Design Document for problem statement 3

**Subrata Paitandi**

**Rwik Mukhopadhyay**

**Karan Agarwal**

**Sourbah Nair**

Change Log

|  |  |  |
| --- | --- | --- |
| Date | Changed By | Details |
| 14/03/15 | **Rwik/Subrata** | **Initial Draft** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Table Of Contents

|  |  |  |
| --- | --- | --- |
| 1. | Problem Statement |  |
| 2. | Assumptions |  |
| 3. | Technology Stack |  |
| 4. | System Design |  |
| 5. | Main principles: |  |
| 6. | Output |  |
| 7. | Exceptions |  |
|  |  |  |

## Problem Statement :

You are an apprentice at a premier Astronomical society. Your job is to work with an extremely efficient automated astronomer system. This system detects millions of new stars and planets in a day (24 hours) via the James Webb and Hubble telescopes and feeds them to you with a distance with respect to Earth along with other details. You need to build a very efficient and fast software that outputs the distance between any two stars.  You need to keep in mind that you are getting data for millions of stars every new day. And you would need to calculate the distances between the existing set and new set of stars daily. So over the period of time the data accumulated is going to be huge. But the design of your software should be able to withstand that. I.e. no reduction in the performance. You have resource constraints too. You are only allowed to use 200 MB of memory. Any more and your process will be killed by a ruthless system monitor process. But on Sundays you are granted 100% of the resources.

For simplicity we shall say that the 3D spatial co-ordinates (X,Y,Z) of the stars are made available to you. The distances X,Y,Z are relative to the Earth on the respective axis. Given the 3D spatial co-ordinates for any two stars (X1, Y1, Z1) & (X2, Y2, Z2) we give the below the formula to compute the distance between these two stars.

You are the main engineer of the Starship Enterprise. You have been asked to work with Spock to chart a course from one star to another in the galaxy.

You would need to stop at some stars on the way to refuel after every 10 light years, as the fuel in your space ship does not last beyond 10 light years.

You also need to avoid stars with hostile civilizations on planets circling it.

You can have a file system storing the following data for each star.

Star ID X Y Z CAN\_REFUEL IS\_HOSTILE

The CAN\_REFUEL column can be  Y / N value based on whether the star has the gas to refuel your spaceship. The IS\_HOSTILE column can be Y / N value based on whether the star is considered Hostile to visit because of the planets encircling it.

Given any source and destination stars as input, you will need to find an optimum path to traverse from source to destination stars considering all the above constraints.

Technology

This is a pure C++ problem and does not require anything else to code the business logic. You can use files for storage and retrieval.

Judging Criteria

1. Architecture and Design – 20%

2. Error Handling (For incorrect inputs, wrong parameters etc.) – 10%

3. Efficiency/Performance – 20%

4. Completeness of the solution – 30%

5. Constraint Handling – 10%

6. Profiling Tool – 10%

## Assumptions:

* Inputs should be provided in comma separated value format.
* In coordinate fields system assumes real numbers.
* In id Field System assumes int values.
* In fileds system assumes characters

## Technology Stack:

* C++
* STL

**System Design :**

DATA FILE

Menu

1. Provide csv File
2. Search Path
3. Quit

SEARCH ALGORITHM

Read 1000 records at each cycle

Print Path

Class Diagram 1

Class Diagram : 2

CSV interface

Abstract class

Calculate\_shortestpath()

Main client

CSV concrete class

Implimentation Class

Calculate\_shortestpath()

CSV Reader Factory

Get\_CSVobject()

CSV Utility class

File\_Reader()

**Main principles:**

Data is stored on disk in chunks (pages,blocks, allocation units) or in CSV File and the RAM reads 1000 unit records from the flat CSV file at a time. To optimise an algorithm for accessing external memory or File system , should minimise disk accesses. Once we load the data in to RAM, we will apply Quick Sort on the 1000 records. The complexity of quick sort for the worst case scenario will be O(n\*2) and average case will be O(logn). Once we sorted the 1000 records based on id, we can apply Binary search.

In first pass we will try to find the co-ordinates of the starting and ending  id. The complexity will be O(logn) where n signifies as number of records in the csv file. Hence total complexity will be as below,

Number of records in the CSV file is = N

Number of reads from the file is = R

Total complexity will be = R \* (O(logn) + O(n\*2))

Once we get the start and end co-or dinate, We will store it in local veriable. Now we load the data from the file based on the limited size (lets say 1000 record at a time) and apply Dijecktras Algorithm to find the shortest path to nearst co-ordinate from the start location and end location based on the fuel availability. We will store this two value in a Vector. On second  pass, we will try to find the nearst neighbor of the previous co-ordinate and similarly push it in to the back of the vector. Also during neghbour selection we will consider the fuel indicator and if there is any common co-ordinate that are nearest of both the location. Algorithms ends when we found a co-ordinate that are common neghbour of both the start and end location.

# Output

* Shortest path between two Star Ids .

### Exceptions:

//To be filled later

## Error Messages:

//To be filled later

| **Module** | **ID** | **Message Type** | **Message** |
| --- | --- | --- | --- |
|  |  |  |  |