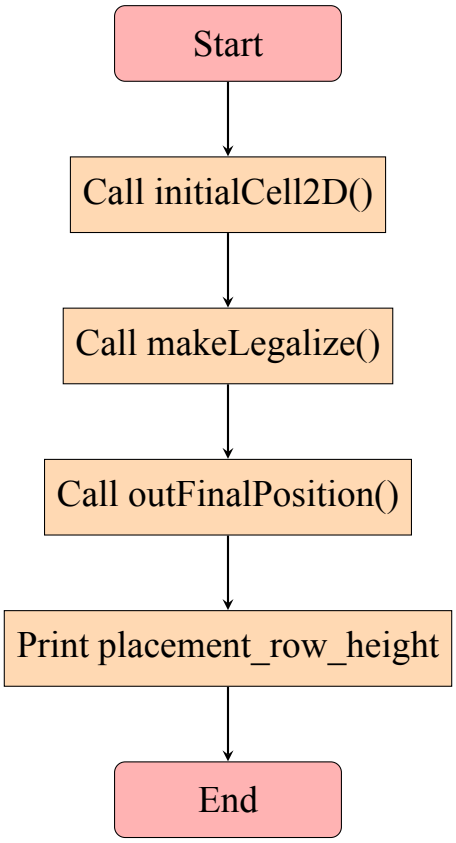
Lab3
Optimizer and Legalizer

313510164　陳緯亭　電子碩一

November 20, 2024

Contents

1 Flowchart


Start
Call initialCell2D()
Call makeLegalize()
Call outFinalPosition()
Print placement_row_height
End

2 Pseudo code

Algorithm 1: isLegal

Input: $target\_cell$, die, $placement\_row\_height$, $cells\_2d$

Output: Boolean value indicating legality

```
1  if target_cell.y_top > die.y_top or target_cell.y_bottom < die.y_bottom then
2      return false // Out of die bounds;
3  for y in [target_cell.y_bottom, target_cell.y_top) step placement_row_height
     do
4      if y ∈ cells_2d then
5          row_map ← cells_2d[y];
6          lower_bound ← row_map.lower_bound(target_cell.x_left);
7          if lower_bound ≠ row_map.begin() and
            prev(lower_bound).x_right > target_cell.x_left then
8              return false // Overlap with left cell;
9          if lower_bound ≠ row_map.end() and
            lower_bound.x_left < target_cell.x_right then
10             return false // Overlap with right cell;
11 return true // Placement is legal;
```

Algorithm 2: makeLegalize (Simplified)

Input: each pair in merged_vec

Output: None

for *each pair in merged_vec* **do**
- Initialize $is\_down \leftarrow$ true and extract ff_merged;
- Create $cell$ from ff_merged;
- **for** *each* ff *in* $ff\_merged.ff\_list$ **do**
  - Remove ff from data structures;
- **while** *not isLegal(cell)* **do**
  - **if** *leftRightLegalize(cell) is false* **then**
    - **break**;
  - **if** *queues are not empty* **then**
    - Adjust $cell.x_{left}$ and $cell.x_{right}$ using queue values;
  - **else**
    - Adjust $cell.y_{bottom}$ and $cell.y_{top}$ based on is_down;
    - Toggle is_down if row bounds are exceeded;
- Add $cell$ to data structures and output its position;
- Clear queues and temporary data;

Algorithm 3: leftRightLegalize

Input: $target\_cell$

Output: Boolean value is_push

```
is_push ← true;
Find left_cell;
Find right_cell;
if isLegal(left_cell) or isLegal(right_cell) then
    Find target location according to left or right cell;
if isLegal(right_cell) then
    Adjust target cell's x position to right cell's x;
    Adjust target cell's right x position to right cell's right x;
    is_push ← false;
for y ∈ [target_cell.y_bottom, target_cell.y_top) step placement_row_height do
    if y ∈ cells_2d then
        row_map ← cells_2d[y];
        lower_bound_it ← row_map.lower_bound(target_cell.x_right);
    if lower_bound_it is not start then
        Find previous and push cells;
    if lower_bound_it is not end then
        Find next and push cell;
return is_push;
```

3 Time complexity analysis

- `makeLegalize( )` 最糟糕的時間複雜度為 $O(n^2)$，其中 n 為 `merged_vec` 的大小。在最糟糕的情況下，每個 cell 都需要進行 legalize，而每次 legalize 都需要檢查所有的 cell 是否合法，因此時間複雜度為 $O(n^2)$。

- `isLegal( )` 的時間複雜度為 $O(h)$，其中 h 為 target_cell 的高度。在最糟糕的情況下，`target_cell` 的高度為 die 的高度，因此時間複雜度為 $O(h) =$ `height/placement_row_height`。

- `leftRightLegalize( )` 的時間複雜度為 $O(h)$，因為只需要找到左右相鄰的

cell 並檢查是否合法。每一次 Map 的操作複雜度 Map operations: $O(\log m)$、Queue operations:$O(1)$。所以全部的 operations 為 $O(h \log m)$。

- `initialCell2D( )` 的時間複雜度是 $O(n * h)$。對於每個 cell，對於每個 row 插入 `cell_2d` ，總共是 h rows。
- `removeCell2D( )` 的時間複雜度為 $O(h)$，因為只需要刪除 cell 並更新 `row_map`。$h =$ `cell height/placement_row_height`
- 全部的空間複雜度為 O(n)，其中 n 為 cell 的數量：
  `cells_map`: O(n)
  `cells_2d` O(n)
  Queue storage during legalization: O(n)

主要的複雜度瓶頸在於：

1. 最差情況下的時間複雜度是 $O(n^2)$，主要來自 makeLegalize( ) 函數
2. 大量使用了 map 操作，每次操作都有 O(log n) 的開銷
3. 二維單元格存儲結構 `cells_2d` 的操作需要額外的對數時間

4 Special features of my program

1. 使用了二維單元格存儲結構 `cells_2d` 來優化查詢
2. 使用了 map 來優化查詢，通過名稱、x 座標、y 座標都能快速訪問
3. 使用不同維度的數據結構來優化不同類型的查詢
4. 使用右端點 (x_rt, y_rt) 優化邊界檢查
5. 實現了逐步移動而不是直接跳躍
6. 支持雙向探索最優位置
7. 隊列基礎的推進機制，防止重複處理同一單元格，維護移動的方向信息。

5 Feedback

總結來說，這是一個非常具有挑戰性的問題：

設計挑戰:

1. 需要處理複雜的幾何約束
2. 要求高效的算法實現
3. 一開始有試著推推看，但是推動之後，還可能會壓到其他 cell (特別是 fixed cell)，有夠麻煩。
4. 必須考慮多個優化目標，讓 cell 位置盡可能接近原位置

實現挑戰:

1. 需要精心設計數據結構
2. 要求高效的算法實現
3. 必須處理大量邊界情況，讓其不要 Overlap

6 Conclusion

6.1 Run( )

這是觸發算法流程的主要入口點：

- 使用 `initialCell2D` 初始化細胞布局。
- 使用 `makeLegalize` 進行布局的合法化。
- 使用 `outFinalPosition` 輸出最終位置。

6.2 makeLegalize( )

此函數通過合併的細胞進行迭代，逐步合法化它們的布局：

- 調整細胞的 `x` 和 `y` 坐標。

- 如果細胞與另一個細胞重疊，則嘗試調整位置並重新檢查合法性。
- leftRightLegalize 和隊列 (q_cell 和 q_is_left) 控制迭代調整過程。
- 該算法旨在通過兩階段系統來調整細胞，以避免重疊：第一階段處理左側鄰居（QL），第二階段處理右側鄰居（QR）。

6.3 legalAlg( )

此函數實現了一個合法布局算法，檢查與相鄰細胞的重疊情況，並嘗試將細胞向左或向右移動以避免重疊。它利用兩個隊列（QL 用於左側，QR 用於右側）執行類似廣度優先搜索的調整。對於每個細胞，算法會移動其鄰居（向左或向右），直到不再需要進行進一步調整。

6.4 outputFile( )

此函數將每個細胞的最終位置寫入文件，並記錄其 x 和 y 坐標。

6.5 removeCellFromMultimap( )

通過匹配 x 坐標和 cellId 從 2D 地圖（cells_by_x 或 cells_by_y）中移除細胞。

6.6 addCell2D( )

將細胞添加到 2D 網格中，y 坐標的範圍由細胞的高度決定。

6.7 outFinalPosition( )

將所有細胞的最終位置寫入指定的輸出文件，並記錄移動的細胞數量及其新位置。

6.8 Key algorithm (Legalized Layout)

1. 兩階段合法化
    - 第一階段（QL）側重於當前細胞左側的細胞，若有重疊，嘗試將其調整到左側。

- 第二階段（QR）側重於當前細胞右側的細胞，若檢測到重疊，則將其移動到右側。

2. 重疊檢測重疊檢查通過確保沒有細胞的位置使得其邊緣與相鄰細胞的邊緣相交來進行。如果檢測到重疊，算法會調整相鄰細胞的 x 坐標。

3. 基於隊列的調整使用隊列（QL 或 QR）將調整傳播到受重疊影響的相鄰細胞。一旦鄰居的位置信息被調整，它會重新添加到隊列中以進行進一步處理。

7 Consideration

- 效率: 該算法使用類似廣度優先搜索的方法，通過隊列逐步調整細胞。這確保了重疊的細胞能夠按順序移動，避免了不必要的迭代。

- 最終位置輸出: 所有細胞調整完成後，最終位置將被保存到文件中。還會記錄移動的細胞數量，這對於調試或性能跟蹤非常有用。

- 錯誤處理: 代碼處理了如文件打開失敗和隊列下溢等錯誤，這對於增強穩健性至關重要。

8 Improvement

- 優化細胞移除: removeCellFromMultimap 中的細胞移除操作可以進行優化，以減少不必要的迭代，尤其是在細胞數量較大時。

- 並行處理: 對於較大的數據集，可以通過並行化左側和右側合法化步驟來減少計算時間，因為這些步驟可能是獨立的。

- 合法化策略: 兩階段的合法化方法（左和右）雖然簡單，但可能並不適用於所有情況。可以探索更複雜的技術，例如模擬退火，以處理更複雜的布局。

9 Performance

```
Elapsed time: 35.94 seconds
./Evaluator testcase/testcase1_16900.lg testcase/testcase1_16900.opt testcase/testcase1_16900_post.lg
+----------------+--------------+----------+--------------+---------------+
|      Cost      |      -       |  Weight  |    Value     | Percentage(%) |
+----------------+--------------+----------+--------------+---------------+
|   Move Times   |     0.00     | 10000.00 |     0.00     |    0.00(%)    |
| Total Distance | 748018320.00 |   1.00   | 748018320.00 |   100.00(%)   |
|     Total      |      -       |    -     | 748018320.00 |   100.00(%)   |
+----------------+--------------+----------+--------------+---------------+
```

Fig. 1: testcase1_16900

```
Elapsed time: 571.81 seconds
./Evaluator testcase/testcase1_ALL0_5000.lg testcase/testcase1_ALL0_5000.opt testcase/testcase1_ALL0_5000_post.lg
+----------------+---------------+----------+---------------+---------------+
|      Cost      |       -       |  Weight  |     Value     | Percentage(%) |
+----------------+---------------+----------+---------------+---------------+
|   Move Times   |     0.00      | 50000.00 |     0.00      |    0.00(%)    |
| Total Distance | 2635175850.00 |  20.00   | 52703517000.00 |   100.00(%)   |
|     Total      |       -       |    -     | 52703517000.00 |   100.00(%)   |
+----------------+---------------+----------+---------------+---------------+
```

Fig. 2: testcase1_ALL0_5000

```
Elapsed time: 2.93 seconds
./Evaluator testcase/testcase2_100.lg testcase/testcase2_100.opt testcase/testcase2_100_post.lg
+----------------+--------------+----------+----------------+---------------+
|      Cost      |      -       |  Weight  |     Value      | Percentage(%) |
+----------------+--------------+----------+----------------+---------------+
|   Move Times   |     0.00     | 50000.00 |      0.00      |    0.00(%)    |
| Total Distance | 773290680.00 |  20.00   | 15465813600.00 |   100.00(%)   |
|     Total      |      -       |    -     | 15465813600.00 |   100.00(%)   |
+----------------+--------------+----------+----------------+---------------+
```

Fig. 3: testcase2_100