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

IIC2133 — Estructuras de datos y algoritmos — 2020-2

Ayudantía 9

Pregunta 1

Dado un texto de n caracteres y un string de m caracteres:

- proponga un algoritmo para determinar si el string está en el texto en O(n).
- Con *regular expressions* (REGEX) se puede especificar búsquedas de strings de manera más general. Se puede especificar que una posición permita distintos caracteres, por ejemplo con este sintaxis: [aA]mig[aeo]s. Proponga una modificación al algoritmo anterior para agregar esta funcionalidad a la búsqueda.

Pregunta 2

A grandes razgos, cómo usarías DFS o BFS o dijkstra para resolver los siguientes problemas?

1. Dada una matriz binaria (de 1's y 0's) de tamaño nxn, cómo encontraría la cantidad de 'islas'? Es decir, la cantidad de componentes conexas distintas en el grafo no dirigido representativo.
2. Determina si un grafo dirigido es acíclico.
3. Eliminar, arbitrariamente, todos los ciclos, de un grafo dirigido, sin eliminar otras aristas.
4. Dado un conjunto de cursos con prerrequisitos, si no hay restricciones aparte de los requisitos, calcular la mínima cantidad de semestres necesarios para rendirlos todos. Se puede asumir que no hay ciclos en el grafo no dirigido representativo.

Solución

Pregunta 1

a)

Primero creamos un hash de un string. Lo importante de este hash es que sea incremental frente al desplazamiento del string, es decir, si quiero calcular el hash del string del mismo tamaño desplazado en uno a la derecha tiene que ser O(1) en complejidad independiente del tamaño del string original. Hay múltiples maneras de solucionar esto, la que propongo es usar tres operaciones binarias. Ojo que lo más importante para esta pregunta es entender lo que se mencionó anteriormente, la implementación que sigue es solo para concretizar una forma de hacerlo, que no es la única manera de resolver este problema.

- XOR
- Circular left shift
- Circular right shift.

El XOR tiene la propiedad $A\ XOR\ B\ XORA = B$, que nos permitirá eliminar un caracter.
El circular right shift, además de hacer un shift right, cuando hay un underflow esos valores son introducidos en los bits más significativos. El circular left shift es equivalente.
Luego, usando estos operadores, se agregarán todos los caracteres del string en un hash en una posición correspondiente a la posición del string, ajustado por el tamaño del hash, es decir:

```
#define INTLEN 32
int hash(char* text, int first_char, int last_char)
{
  int out = 0;
  for (int i=0; i <= last_char - first_char; i++)
  {
    out ^= circular_shift_right((int) text[first_char + i], i % INTLEN);
  }
  return out;
}
```

Luego, para hacer el hash incremental se puede usar el hash del elemento anterior:

```
int next_hash(char* text, int first_char, int last_char, int prev_hash)
{
  int out = prev_hash;
  out ^= text[first_char-1]; // remueve el primer caracter
  out = circular_shift_left(out, 1);
  out ^= circular_shift_right((int) text[last_char],
                              (last_char - first_char) % INTLEN);
                                      //Agrega el último caracter
  return out;
}
```

Usando estas funciones de hash, se puede comparar el hash del string original con los hashes que aparecen al desplazarse por el texto. Ya que el hash incremental es O(1), este algoritmo sería O(n) porque se revisaría todos los caracteres del texto con operaciones O(1).

b)

Para incorporar esa funcionalidad, se puede cambiar el valor del caracter incorporado en el hash por otros, dependiendo si ese caracter incorporado cumpla con la consulta del REGEX. esto se puede limitar solamente en las posiciones del string que se requieren esa generalización. Para usar esta estrategia habría que revertir y agregar valores de x elementos, siendo x la cantidad de veces que se incluyen los elementos []. Esto daría una complejidad $O(n \cdot x)$.

Pregunta 2

1. Primero hay que representar la matriz binaria como un grafo. Los unos son como tierra y los 0's son como agua. Para representar el grafo tomamos los nodos como los 1's, y las aristas como los vecinos de cada 1. Luego, para conseguir la cantidad de componentes conexas distintas, o islas, bastaría con agregar un contador en un DFS que cuente cuántas veces se visita un nodo fuera del DFS search.

2. Se puede saber si un grafo dirigido es acíclico en la condición de búsqueda del DFS search. En esta condición, si el nodo ya se ha visitado antes en ese DFS search, en el algoritmo visto en clases esto sería color gris, entonces este grafo es cícilo.

3. Sabemos que hay que cumplir la condición del punto anterior para que el grafo sea acícico. Notamos la propiedad que en DFS search se va a revisar todas las aristas y todos los nodos de un grafo, y se colore un nodo solamente después de haber considerado todos las aristas de ese nodo. Por esto mismo, bastaría que cada vez que se encuentre una arista que esté conectado con un nodo gris, entonces se elimina esa arista.

4. Esta pregunta es más difícil que las anteriores. Primero se puede calcular la cantidad de requisitos de cada curso usando un algoritmo DFS. Luego, teniendo esta información en cada nodo podemos modificar la condición de entrada a un nodo. En vez de usar DFS visit en un nodo cuando es blanco, se puede modificar para que se entre al nodo cuando la cantidad de requisitos es 1, y después de eso bajar la cantidad de requisitos a 0. En el otro caso se puede bajar la cantidad de requisitos por 1. Ya que se calculó anteriormente la cantidad de requisitos, se va a poder entrar a todos los nodos en algún momento de esta manera. Además, se puede calcular en cada nodo en qué semestre se puede rendir ese curso, cuando se observe una arista entonces se recalcula el valor del semestre en el que se puede rendir al poner el máximo del valor actual y el número del semestre + 1 que hay que tomar el nodo padre de este. Esto intenta imitar el comportamiento de la realidad, donde uno solamente va a poder cursar el semestre cuando ya haya tomados todos los cursos previos, por lo que el semestre que se cursaría sería ese valor. Además, este valor va a ser correcto cuando la cantidad de requisitos que se necesitan cumplir es 0, ya que se actualizó este valor con todos sus padres y por eso se asegura la condición de entrada dicha anteriormente. Luego, la cantidad de semestres mínima para rendir todos los cursos será el máximo valor del semestre rendido calculado aquí. **Ojo:** En el video de la ayudantía se mostró una solución usando Dijkstra para este problema. Esta solución está incorrecta, lo siento por la confusión. La razón por la cual usar Dijkstra aquí es incorrecto se puede ver con un contraejemplo. En un grafo con solo un nodo que no es prerrequisito de otro, es posible que ese nodo tenga la misma distancia que un nodo intermedio, ya que pueden haber rutas alternativas a cada nodo. Luego, necesariamente el nodo final va a tener que ser cursado después de ese nodo intermedio, entonces la cantidad de semestres totales no sería esa distancia.