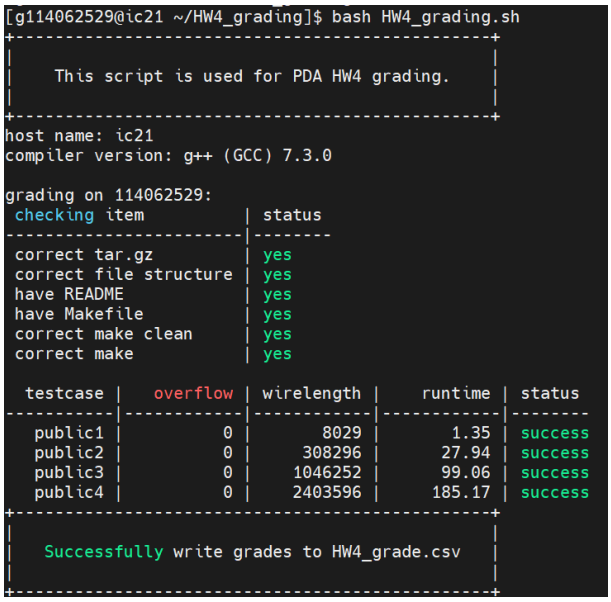
一、 執行結果



二、 程式碼詳細說明

1. 用 vector<vector<pair<int, int>>> edgesUse 記錄各邊的 demand，對於座標(x,y)，edgesUse[x][y].first 通向座標(x+1,y)，edgesUse[x][y].second 通向座標(x,y+1)。
2. Initial Stage

   所有 net 的兩個 L-shaped path 都放置 0.5 的 demand

   計算所有邊的 cost

   將所有 net 放到其總 cost 較小的 L-shaped path

   計算更新所有邊的 cost

   重複以下 t 次，直到結果不再變好

   計算所有 net 上的最大 cost，依序從大排到小

   對於所有 net，直到該 net 沒有 overflow

   檢查搬動到另一邊 L-shaped path 的 cost 是否有變小

   若有，搬過去並更新 cost
3. Main Stage

   重複以下 t 次，直到結果不再變好

   計算所有 net 上的最大 cost，依序從大排到小

   對於所有 net，直到該 net 沒有 cost 足夠小

   Rid-up 該 net 再使用 A*-search routing 來 re-route

4. 兩階段計算 overflow cost 的公式都是 1 + h / (1 + exp(-k * (d - s)))。A*-search routing 中 distance cost 的公式則是 max(|x0 - x1|, |y0 - y1|)。
5. 使用 priority_queue<Node, vector<Node>, greater<Node>> pq 來實作 A*-search routing，Node 為自定義的結構方便我們計算 cost 以及找到目的地後進行 trace back。具體作法為，從起點開始將相鄰的點放入 pq，pq 會依照 cost 來由小到大排序，運作方式就是加入 cost 的 BFS 演算法，不斷擴張尋找直到找到終點。
6. 遍歷多種參數組合來尋找結果最好的參數。

三、 是否有 rip-up 以及 re-route

1. 有，在 main stage 會 rip-up 以及使用 A*-search routing 來 re-route。
2. 決定是否 rip-up 以及 re-route 的標準為，將所有 net 依照他的最大 cost 由大排到小，然後依序處理。

四、 學到什麼/遇到的問題

原先想使用 history base 的 cost，但實作後發現結果沒有很好，其公式內有太多參數我不確定該怎麼調，於是改成先使用最簡單的 cost，反而能有很好的結果，推測可能是因為此次側資是簡化過後的版本，所以反而不需要使用太複雜的公式。

在撰寫初期發現在程式碼中需要不斷遍歷 net，會有大量重複的程式碼，會讓整體程式碼變得冗長，因此促使我學到 template<typename Func>的方式將程式碼包裝起來，大幅增加程式碼可讀性。

五、 AI 工具使用紀錄

1. 使用 chatgpt 來協助我完成這項作業，內容如下。
2. 協助我了解 template<typename Func>的使用方式。
3. 協助我發現程式碼中誤植的小錯誤。
4. 協助我了解 Node 中排序程式碼的寫法。
5. 協助我思考為何使用 history base 的 cost 效果會不好以及如何解決。