**Programming Assignment Report**

**工海三**

**B05602022**

**盧庭偉**

1. **演算法流程 (Algorithm Flow) <解釋程式的運作>**依照題目的hint，將grid model成一個2D array of node，其中node是我們自訂的class，包含一些計算overflow及跑Dijkstra演算法所需要的資訊，然後每個net跑一次Dijkstra演算法，跑完後將路上經過的邊demand + 1。  
   Pseudo code如下:  
   create graph  
   for all nets:  
    graph.initialize();  
    path = graph.dijkstra();  
    graph.update\_weight(path);  
    output(path);
2. **問題與討論 (Discussion) <討論實作中遇到的問題及疑問>**(1)自訂class: node  
   代表題目中表格的一格。包含四項資料:  
   short processor // 紀錄Dijkstra演算法中的processor  
   double distance // 紀錄. Dijkstra演算法中的距離  
   bool relaxed // 紀錄Dijkstra演算法中使否被看過  
   double edges[4] // 記錄四個邊界的wire數量  
   (2) std::map (RB tree)  
   為了在Dijkstra演算法中的Extract-Min有更好的performance，因此每當我們relax一個vertex時，我們就將它放進一個map(RB tree)，Extract-Min則是從這個map中pop出root。這樣Extract-Min的time complexity即可由linear search的下降為  
   (3) std::stack  
   在跑完Dijkstra演算法後，沿著終點往回推到起點，沿路把經過的vertices放到一個stack裡，再回傳給主程式。
3. **問題與討論 (Discussion) <討論實作中遇到的問題及疑問>**

如同題目所說，我們對edge weight的定義會影響到path的選擇，也就會影響到最終overflow的數量。  
由於overflow的計算是「超過capacity後多一條 + 1」，也就是說低於capacity前有幾條都不影響，超過capacity後超過多少都一樣多一條 + 1。因此我們要限制的條件主要會放在接近capacity的附近。所以我們很自然想到了sigmoid function，定義如下(定義z = demand/capacity):  
  
sigmoid function符合了我們想要的條件，在低於capacity的地方沒有明顯限制，因為都不算overflow；高於capacity的限制也不會隨著demand有明顯的增加因為都一樣是多一條overflow + 1。  
比較不同edge weight定義的overflow差別:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 4x4 | 5x5 | 10x10 | 20x20 | 60x60 |
| F1 | 0 | 0 | 11 | 2267 | 128119 |
| F2 | 0 | 0 | 13 | 2128 | 141422 |
| F3 | 0 | 0 | 11 | 5806 | 180412 |
| sigmoid | 0 | 0 | 8 | 14 | 52533 |

F1 = log(demand / capacity + 1)

F2 = demand / capacity

F3 =