Programming Assignment #3

資料結構：

這次的程式設計中使用了以下的結構

1. struct Point：

   表示一個座標點 (x, y)，支援比較（可當 std::map 或 std::set 的 key），並提供建構子與相等/不等的判斷。

```
16  struct Point {
17      int x, y;
18      // 加上這個才能當作 std::map 的 key 使用
19      bool operator<(const Point& other) const {
20          return (x < other.x) || (x == other.x && y < other.y);
21      }
22
23      bool operator==(const Point& other) const {
24          return x == other.x && y == other.y;
25      }
26
27      bool operator!=(const Point& other) const {
28          return !(*this == other);
29      }
30
31      Point();
32      Point(int x ,int y);
33  };
```

2. struct Net：

   表示一條網路連線，包含索引、起點與終點 Point、傳輸損耗值 loss，以及實際走線的路徑 path（點的序列）。

```
35  struct Net {
36      int index;
37      Point src, dst;
38      float loss;
39      std::vector<Point> path;
40
41      Net(int index, int srcX, int srcY, int dstX, int dstY);
42  };
```

3. struct Node：

   用於路由演算法的節點資訊，包含座標 p、目前成本 g、總成本 f（如 A*）、方向 dir，並支援優先隊列中的比較（依 f 值遞增排序）。

```cpp
struct Node {
        Point p;
        int g = 0;
        float  f = 0;
        int dir = -1;
        bool operator<(const Node &o) const { return f > o.f; }
    };
```

4. struct PointHash：

   自訂的雜湊函數，讓 Point 可以被用作 unordered_map 或 unordered_set 的 key。

```cpp
struct PointHash {
    std::size_t operator()(const Point& p) const noexcept {
        return (static_cast<std::size_t>(p.x) << 20) ^ p.y;
    }
};
```

演算法：

可以說明三個部分

1. 主要路徑演算法：AStar

   因為現在的問題較為單純，每個線段都是只有兩個點，所以在這個方面上 AStar 是一個滿優秀的解決方法，最主要影響 AStar 的表現是他的 heuristic function 的細節。

2. Heuristic function

   用曼哈頓距離估算傳播損耗，並加入轉彎次數的額外成本，之後兩者都乘上對應的權重與比例，最後再乘上全域比例調整搜尋強度，最後這個全域係數就是個神祕的數值了，需要多次的調整。

3. 後續優化

   後續我去將前 10%的高 loss 的線段去做從新的佈局，看看是否可以得到更好的結果，然後有做一個小小的優化就是在嘗試一定的次數後我將我的 crossing loss 的成本給提高了 1.5 倍，讓他在後面嘗試的時候不要有太多的 crossing loss。