Semana 5

Pedro O. Pérez M., PhD.

Análisis y diseño de algoritmos avanzados
Tecnológico de Monterrey

pperezm@tec.mx

08-2021

Subcadena común más larga

Trie

- Trie

Grafos

- Definición
- Problema del camina más corto
- All-Pairs Shortest Path

Mochila 0/1

- Usando Programación Dinámica

Subcadena común más larga

El problema de la subcadena común más larga (Longest Common Substring) consiste en encontrar aquella subcadena continua que puede coincidir en todos las cadenas de entrada no necesariamente del mismo tamaño. Este problema puede arrojar múltiples soluciones. Por ejemplo, teniendo los strings: "AABCAB" y "CABCBABACC", el tamaño de la subcadena más larga es 3: "ABC", "CAB" y "ABA".

Ahora veamos como se plantea el problema: considere dos secuencias A = [a_1, a_2, ..., a_m] y B = [b_1, b_2, ..., b_n] . Para resolver el problema nos fijaremos en los últimos dos símbolos: a_m y b_n . Como podemos ver hay dos posibilidades:

- Caso 1: $a_m = b_n$. En este caso, la subcadena común más larga debe contener a_m . Primero debemos encontrar la subcadena común más larga de [a_1, a_2, ..., a_{m-1}] y [b_1, b_2, ..., b_{n-1}] + 1, y luego comparar ese resultado con el máximo previo.
- Caso 2: $a_m \neq b_n$. En este caso, la longitud más larga será cero.

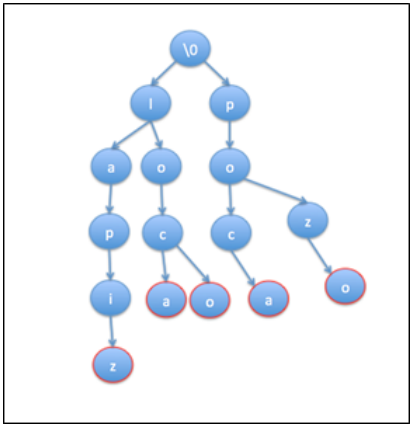
Procedure 1 LCS

Input: $A, B : Array$
  $M : Matrix[0..A.length][0..B.length]$
  $m \leftarrow A.length$
  $n \leftarrow B.length$
  for $i \leftarrow 1$ to m **do**
    if $A[i] = B[0]$ **then**
      $M[i][0] \leftarrow 1$, $maximum \leftarrow 1$
    end if
  end for
  for $i \leftarrow 1$ to n **do**
    if $A[0] = B[i]$ **then**
      $M[i][0] \leftarrow 1$, $maximum \leftarrow 1$
    end if
  end for

```
for i ← 1 to m do
    for j ← 1 to n do
        if A[i] = B[j] then
            M[i][j] ← 1 + M[i − 1][j − 1]
            maximum ← MAX(M[i][j], max)
        else
            M[i][j] ← 0
        end if
    end for
end for
return maximum
```

Trie

El trie, también conocido como árbol de prefijos, es una estructura de datos muy útil para resolver problemas relacionados con cadenas. El nombre proviene de la palabra **RETRIEVAL**, que significa recuperación. Internamente, funciona como un árbol ordenado, donde cada nodo representa un carácter de una palabra insertada en el trie, y los nodos hijos de éste son caracteres que aparecen después de él en la palabra.



Al construir un trie, se requiere almacenar en cada nodo lo siguiente:

- Letra que representa.
- Si es o no la última letra de una palabra.
- Nodo padre.
- Nodos hijos.

Revisar el código en Github.

Definición

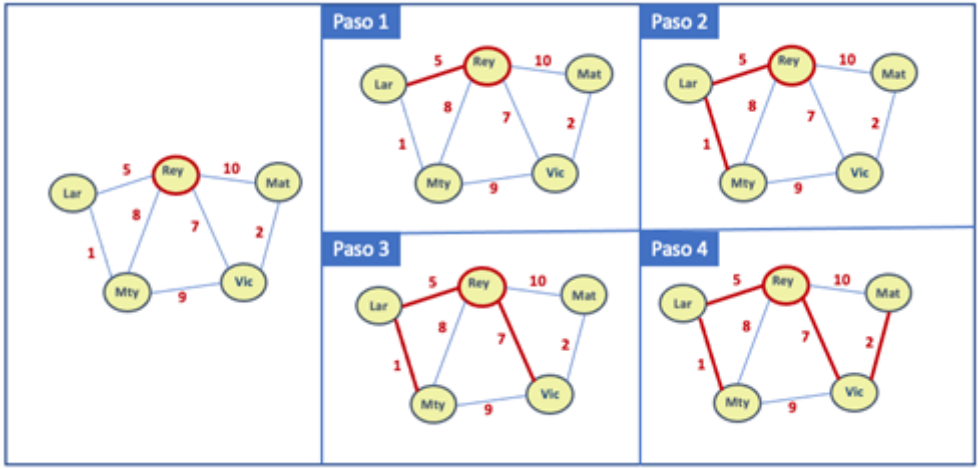
Un grafo no direccionado $G = (V, E)$ consiste de una colección de vértices, V y una colección de arcos, E. Representamos cada arco, $e \in E$, como un subcojunto de dos elementos $e = \{u, v\}$ siendo $u, v \in V$, llamando a u y v los puntos terminales de e.

Un grafo direccionado $G = (V, E)$ consiste de una colección de vértices, V y una colección de arcos, E. Representamos cada arco, $e \in E$, como un par ordenado de dos elementos $e = (u, v)$ siendo $u, v \in V$. Llamamos a u el punto inicial y a v el punto final de e.

- Un grafo no direccionado está conectado si, para cada par de nodos u y v, existe un camino de u a v.
- Un grafo direccionado está fuertemente conectado si, para cada par de vértices u y v, existe un camino de u a v y de v a u.
- En un grafo no direccionado $G = (V, E)$ una secuencia de nodos P = [v_1, v_2, ..., v_{k-1}, v_k] con la propiedad de que cada par consecutivo v_i, v_{i+1} está conectado por un arco en G, es llamado un camino (path) de v_1 a v_k.
- Un ciclo es un camino P = [v_1, v_2, ..., v_{k-1}, v_k] en el cual, para cualquier $k > 2$, los primeros k˘1 vértices son distintos y $v_1 = v_k$.

Algoritmos de Dijkstra

En muchas ocasiones requerimos saber la ruta mas corta de un punto a El algoritmo de Dijkstra también conocido como el algoritmo de caminos mínimos, es un algoritmo desarrollado por el científico en computación Edsger Dijkstra para determinar el camino más corto para un grafo ponderado dado un vértices inicial al resto de los vértices, utilizando la técnica de Algoritmos Avaros.


Lar
Rey
Mat
Mty
Vic
5
10
8
7
2
1
9
Paso 1
Paso 2
Paso 3
Paso 4

Procedure 2 DIJKSTRA

Input: $u : Vertex, G : Graph$

```
Set reached
PiorityQueue pending
pending.enqueue(PAIR(u, 0))
while pending is not empty do
    e ← pending.dequeue()
    if e.first is not in reached then
        Add to Reached
        for each (e, v) incident to e do
            Q.enqueue(PAIR(v, COST(e, v) + e.second))
        end for
    end if
end while
return reached
```

All-Pairs Shortest Path

El algoritmo Floyd – Warshall permite encontrar las rutas más cortas en un grfo ponderado con costos positivos o negativos (pero sin ciclos negativos). Una sola ejecución del algoritmo encontrará los costos de las rutas más cortas entre **todos los pares de vértices**. Aunque no devuelve detalles de las rutas en sí, es posible reconstruir las rutas con simples modificaciones al algoritmo. Las versiones del algoritmo también se pueden usar para encontrar el cierre transitivo de una relación R.

1 = Rey
2 = Lar
3 = Mty
4 = Vic
5 = Mat

K = 0

	1	2	3	4	5
1	0	5	8	7	10
2	5	0	1	INF	INF
3	8	1	0	9	INF
4	7	INF	9	0	2
5	10	INF	INF	2	0

K = 1

	1	2	3	4	5
1	0	5	8	7	10
2	5	0	1	12	15
3	8	1	0	9	18
4	7	12	9	0	2
5	10	15	18	2	0

K = 2

	1	2	3	4	5
1	0	5	6	7	10
2	5	0	1	12	15
3	6	1	0	9	16
4	7	12	9	0	2
5	10	15	16	2	0

K = 3

	1	2	3	4	5
1	0	5	6	7	10
2	5	0	1	10	15
3	6	1	0	9	16
4	7	10	9	0	2
5	10	15	16	2	0

K = 4

	1	2	3	4	5
1	0	5	6	7	9
2	5	0	1	10	12
3	6	1	0	9	11
4	7	10	9	0	2
5	9	12	11	2	0

K = 5

	1	2	3	4	5
1	0	5	6	7	9
2	5	0	1	10	12
3	6	1	0	9	11
4	7	10	9	0	2
5	9	12	11	2	0

Procedure 3 VERSION1

Input: $M : Adjacent\_Matrix$

```
for k ← 1 to M.length do
  for i ← 1 to M.length do
    for j ← 1 to M.length do
      M[i][j] ← M[i][k] and M[k][j]
    end for
  end for
end for
```

Procedure 4 VERSION2

Input: $M : Adjacent\_Matrix$

```
for k ← 1 to M.length do
  for i ← 1 to M.length do
    for j ← 1 to M.length do
      M[i][j] ← MIN(M[i][j], M[i][k] + M[k][j])
    end for
  end for
end for
```

Mochila 0/1 - Versión DP

Sean N objetos no fraccionables de pesos w_i, y beneficios v_i, y sea C el peso máximo que puede llevar la mochila. El problema consiste en llenar la mochila con objetos, tal que se maximice el beneficio. Por ejemplo, W = [10, 4, 6, 12], V = [100, 70, 50, 10], y C = 12. ¿Cuál es la beneficio máximo que podemos obtener?

Procedure 5 KNAPSACK

Input: $W, V : Array, C : Integer$
  $M : Matrix[0...S.length][0..C]$
  $n \leftarrow W.length$
  $INIT(M, 0)$
  for $i \leftarrow 1$ to n **do**
    for $j \leftarrow 1$ to C **do**
      /* There is not space in the container */
      if $j < W[i]$ **then**
        $M[i][j] \leftarrow M[i-1][j]$
      else
        $M[i][j] \leftarrow MAX(V[i] + M[i-1][j - W[i]], M[i-1][j])$
      end if
    end for
  end for
  return $M[n][C]$

SuperSale [1]

Hay una Superventa en un SuperHiperMarket. Cada persona puede llevar solo un objeto de cada tipo, es decir, un televisor, una zanahoria, pero a un precio muy bajo. Vamos con toda una familia a ese SuperHiperMarket.
Cada persona puede tomar tantos objetos como pueda llevar a cabo desde la Superventa. Hemos dado una lista de objetos con precios y su peso. También sabemos cuál es el peso máximo que toda persona puede soportar. ¿Cuál es el valor máximo de los objetos que podemos comprar en SuperSale?

[1] https://onlinejudge.org/external/101/10130.pdf

Entrada

La entrada consta de casos de prueba T. El número de ellos (1 <= T <= 1000) se da en la primera línea del archivo de entrada. Cada caso de prueba comienza con una línea que contiene un solo número entero N que indica el número de objetos (1 <= N <= 1000). Luego sigue N líneas, cada una con dos enteros: P y W. El primer entero (1 <= P <= 100) corresponde al precio del objeto. El segundo entero (1 <= W <= 30) corresponde al peso del objeto. La siguiente línea contiene un número entero (1 <= G <= 100) es el número de personas en nuestro grupo. Las siguientes líneas G contienen el peso máximo (1 <= MW <= 30) que puede soportar esta i-ésima persona de nuestra familia (1 <= i <= G).

Salida

Para cada caso de prueba, su programa tiene que determinar un número entero. Imprima el valor máximo de los bienes que podemos comprar con esa familia.

Ejemplo de entrada

```
6
64 26
85 22
52 4
99 18
39 13
54 9
4
23
20
20
26
```

Ejemplo de salida

```
514
```