NOMBRE: Felipe Baitelman

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 — Estructuras de Datos y Algoritmos

Informe Tarea 1

1 Introducción

Es necesario plantear desde un comienzo que la tarea entregada no funciona correctamente, y que solo resuelve ejercicios muy simples, en particular, el test básico de 1x1 (también resuelve el de 2x1 pero sin colorear). No obstante, se procede a incluir en este informe tanto lo que efectivamente se desarrollo para buscar soluciones, como lo que que se planificó y no logro llevarse a cabo.

2 Podas

2.1 Poda 1

Descripción: El backtracking implementado buscaba recorrer todos los caminos hasta que se encontrara una conexión que partiera desde un core y encontrara su core de destino. Al comenzar desde un core, se recorría cada zona utilizando fores y siempre que se encontraba un camino posible por el cual avanzar se realizaban conexiones, hasta que eventualmente esta conexión resultara en la conexión final con el core respectivo. Para reducir estos tiempos, la poda se encargaba de realizar primero una búsqueda inteligente dentro de la zona revisando si había un core del mismo color, lo que permitía que el tiempo de búsqueda disminuyera considerablemente. Esto se observa en la función **SolveMap**, la cual utiliza una variable local del tipo *int* llamada k, la que variaba entre 0 y 1 dependiendo de si se encontraba o no el core de destino inteligentemente.

Análisis de Rendimiento: La poda claramente no influye en el comportamiento asintótico que tiene un backtracking simple sin ella. El backtracking por si solo es $\mathcal{O}(n!)$. Eventualmente, este backtracking se puede realizar complete sin nunca caer en la condición de la poda (o caer solo en el caso final, cuando ya no se puede encontrar ningún camino a menos que nos conectemos con un core del mismo color), por lo que en el peor de los casos el algoritmo tardará lo mismo con o sin poda. No obstante, la poda si puede contribuir a disminuir el tiempo promedio real que tarda el algoritmo notablemente, encontrando soluciones mucho antes que lo que se demora cada core en revisar todos sus caminos posibles.

3 Heurísticas

3.1 Heurística 1

Descripción: La función **SolveMapRecursive** buscaba resolver el mapa de la ciudad utilizando un backtracking simple. En ese sentido, es la heurística más simple de todas, un backtracking que busca una solución haciendo recursiones y retornos reiteradas veces. El método trataba de juntar pares de edificios como tuplas, implementadas como el **struct** *BuildingTuple*, y guardar estas tuplas en un stack. De esta forma, el método

de resolución consistía en resolver el camino para un core, pasar al siguiente y así sucesivamente. Si eventualmente un core no encontraba una solución factible, nos devolvíamos y modficabamos el camino anterior para el core previo. Estaba planeado implementar esta función en conjunto con la función **SolveMap**, para así incluir la Poda 1 y resolver el problema en un tiempo menor.

Análisis de Rendimiento: El algoritmo tiene un comportamiento asintótico de $\mathcal{O}(n!)$ siendo n la cantidad total de links distintos que se pueden hacer. Esto se debe a que fijar ciertos links no asegura su permanencia más adelante. Eventualmente hay que desconectar todos y volver a hacer las conecciones, luego para cada uno de los n posibles links habrán n posibles posiciones, en funciónd e como se fijan los demas links en el mapa. El problema guarda bastante semejanza con el Ciclo de Hamilton.

4 Conclusiones

La capacidad del programa de resolver problemas es sin duda limitada. A lo largo del trabajo se intentó generar un backtracking simple tal como se explica previamente, sin resultados positivos. Esto se debió, más que nada, a falta de experiencias con el lenguaje utilizado. El trabajo adecuado con la memoria utilizando C es crucial, lo que no favoreció el progreso a medida que se programaba. Errores del tipo *segmentation fault* aparecían una vez tras otra y hubo un momento en el cual seguir avanzando se hizo imposible. Aparecían *buildings* con índices no consistentes y el output del programa variaba según donde su ubicaba un *puts* o un *printf*. El enfoque de resolución falló y hacer cambios para seguir trabajando ya no era una solución viable. De este modo, el principal punto a mejorar para una futura tarea es seguir estudiando cuidadosamente C y entenderlo en profundidad para no volver a caer en estancamientos como el de esta tarea.