Análisis y Diseño de Algoritmos - Parcial No. 2

Sección 20 - Gabriel Brolo

Abril 2024

1 Instrucciones

- Debe realizar este examen de forma **individual**.
- Debe entregar la respuesta a las preguntas planteadas acompañada de un análisis riguroso y explicación pertinente, i.e., no se aceptarán respuestas sin procedimiento ni explicación y la nota será automáticamente de **cero** puntos.
- Entregue un PDF con la resolución para su parcial y además grabe un video de no más de 5 minutos de longitud en donde muestre la ejecución de los programas planteados en cada problema que usted realice. Súbalo a YouTube como no listado e inclúyalo en su PDF de respuestas.
- Duración: este examen se encontrará habilitado para su realización desde el **08 de abril de 2024** hasta el **14 de abril de 2024**, por lo que podrá realizarlo a su conveniencia en los tiempos que mejor se ajusten a su intinerario. Solo podrá subir sus respuestas una vez, así que esté seguro de las mismas.
- Este parcial contiene múltiples problemas, con diferente valoración de puntos. Elija aquella combinación de problemas que lo lleve a un punteo completo. Si elije problemas extra, se tomarán como puntos extra que puede utilizar en otras entregas. Se hará una regla de tres con base en la puntuación total del parcial en puntos netos para definir cuántos puntos netos extra tendrá.
- Buena suerte y recuerde que Diosito lo/la está viendo.

2 Problemas

A continuación encontrará un conjunto de problemas a resolver, relacionados con los siguientes temas vistos en clase: “Programación Dinámica” y “Algoritmos Greedy”. Resuelva los que crea conveniente para obtener un punteo completo:

2.1 Problema 1 [40 puntos]

Dado un número positivo n y un teclado móvil de un Nokia 3230, que tiene dígitos del 0 al 9 asociados con cada tecla, se desea contar el total de combinaciones posibles de dígitos de longitud n. Podemos comenzar con cualquier dígito y presionar solo cuatro teclas adyacentes a cualquier dígito. El teclado también contiene las teclas $*$ y #, las cuales no están permitidas presionar. Por ejemplo para:

```
# Cantidad de digitos n
n = 2

# Mapping del teclado del Nokia
keypad = [
    ['1', '2', '3'],
    ['4', '5', '6'],
    ['7', '8', '9'],
    ['*', '0', '#']
]

Input:  n = 2
Output: 36
Explicacion: El total de combinaciones posibles son 36
    [00, 08, 11, 12, 14, 21, 22, 23, 25, 32, 33, 36, 41, 44, 45, 47,
    ... ,96, 98, 99]
```

- Explique porqué este problema exhibe subestructura óptima. **(5 puntos)**.
- Explique una idea/solución que exhiba subproblemas traslapados e indique cómo los mismos subproblemas se computan repetidamente. **(5 puntos)**.
- Sabemos que los problemas con subestructura óptima y subproblemas traslapados pueden resolverse utilizando programación dinámica, donde las soluciones de subproblemas se memoizan en lugar de calcularse repetidamente. Escriba un código en Python con enfoque de memoización top-down que resuelva este problema. Coloque un enlace a un **GitHub Gist privado** con la solución. Recuerde que debe crear también un video en dónde muestre la ejecución de su código. **(15 puntos)**.
- Encuentre el tiempo de complejidad para este algoritmo. Recuerde, deje su procedimiento. **(10 puntos)**.
- Usando su programa, encuentre las combinaciones totales posibles para $n = 10$. **(5 puntos)**.

2.2 Problema 2 [40 puntos]

Dada una matriz cuadrada de 0's y 1's, calcule el tamaño de la cruz (símbolo de suma +) más grande formada por 1's.

Input:

```
grid = [
         [1, 0, 1, 1, 1, 1, 0, 1, 1, 1],
         [1, 0, 1, 0, 1, 1, 1, 0, 1, 1],
         [1, 1, 1, 0, 1, 1, 0, 1, 0, 1],
         [0, 0, 0, 0, 1, 0, 0, 1, 0, 0],
         [1, 1, 1, 0, 1, 1, 1, 1, 1, 1],
         [1, 1, 1, 1, 1, 1, 1, 1, 1, 0],
         [1, 0, 0, 0, 1, 0, 0, 1, 0, 1],
         [1, 0, 1, 1, 1, 1, 0, 0, 1, 1],
         [1, 1, 0, 0, 1, 0, 1, 0, 0, 1],
         [1, 0, 1, 1, 1, 1, 0, 1, 0, 0]
]
```

Output: 17

Explicacion: La cruz (simbolo de suma +) mas grande de 1s se encuentra abajo, teniendo un tamano de 17.

```
1  0  1  1  1  1  0  1  1  1
1  0  1  0  |  1  1  0  1  1
1  1  1  0  |  1  0  1  0  1
0  0  0  0  |  0  0  1  0  0
1  1  1  0  |  1  1  1  1  1
-  -  -  -  |  -  -  -  -  0
1  0  0  0  |  0  0  1  0  1
1  0  1  1  |  1  0  0  1  1
1  1  0  0  |  0  1  0  0  1
1  0  1  1  |  1  0  1  0  0
```

Input:

```
grid = [
         [1, 1, 1, 1, 1, 1],
         [1, 0, 1, 1, 0, 1],
         [0, 1, 1, 0, 0, 1],
         [1, 1, 1, 1, 1, 1],
         [1, 0, 0, 1, 0, 1],
         [1, 0, 1, 1, 0, 0]
]
```

```
Output: 0

Explicacion: no se puede construir una cruz (+) mas grande de 1s.
```

- Escriba un código en Python que utilice un algoritmo de programación dinámica para resolver este problema. Comience por crear cuatro matrices auxiliares `left[][]`, `right[][]`, `top[][]`, `bottom[][]`, donde `left[j][j]`, `right[i][j]`, `top[i][j]` y `bottom[i][j]` que almacenan el número máximo de 1's consecutivos presentes a la izquierda, derecha, arriba y abajo de la celda `(i, j)`, incluida la celda `(i, j)`, respectivamente, utilizando programación dinámica.

```
if grid[i][j] == 1
  left[i][j] = left[i][j - 1] + 1

if grid[i][j] == 1
  top[i][j] = top[i - 1][j] + 1

if grid[i][j] == 1
  bottom[i][j] = bottom[i + 1][j] + 1

if grid[i][j] == 1
  right[i][j] = right[i][j + 1] + 1
```

  Después de calcular las matrices anteriores, encuentre la celda `(i, j)` que tiene el valor máximo en cada dirección (considerando el mínimo entre `left[i][j]`, `right[i][j]`, `top[i][j]` y `bottom[i][j]`).

  Coloque un enlace a un **GitHub Gist privado** con la solución. Recuerde que debe crear también un video en dónde muestre la ejecución de su código. Use como prueba el input proporcionado. **(30 puntos)**.

- Encuentre el tiempo de complejidad para este algoritmo. Recuerde, deje su procedimiento. **(10 puntos)**.

2.3 Problema 3 [5 puntos]

Verdadero o Falso: Una solución óptima para un *knapsack problem* siempre contendrá el objeto i con la mayor relación valor-costo v_i/c_i. Explique.

2.4 Problema 4 [5 puntos]

Supongamos que se nos da un grafo ponderado $G = (V, E, w)$ de autopistas, y el nuevo gobierno ha implementado una nueva regla de impuestos (para tratar de conseguir algo de dinero) en la que el costo de un camino se duplica como penalización si el número de aristas en el camino es mayor que 10. Explique cómo reducir el encontrar el peso del camino más corto entre cada par de vértices (bajo esta penalización) al problema usual de todos los caminos más cortos (*shortest-path*) entre pares (como se resuelve con Floyd-Warshall).

2.5 Problema 5 [15 puntos]

Dada una matriz T de $m \times n$ sobre un campo (como los reales), demuestre que (S, I) es una matroide, en dónde S es el conjunto de las columnas de T y $A \in I$ si y sólo si las columnas en A son linealmente independientes.

2.6 Problema 6 [5 puntos]

Resuelva la siguiente instancia del *scheduling problem*, i.e., provea el *schedule* final que sea óptimo y provea la penalización total:

a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	10	20	30	40	50	60	70

Table 1: Tabla de deadlines y pesos