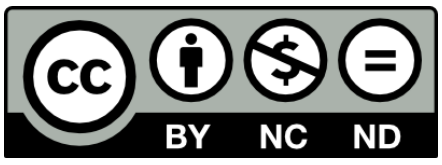
Algorítmica

Los Del DGIIM, `losdeldgiim.github.io`

Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada



Algorítmica

Los Del DGIIM, `losdeldgiim.github.io`

José Juan Urrutia Milán
Arturo Olivares Martos

Granada, 2023-2024

Índice general

1. Algoritmos Greedy **7**
1.1. Grafos 8

2. Relaciones de Problemas **11**

Introducción

El siguiente documento no es sino el mero resultado proveniente de la amalgación proficiente de un cúmulo de notas, todas ellas tomadas tras el transcurso de consecutivas clases magistrales, primordialmente —empero, no exclusivamente— de teoría, en simbiosis junto con una síntesis de los apuntes originales provenientes de la asignatura en la que se basan los mismos. Como motivación para la asignatura, introducimos a continuación un par de problemas que sabremos resolver tras la finalización de esta:

Ejercicio (Parque de atracciones)**.** Disponemos de un conjunto de atracciones

$$A_1, A_2, \ldots, A_n$$

Para cada atracción, conocemos la hora de inicio y la hora de fin. Podemos proponer varios retos de programación acerca de este parque de atracciones:

1. Seleccionar el mayor número de atracciones que un individuo puede visitar.
2. Seleccionar las atraciones que permitan que un visitante esté ocioso el menor tiempo posible.
3. Conocidas las valoraciones de los usuarios, $val(A_i)$, seleccionar aquellas que garanticen la máxima valoración conjunta en la estancia.

Ejercicio. Una empresa decide comprar un robot que deberá soldar varios puntos (n) en un plano. El software del robot está casi terminado pero falta diseñar el algoritmo que se encarga de decidir en qué orden el robot soldará los n puntos. Se pide diseñar dicho algoritmo, minimizando el tiempo de ejecución del robot (este depende del tiempo de soldadura que es constante más el tiempo de cada desplazamiento entre puntos, que depende de la distancia entre ellos). Por tanto, deberemos ordenar el conjunto de puntos minimizando la distancia total de recorrido.

Nociones de conceptos

A lo largo de la asignatura, será común ver los siguientes conceptos, los cuales aclararemos antes de empezar la misma:

- Instancia: Ejemplo particular de un problema.
- Caso: Instancia de un problema con una cierta dificultad.

Generalmente, tendremos tres casos:

- El mejor caso: Instancia con menor número de operaciones y/o comparaciones.
- El peor caso: Instancia con mayor número de operaciones y/o comparaciones.
- Caso promedio. Normalmente, será igual al peor caso.

Para notar la eficiencia del peor caso usaremos $O(\cdot)$, mientras que para el mejor caso, $\Omega(\cdot)$.

Diremos que un algoritmo es *estable* en ordenación si, dado un criterio de ordenación que hace que dos elementos sean iguales en cuanto a orden, el orden de stos vendrá dado por el primero se que introdujo en la entrada.

Ejemplo. Dado el criterio de que un número es menor que otro si es par, ante la instancia del problema: $1, 2, 3, 4$. La salida de un algoritmo de ordenación estable según este criterio será:

$$2, 4, 1, 3$$

Sin embargo, un ejemplo de salida que podría dar un algoritmo no estable sería:

$$4, 2, 3, 1$$

Los datos se encuentra ordenados pero no en el orden de la entrada.

Algoritmos de ordenación

A continuación, un breve reapso de algoritmos de ordenación:

- Burbuja es el peor algoritmo de ordenación.
- Si tenemos pocos elementos, suele ser más rápido un algoritmos simple como selección o inserción. Entre estos, selección hace muchas comparaciones y pocos intercambios, mientras que inserción hace menos comparaciones y más intercambios. Por tanto, ante datos pesados con varios registros, selección será mejor que insercción.
- Cuando se tienen muchos elementos, es mejor emplear un algoritmo de ordenación del orden $n \log(n)$.

1. Algoritmos Greedy

Los algoritmos Greedy son algoritmos que siempre toman la mejor decisión en cada momento, la cual sólo depende de datos locales. Por ejemplo, un algoritmo que busque recorrer n ciudades en el plano podría implementarse mediante la técnica Greedy cogiendo en cada paso la ciudad que se encuentre más cerca de la que estamos. Notemos que de esta forma no obtenemos la solución óptima al problema, aún habiendo tomado la solución óptima en cada caso.

Los algoritmos Greedy tienen como características:

- Se suelen utilizar para resolver problemas de optimización: búsqueda de máximo o mínimo.
- Toman decisiones en función de la información local de cada momento, tomando la mejor alternativa.
- Una vez tomada la decisión, no se replantea en el futuro.
- Son fáciles y rápidos de implementar.
- Los algoritmos Greedy no siempre proporcionan la solución óptima, como ya hemos podido comprobar.

Elementos de un algoritmo voraz:

- Conjunto de candidatos: representa al conjunto de posibles soluciones que se pueden tomar en cualquier momento.
- Conjunto de seleccionadas: representa al conjunto de decisiones tomadas hasta el momento.
- Función solución: determina si se ha alcanzado una solución al problema.
- Función de selección: determina el candidato que proporciona el mejor paso.
- Función de factibilidad: determina si es posible o no llegar a la solución del problema mediante el conjunto de seleccionados.
- Función objetivo: da el valor de la solución alcanzada.

Un algoritmo Greedy puede resolverse de la forma:

- Se parte de un conjunto de candidatos a solución vacío.
- De la lista de candidatos que hemos podido identificar, con la función de selección, se coge el mejor candidato posible.

- Vemos si con ese elemento podríamos llegar a constituir una solución, es decir, si se verifican las condiciones de factibilidad en el conjunto de candidatos.
- Si el candidato anterior no es válido, lo borramos de la lista de candidatos y nunca más se considera.
- Evaluamos la función solución (si no hemos terminado, seleccionamos otro candidato con la función de selección y repetimos el proceso hasta alcanzar una solución).

Normalmente, si conseguimos un algoritmo Greedy óptimo, es probable que sea el mejor algoritmo que resuelve el problema.

Problema de dar cambio

Normalmente, las máquinas que dan cambio están programadas de forma que nos lo devuelven con el mínimo número posible de monedas.

En este caso, el conjunto de candidatos es el conjunto de monedas que tiene disponible la máquina. El conjunto de seleccionados son las monedas que voy incorporando a la solución. Habremos encontrado una solución cuando hayamos llegado a la cifra que se pedía. Si superamos o no llegamos a la cantidad a devolver, nuestra solución no es factible.

Como criterio de selección inicial, podemos elegir la moneda de mayor valor que sea menor o igual que el precio a pagar.

¿Este algoritmo consigue la solución óptima?
Pues si el valor de las monedas corresponde con los billetes y monedas de euro en circulación sí; pero si tenemos otro sistema monetario, quizás no.

Problema de selección de programas

Tenemos un conjunto T de n programas, cada uno con tamaño $t_1, \ldots, t_n$ y un dispositivo de capacidad máxima C, seleccionar el mayor número de programas que pueden meterse en C.

1.1. Grafos

Ejemplo. Dados varios ficheros de libros (algunos parecidos pero con leves diferencias), también tenemos libros incluidos dentro de otros. El problema es quedarnos con un conjunto de libros suficientemente pequeño que no contemple libros repetidos (o muy similares).

Podemos llegar a una solución a este problema mediante un grafo, de forma que ciertos libros estén conectados si tienen una estructura similar (mayor que un cierto umbral). Después de conectar todos los nodos similares, sustituiremos la estructura de nodos conectados por un sólo nodo, que cumpla una cierta propiedad (menor tamaño, mayor similitud con el resto, ...).

Definición 1.1 (Grafo)**.** Un grafo es un conjunto de vértices (nodos) y de aristas (que unen los vértices).

- En el caso de que las aristas tengan un inicio y una final determinado (es decir, un sentido), decimos que tenemos grafos dirigidos. En contraparte, grafos no dirigidos.
- Si las aristas presentan un peso (un valor), hablamos de grafos ponderados.

Conceptualmente, todo lo siguiente está permitido (aunque su permisibilidad depende del prblema):

- Unir dos vértices con más de una arista.
- Unir un vértice consigo mismo.

Definición 1.2 (Grafo completo)**.** Decimos que un grafo es completo si dados dos vértices cualesquiera, existe un camino que los une. Es decir, si no hay puntos sueltos.

En los grafos dirigidos, sólo podremos avanzar en el sentido de las aristas y no al revés.

Definición 1.3 (Grafo plano)**.** Un grafo se llama plano si puede dibujarse en un plano o esfera (espacios homeomorfos) sin que se crucen sus aristas.

2. Relaciones de Problemas