Popescu Christian

[Email address]

Abstract

[Draw your reader in with an engaging abstract. It is typically a short summary of the document.   
When you’re ready to add your content, just click here and start typing.]

Notes on challanges

Algorithms

**History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Content** |  |
|  |  |  |  |

**Table of Contents**

[1 Introduction 3](#_Toc508404102)

[2 Level 1 3](#_Toc508404103)

[2.1 Snakes and Ladders: The Quickest Way Up 3](#_Toc508404104)

[2.1.1 The problem 3](#_Toc508404105)

[2.1.2 Notes on solution 3](#_Toc508404106)

[2.1.3 Current status 4](#_Toc508404107)

[3 Data Structure 4](#_Toc508404108)

[3.1 Queues 4](#_Toc508404109)

[3.1.1 Castle on the grid 4](#_Toc508404110)

[3.1.1.1 The problem 4](#_Toc508404111)

[3.1.1.2 Notes on solution 4](#_Toc508404112)

1. Introduction

The purpose of the document is to keep information about the challenges, analysis, solutions problems.

1. Algorithms
   1. Snakes and Ladders: The Quickest Way Up
      1. The problem
2. Markov takes out his [Snakes and Ladders](http://en.wikipedia.org/wiki/Snakes_and_Ladders) game and stares at the board, and wonders: If he had absolute control on the die (singular), and could get it to generate any number (in the range ) he desired, what would be the least number of rolls of the die in which he'd be able to reach the destination square (Square Number ) after having started at the base square (Square Number )?
3. **Rules**
4. Markov has total control over the die and the face which shows up every time he tosses it. You need to help him figure out the minimum number of moves in which he can reach the target square (100) after starting at the base (Square 1).
5. A die roll which would cause the player to land up at a square greater than 100, goes wasted, and the player remains at his original square. Such as a case when the player is at Square Number 99, rolls the die, and ends up with a 5.
6. If a person reaches a square which is the base of a ladder, (s)he has to climb up that ladder, and he cannot come down on it. If a person reaches a square which has the mouth of the snake, (s)he has to go down the snake and come out through the tail - there is no way to climb down a ladder or to go up through a snake.
   * 1. Notes on solution

* It is a classical single source shortest path.
* We construct a graph with an edge i->i+1 for each **i** between 1 and 99 and cost 1.
* We construct an edge for each Snake and Ladder at cost 0.
* We apply the Dijkstra Algorithm with 1 as start node
* To apply **Dijkstra Algorithm** an implementation of Min Priority Queue is necessary. Good implementation is with **Min Heap Queue**.
* The Min Heap Que should support the operation **Decrease Key** on a given element.
  + 1. Current status

Should be redone using min heap que from competitive programming example

* 1. Coin Change Problem
     1. The problem

You are working at the cash counter at a fun-fair, and you have different types of coins available to you in infinite quantities. The value of each coin is already given. Can you determine the number of ways of making change for a particular number of units using the given types of coins?

* + 1. Notes on solution

It is a classical Dynamic Programming problem. There is a trick, the simple algorithm could generate duplicate solution. So to eliminate duplicates solutions should be enumerated.

In my implementation I have an optimization problem. I use sets to implement solution memorization.

* + 1. Current status

I’ve got only 24/60 points. I have to optimize further the solution.

1. Data Structure
   1. Queues
      1. Castle on the grid

The problem

[Data Structures\Queues\castle-on-the-grid-English.pdf](Data%20Structures/Queues/castle-on-the-grid-English.pdf)

Notes on solution

There is a classical minimum search path algorithm in a graph.

To represent the grid as an array an object that translates grid to array index and back will be constructed.

Graph description: edges will be constructed between each vertex that are reachable vertically or horizontally.

Algorithm for one vertex. Go in each direction: up, down, left, right till the edge of the grid or an obstacle is reached, connecting the vertex.

Data structure used:

* Bidimensional array of Boolean to keep the status of the cells: allowed/forbidden.
* Graph stored as vector of vertices. Where vertices keep an adjacency list.

Step 1 – Construct graph

Step 2 – Run BFS on the graph