikti užduočiai privalu su tam tikru kelio radimo algoritmu surasti greičiausią kelia iš vienos vietos šachmatų lentoje į kita atkartojant žirgo judesius. Mūsų užduotis reikalauja surasti mažiausiai ėjimų kiekį nuo taško A iki taško B dėlto šioje užduotijje naudoju A* kelio paieškos algoritmą,

Po kelio suradimo pasinaudojant A* algoritmu, yra apskaičiuojamas ėjimų skaičius, ir pagal tai yra nusprendžiamas žirgas kuris eis gelbėti karaliaus.

Tuo atveju kiti žirgai keliauja iškart į finišo lauką.

3. Programos tekstas

Gali būti rastas čia:

https://github.com/JustinasGaripovas/GT_LAB

```
public static void Main(string[] args)
{
    InitBoard();

    Node kingNode = board[8, 8];
    Node finishNode = board[0, 0];

    List<Knight> allKnights = new List<Knight>();

    int countOfKnights = getInput();
```

```

        Console.WriteLine(countOfKnights);

        Random rnd = new Random();

        for (int i = 1; i <= countOfKnights; i++)
        {
            allKnights.Add(new Knight(board[rnd.Next(0, i), rnd.Next(0, i)],
ConsoleColor.Blue, i + " knight"));
            Console.WriteLine();
        }

        if (countOfKnights == 0)
        {
            Console.WriteLine("Nera zirgu ");
        }
        else
        {
            calculations(allKnights, kingNode, finishNode);
        }
    }

    public static Path ShowMoves(Node start, Node end)
    {
        ResetBoard();

        Path currentPath = new Path(start);

        currentPath.start = start;
        currentPath.end = end;
    }

```

```

stack = new List<Node>();

start.score = heuristic(start, end);
stack.Add(start);

while(stack.Count > 0)
{
    Node currentNode = ReturnMin();

    if(currentNode.x == end.x && currentNode.y == end.y)
    {
        break;
    }

    currentNode.open = false;
    currentNode.seen = true;

    stack.Remove(currentNode);

    List<Node> neighborsList = new List<Node>();

    neighborsList.Add(BorderPatrol(-2, 1, currentNode.y, currentNode.x));
    neighborsList.Add(BorderPatrol(-2, -1, currentNode.y, currentNode.x));
    neighborsList.Add(BorderPatrol(2, 1, currentNode.y, currentNode.x));
    neighborsList.Add(BorderPatrol(2, -1, currentNode.y, currentNode.x));
    neighborsList.Add(BorderPatrol(-1, -2, currentNode.y, currentNode.x));
    neighborsList.Add(BorderPatrol(1, -2, currentNode.y, currentNode.x));
    neighborsList.Add(BorderPatrol(-1, 2, currentNode.y, currentNode.x));

```

```

neighborsList.Add(BorderPatrol(1, 2, currentNode.y, currentNode.x));

foreach(Node neighbor in neighborsList)
{
    if (neighbor == null || !neighbor.open)
    {
        continue;
    }

    double temp_score = currentNode.step + heuristic(currentNode,
neighbor);

    if (neighbor.seen == false){
        stack.Add(neighbor);
    }
    else if (temp_score >= neighbor.step)
    {
        continue;
    }

    neighbor.parent = currentNode;
    neighbor.step = temp_score;
    neighbor.seen = true;
    neighbor.score = neighbor.step + heuristic(neighbor, end);
}
}

Node node = end;
while (node.parent != null)

```

```

    {
        board[node.y, node.x].step = -1;

        currentPath.stepCount++;

        node = node.parent;
    }

return
currentPath;
}

```

4. Testiniai pavyzdžiai

```

Node kingNode = board[8, 8];
Node finishNode = board[0, 0];

List<Knight> allKnights = new List<Knight>();

allKnights.Add(new Knight(board[0,8], ConsoleColor.Blue, "First knight"));
allKnights.Add(new Knight(board[15, 8], ConsoleColor.Yellow, "Second knight"));
allKnights.Add(new Knight(board[6, 4], ConsoleColor.Yellow, "Third knight"));
allKnights.Add(new Knight(board[4, 1], ConsoleColor.Yellow, "Forth knight"));

```

First knight

To	End	0	1	2	3	4	5	6	7	8	
0	4	0	0	0	2	0	0	0	0	0	0
1	0	0	3	0	0	0	1	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0	2
3	0	0	0	0	0	0	0	0	0	0	3
4	0	0	0	0	0	0	0	0	0	0	4
5	0	0	0	0	0	0	0	0	0	0	5
6	0	0	0	0	0	0	0	0	0	0	6
7	0	0	0	0	0	0	0	0	0	0	7
8	0	0	0	0	0	0	0	0	0	0	8

4

Second knight

To	End	0	1	2	3	4	5	6	7	8	
0	5	0	0	0	3	0	1	0	0	0	0
1	0	0	4	0	0	0	0	0	0	0	1
2	0	0	0	0	0	2	0	0	0	0	2
3	0	0	0	0	0	0	0	0	0	0	3
4	0	0	0	0	0	0	0	0	0	0	4
5	0	0	0	0	0	0	0	0	0	0	5
6	0	0	0	0	0	0	0	0	0	0	6
7	0	0	0	0	0	0	0	0	0	0	7
8	0	0	0	0	0	0	0	0	0	0	8

5

To	End	0	1	2	3	4	5	6	7	8	
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	2
2	0	0	0	0	0	0	0	0	0	0	3
3	0	0	0	0	0	0	0	0	0	0	4
4	0	0	0	0	0	0	0	0	0	0	5
5	0	0	0	0	0	0	0	0	0	0	6
6	0	0	0	0	0	0	0	0	0	0	7
7	0	0	0	0	0	0	0	0	0	0	8
8	0	0	0	0	0	0	0	0	0	0	9

2

To	End	0	1	2	3	4	5	6	7	8	
0	6	0	0	0	0	0	0	0	0	0	0
1	0	0	5	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0	2
3	0	4	0	0	0	0	0	0	0	0	3
4	0	0	0	3	0	0	0	0	0	0	4
5	0	0	0	0	0	2	0	0	0	0	5
6	0	0	0	0	0	0	0	0	0	0	6
7	0	0	0	0	0	0	1	0	0	0	7
8	0	0	0	0	0	0	0	0	0	0	8

6

Pradiniai
duomenis

Forth knight

To End

	0	1	2	3	4	5	6	7	8	
	4	0	2	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	1
	0	3	0	1	0	0	0	0	0	2
	0	0	0	0	0	0	0	0	0	3
	0	0	0	0	0	0	0	0	0	4
	0	0	0	0	0	0	0	0	0	5
	0	0	0	0	0	0	0	0	0	6
	0	0	0	0	0	0	0	0	0	7
	0	0	0	0	0	0	0	0	0	8

4

5. Išvados

Sprendžiant iš pateiktų testinių pavyzdžių, programa veikia teisingai.

Kadangi naudojamas A* algoritmas tai paieška vyksta trumpiau, nes yra atsižvelgiama į ieškomo lauko poziciją, dėlto yra neapžiurimi visi laukai.