1.On the aspect of heuristic, we realize that our old heuristic underestimates too much. The old heuristic is from a math model that includes too many cases that are almost impossible to happen in the maps being analyzed, leading to significant amount of tile visiting and cost of time. Thus we try to get rid of cases that is not possible and be more greedy in heuristic. We eventually come up with new heuristic

$$\textbf{Overall Cost} = h(n) = \textbf{LeastDistance} * 2^{\frac{Difference\ of\ Height}{LeastDistance}}$$

where LeastDistance is defined by least steps from current tile to end tile. Consistency of this heuristic is proven in report of part 1. To be more cautious, we scale down the whole heuristic by multiplying with 0.8.

2. On the aspect of improving A* algorithm, we implement the bi-direction search. It runs two simultaneous search :

Forward search form source/initial vertex toward goal vertex

Backward search form goal/target vertex toward source vertex

And the search terminates when two sub-search has intersection point in their visited point. Based on the version of A* algorithm, we check if both visited set contains same point after every time we pulling out point from both open set for two bi-direction search .

```
Point isIntersecting(ArrayList<Point> s_closed,ArrayList<Point> e_closed){
  Point intersectPoint = new Point(-100, -100);
  for(int i = 0; i < s_closed.size();i++)
  {
    if(e_closed.contains(s_closed.get(i))) {
      intersectPoint =  s_closed.get(i);
      return intersectPoint;
    }
  }
  return intersectPoint;
}
```

We can track back the path by adding the 'parent' field in the node of our search tree and we start from the intersection track back to start point of search starting from start point. Then we track back the path from the intersection track back to end point of search starting from end point.

```
Cell current;
int intersectI = intersectPoint.x;
int intersectJ = intersectPoint.y;
if (s_closed.contains(intersectPoint)) {
    //Trace back the path
    current = s_grid[intersectI][intersectJ];
    while (current.parent != null) {
        path.add(current.point);
        current = current.parent;
    }
    // add startNode;
    path.add(current.point);
}
Collections.reverse(path);
if (e_closed.contains(intersectPoint)) {
    //Trace back the path
    current = e_grid[intersectI][intersectJ];
    while (current.parent != null) {
        path.add(current.point);
        current = current.parent;
    }
    // add startNode;
    path.add(current.point);
}
return path;
```

The reason why we use bi-direction approach is that if we execute two search operation then the complexity would be O(b^{d/2}) for each search and total complexity would be O(b^{d/2}+b^{d/2}) which is far less than O(b^d).