河內塔問題系統規劃文件

目錄

1. 簡介

本文件旨在定義河內塔問題的解決方案，提供開發人員清晰的思路和指導，以實現一個有效的河內塔求解系統。

2. 問題描述

河內塔問題的目標是將一堆不同大小的圓盤從「三個塔」中起始塔上移動到目標塔上，並遵循以下規則：

- 每次只能移動一個圓盤。
- 大圓盤不能疊在小圓盤之上。

3. 解決方案概述

我們將開發一個河內塔求解系統，該系統將根據輸入的圓盤數目，計算出移動圓盤的步驟，並提供一個漸進式的解決方法。我們將使用「遞迴」的技巧來解決河內塔問題。核心思想是將大問題分解為更小的子問題，然後遞迴地解決這些子問題，最終組合成大問題的解。

大問題：將 n 個圓盤依盒內塔規則由大到小排序到目標塔上。
子問題：

1. 移動 n-1 個圓盤從起始針到暫存塔。
2. 移動最大的圓盤從起始針到目標塔。
3. 移動 n-1 個圓盤從輔助針到目標塔。

第一個子問題會在拆解成更多子問題。

子問題的「大問題」：將 n -1 個圓盤依盒內塔規則由大到小排序到子問題的「目標塔」上。
子問題的「子問題」：

- 移動 n-2 個圓盤從起始針到暫存塔。
- 移動最大的圓盤從起始針到目標塔。
- 移動 n-2 個圓盤從輔助針到目標塔。

…一直延伸子問題，

最後將最小的圓盤放到目標塔上。

4. 系統架構

前端：

河內塔求解系統只由前端 React 完成，將包含以下組件及功能：

- Header 組件
- 遞迴功能
- 結果顯示模組：
  - List 組件
  - Item 組件

後端：

- Node.js server

5. 模組說明

Header 組件：

1. 負責接收使用者輸入的圓盤數目。
2. 確認輸入參數的合法性，避免無效輸入。
3. 初始化河內塔的初始狀態，將圓盤放置在起始塔上。

遞迴功能：

1. 實現遞迴算法，解決河內塔問題。
2. 記錄每一步的移動過程，以便後續顯示。

結果顯示模組：

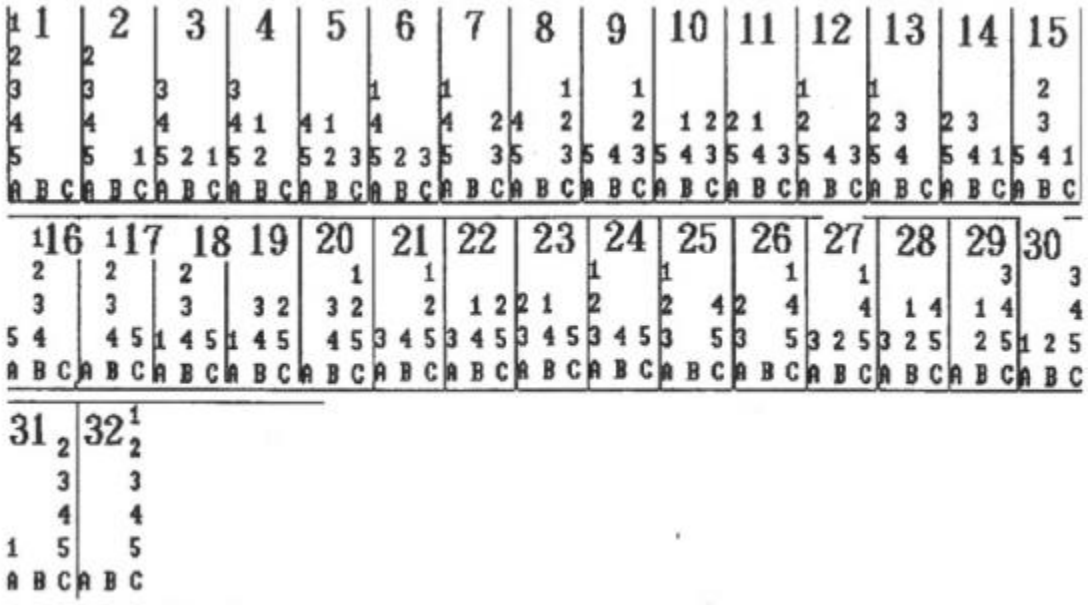
List 組件：顯示每個移動步驟的 Item 組件列表。

Item 組件：顯示特定移動步驟的詳細資訊。

Node.js 伺服器：

負責提供前端所需的公共資源，例如 React 程式碼、CSS 樣式和圖片等。

6. 範本



7. 測試計劃

1. 測試小規模盤數
2. 測試中等規模盤數
3. 測試大規模盤數
4. 測試輸入非法參數的情況

8. 開發環境

- 前端：React
- 後端：Node.js

9. 風險評估

- 遞迴可能導致堆疊溢位。

10. 參考資料

- 河內塔問題相關文獻和資源。

11. 程式架構

1. 數據格式

利用根組件的 state 來儲存河內塔層數及每一步的河內塔的狀態，並會用下列以下格式存取，以利於最終數據呈現。

```JSON
{
"tables":[
{
"A":[1,2,3,4,5]
"B":[]
"C":[]
}
{
"A":[2,3,4,5]
"B":[]
"C":[1]
}
...
]
"count":5 //表示輸入層數
}
```

2.設計思路

用 3 個 Array A, B, C 來表示 3 塔，數字 1, 2, 3….來表示圓盤，如果對圓盤有做移動動作則會在 tables 中心增一個元素來表示移動後的狀態，直到最後一步也就是河內塔完成，state 中就會有每一個動作後的數據，最後依造 tables 的數據即在畫面呈現。

3.河內塔算法

```
buildTowers(n, sourceTower, tampTower, targetTower) {
  const { tables, count } = this.state;
  let towerObj = JSON.parse(JSON.stringify(tables[tables.length - 1]));
  // eslint-disable-next-line
  if (n == 1) {
    towerObj[targetTower].unshift(towerObj[sourceTower].shift());
    towerObj.id = nanoid();
    tables.push(towerObj);
    // eslint-disable-next-line
    if (count == 1) {
      this.setState({ tables });
    }
    return;
  }

  this.buildTowers(n - 1, sourceTower, targetTower, tampTower);
  towerObj = JSON.parse(JSON.stringify(tables[tables.length - 1]));
  towerObj[targetTower].unshift(towerObj[sourceTower].shift());
  towerObj.id = nanoid();
  tables.push(towerObj);
  this.buildTowers(n - 1, tampTower, sourceTower, targetTower);

  this.setState({ tables });
}
```

這是 code 中最核心的河內塔算法，

第一步 將 state 的 tables 中的最後一筆元素取出，目的是為了用最新一筆的盒內塔狀態去推演下一步後河內塔狀態。
第二步 做河內塔推演，推演的方式在「解決方案概述」中有敘述，主要就是使用遞迴來將問題最拆解，如果要去將最底層的圓盤移動到指定塔上，則需要將他上方的用盤都先移走，在問題就會變成

1.如何將上放的圓盤放到暫存塔上

```
  this.buildTowers(n - 1, sourceTower, targetTower, tampTower);
```

2.之後如何將暫存塔上的圓盤一道目標塔上，

```
  this.buildTowers(n - 1, tampTower, sourceTower, targetTower);
```

一直遞迴直到完成為止。

12. 使用 CMD 呈現

如果要使用要使用 CMD 呈現，數據格式並不會有太大改變，只是會去設計一個橫列最多會去輸出幾個 step，去劃分到哪為止，一行一行呈現，畫完後再接這去輸出下一個橫列的 steps。