Autores e identificación

William Méndez – 202012662

Juliana Galeano – 202012128

Daniel Aguilera – 202010592

Boris N. Reyes R. – 202014743

PROYECTO, PARTE 1

Introducción

El propósito de este informe es realizar la documentación del proyecto de curso en su parte primera y brindar respuesta a los escenarios planteados en su enunciado. Se presentan tres componentes respectivos al algoritmo solución, análisis de complejidad tanto temporal como espacial, y la respuesta ante los escenarios de comprensión de problemas algorítmicos.

1. Algoritmo de solución

El algoritmo diseñado está basado en el algoritmo de Dijkstra para encontrar el menor costo para avanzar de un nodo a otro en un grafo con caminos bidireccionales y peso. En nuestra implementación tomamos los datos en una estructura distinta a un verdadero grafo, como sabemos que los vértices son todas las parejas x,y donde $x \in [0, n)$ y $y \in [0, m)$ no los debemos almacenar y lo que sí almacenamos son las coordenadas de los portales en un diccionario de tamaño $x \leq p$ donde (x_{start}, y_{start}) es la llave y (x_{end}, y_{end}) es el valor asignado sin embargo en la carga revisamos también cuantos pisos no tienen ninguna salida de portal y se eliminan los portales de ese piso para salvar tiempo de ejecución. Luego de leer los datos y almacenarlos de esta manera se usan n, m, una lista con los valores de P y el diccionario de portales para ejecutar el método de búsqueda de la siguiente manera:

Se crea un diccionario con llave (x,y) y valor z donde la llave significa el mínimo costo de energía para llegar a esa habitación y se asigna a la habitación inicial (0,0) un coste de 0.

Se crea un diccionario para almacenar en sus llaves las coordenadas de los nodos que han sido visitados y uno para almacenar los que no han sido visitados.

Se crea una cola de prioridad para almacenar los nodos que no han sido visitados, se agrega el nodo inicial y mientras la cola de prioridad tenga elementos se realiza lo siguiente:

Se quita el elemento “actual” de mayor prioridad de la cola y del diccionario de pendientes y se agrega a los nodos visitados. Se revisa si este elemento tiene

vecinos a la izquierda, derecha y arriba por medio de un portal y se agregan a una lista de vecinos de tamaño < 4 de la cual por cada elemento revisamos: Si el vecino está en un piso superior el nuevo costo para llegar al vecino es el costo para llegar a el nodo actual, si está en el mismo piso el nuevo costo va a ser el costo para llegar al nodo actual más el costo para moverse en ese piso. Luego, si el vecino no se encuentra en la memoria se le asigna el nuevo costo en la memoria, si ya existía en memoria se revisa este gasto anterior y si el nuevo gasto es menor al gasto anterior se reemplaza en la memoria.

Después, si el vecino no está pendiente de ser revisado ni ha sido revisado se agrega a la cola para ser visitado y se revisa el siguiente elemento en la cola de espera.

Cuando la cola de prioridad está vacía significa que ya recorrimos todos los nodos a los que es posible llegar desde (0,0) y se revisa si el nodo destino (n-1,m-1) está en la memoria, es decir, es un nodo al que se puede llegar desde el inicio, si no lo está se retorna "NO EXISTE" y si sí existe se retorna su valor en memoria, que es el mínimo para llegar a este punto.

Finalmente, este valor se imprime por salida estándar y se continua al siguiente caso hasta acabar.

2. Análisis de complejidades espacial y temporal

Tiempos:

- Asignación (c_1)
- Comparaciones (c_2)
- +,*,-,/ (c_3)

Método lectura()

```
14  def lectura():
15
16      # file = open("1.in", "r")
17      # file = open("2.in", "r")
18      # file = open("Proyecto/P1_casesFP.in", "r")
19
20      # nCasos = int(file.readline())
21      nCasos = int(stdin.readline())
22
23      while nCasos != 0:
24          # datos = file.readline().split(" ")
25          datos = stdin.readline().split(" ")
26          nPisos, nHabitaciones, nPortales = int(datos[0]), int(datos[1]), int(datos[2])
27          # gastoEnergia = [int(x) for x in file.readline().split(" ")]
28          gastoEnergia = [int(x) for x in stdin.readline().split(" ")]
29          portales, pisosNoVisitados = {}, {x: 0 for x in range(1,nPisos)}
```

21. C1 + O(1)

23. (O(nCasos) + C2) * (**Interno**)

Interno

25. C1 + O(n)

26. C1

28. C1 + O(n)

29. C1 + O(nPisos - 1)

```
31      for i in range(nPortales):
32          # datos = file.readline().split(" ")
33          datos = stdin.readline().split(" ")
34          portales[(int(datos[0])-1, int(datos[1])-1)] = (int(datos[2])-1, int(datos[3])-1)
35          # if inDict(pisosNoVisitados, int(datos[2])-1):
36          if pisosNoVisitados.get(int(datos[2])-1, -1) != -1:
37              pisosNoVisitados.pop(int(datos[2])-1)
38
39      entradas = list(portales.keys())
40
```

31. O(nPortales)(**Adentro**)

Adentro

33. C1 + O(n)

34. 4*C3 + C1

36. O(1) + C3 + C2

37. O(1) + C3

Fin de Adentro = 2C1 + C2 + 6C3 + O(n) + 2O(1)

39. C1 + O(n)

```
41      # start = timer() # TODO: Quitar esto
42      # if inDict(pisosNoVisitados, nPisos - 1):
43      if pisosNoVisitados.get(nPisos - 1, -1) != -1:
44          print("NO EXISTE")
45      else:
46          for i in entradas:
47              # if inDict(pisosNoVisitados, i[0]):
48              if pisosNoVisitados.get(i[0], -1) != -1:
49                  portales.pop(i)
50          print(calcularMinEnergiaDijkstra(nPisos, nHabitaciones, gastoEnergia, portales))
51
52      # elapsed_time = timer() - start # TODO: Quitar esto
53      # print("Caso:", 1001 - nCasos,"tamaño de la torre",nPisos,"x",nHabitaciones, "Tiempo:
54      nCasos -= 1
55
```

43. O(1) + C3 + C2

46. O((n-1)m)(**In**)

In

48. O(1) + C2

49. O(1)

Fin de In

54. C1 + C3

Fin de Interno

Complejidad Temporal del método lectura():

C1 + O(1) + O(nCasos)(4C1 + 2O(n) + O(nPisos-1) + O(p)C1 + O(p)C2 + O(p)6C3 + O(p)2O(1) + O(p)O(n) + C1 + O(n) + O(1) + C3 + C2 + O((n-1)m)2O(1) + O((n-1)m)C2 + C1 + C3) + C2(4C1 + 2O(n) + O(nPisos-1) + O(p)C1 + O(p)C2 + O(p)6C3 + O(p)2O(1) + O(p)O(n) + C1 + O(n) + O(1) + C3 + C2 + O((n-1)m)2O(1) + O((n-1)m)C2 + C1 + C3)

Método calcularMinEnergiaDijkstra()

```
56  def calcularMinEnergiaDijkstra(nPisos, nHabitaciones, gastoEnergia, portales):
57      inicio = (0,0)
58      memoria, visitados, porVisitar = {inicio: 0}, {}, {inicio: 0}
59
60      pq, entry = [], (0, inicio)
61      hq.heappush(pq,entry)
62
63      while len(pq)>0:
64          (dist, actual) = hq.heappop(pq)
65          porVisitar.pop(actual)
66
67          # print(actual, dist)
68          visitados[actual] = 0
```

57. C1

58. C1

60. C1

61. O(1)

63. O(nm) (**Adentro**)

Adentro

64. C1 + O(log n)

65. O(1)

68. C1

```
70          vecinos = []
71          # if actual[1] > 0 and not inDict(visitados, (actual[0], actual[1]-1)):
72          if actual[1] > 0 and visitados.get((actual[0], actual[1]-1), -1) == -1:
73              vecinos.append((actual[0], actual[1] - 1))
74
75          # if actual[1] < nHabitaciones and not inDict(visitados, (actual[0], actual[1]+1)):
76          if actual[1] < nHabitaciones and visitados.get((actual[0], actual[1]+1), -1) == -1:
77              vecinos.append((actual[0], actual[1] + 1))
78
79          # if inDict(portales, actual) and not inDict(visitados, portales[actual]):
80          if portales.get(actual, -1) != -1 and visitados.get(portales[actual], -1) == -1:
81              vecinos.append(portales[actual])
```

70. C1

72. 2 * C2 + O(1) + C3

73. O(1) + C3

76. 2*C2 + 2*C3 + O(1)

77. O(1) + C3

80. 2 * (O(1) + C2)

81. O(1)

```
83          for vecino in vecinos:
84              if vecino[0] > actual[0]:
85                  nuevoGasto = memoria[actual]
86              else:
87                  nuevoGasto = gastoEnergia[vecino[0]] + memoria[actual]
88
89              # if not inDict(memoria, vecino):
90              if memoria.get(vecino, -1) == -1:
91                  memoria[vecino] = nuevoGasto
92
93              viejoGasto = memoria[vecino]
94              if nuevoGasto < viejoGasto:
95                  memoria[vecino] = nuevoGasto
96              # if not inDict(visitados, vecino) and not inDict(porVisitar, vecino):
97              if visitados.get(vecino, -1) == -1 and porVisitar.get(vecino, -1) == -1:
98                  hq.heappush(pq,(nuevoGasto, vecino))
99                  porVisitar[vecino] = 0
```

83. O(3)(**Interno**)

Interno

84. C2

85. C1

87. C1 + C3

90. O(1) + C2

91. C1

93. C1

94. C2

95. C1

97. 2 * (O(1) + C2)

98. log(n) #

99. C1

Fin de Interno = C2 + C1 + C1 + C3 + O(1) + C2 + C1 + C1 + C2 + C1 + 2 * (O(1) + C2) + log(n) + C1

Fin de Adentro = C1 + O(log n) + O(1) + C1 + C1 + 2 * C2 + O(1) + C3 + O(1) + C3 + 2*C2 + 2*C3 + O(1) + O(1) + C3 + 2 * (O(1) + C2) + O(1) + O(3)(Interno)

```
101     goal = (nPisos - 1, nHabitaciones - 1)
102
103     if visitados.get(goal, -1) == -1:
104         return "NO EXISTE"
105     else:
106         return memoria[(nPisos - 1,nHabitaciones - 1)]
107
108 lectura()
```

101. C1 + 2 * C2

103. O(1) + C2

106. 2 * C3

Complejidad Temporal del método calcularMinEnergiaDijkstra():

C1 + O(1) + C1(O(nm)) + O(log n)(O(nm)) + 8*O(1)(O(nm)) + 6*C2(O(nm)) + 5*C3(O(nm)) + (O(3))C1(O(nm))+ (O(3))(O(log n))(O(nm)) + 8(O(3))(O(1))(O(nm)) + 6(O(3))C2(O(nm)) +3(O(3))C3(O(nm)) + 6((O(3))^2)C1(O(nm)) + 5((O(3))^2)C2(O(nm)) + ((O(3))^2)C3(O(nm)) + 3((O(3))^2)(O(1))(O(nm)) + ((O(3))^2)(log(n))(O(nm)) + C1 + 2 * C2 + O(1) + C2

Complejidad Temporal Ponderada:

Tras llevar a cabo la reducción de términ

Tras llevar a cabo la simplificación de términos, y al adicionar el llamado al método **calcularMinEnergiaDijkstra()** en lectura() se obtiene una complejidad temporal es O(nm(log n))

C1 + 1 + C1*O(nm) + O((nm)log n)) + 8*O(nm) + 6*C2*O(nm) + 5*C3*O(nm) + 3*C1*O(nm)+ 3*O(nm*log n) + 24*O(nm) + 18*C2*O(nm) + 9*C3O(nm) + 6((O(3))^2)C1(O(nm)) + 45*C2*O(nm) + 9*C3*O(nm) + 27*O(nm) + 9*(O(nm*log(n))) + C1 + 2 * C2 + 1 + C2

Complejidad espacial:

A lo largo de la solución se usaron las siguientes estructuras: Diccionarios, HeapQs, Listas.

Teniendo esto en cuenta es posible calcular la complejidad espacial, ya que sabemos la complejidad espacial de cada estructura. En el orden dado tenemos, $O(n)$, $O(n)$, $O(n)$. En conclusión, es posible afirmar que la complejidad espacial es $O(n)$ debido a las razones dadas.

3. Respuestas a los escenarios de comprensión de problemas algorítmicos.

(i) que nuevos retos presupone este nuevo escenario -si aplica-?

(ii) que cambios -si aplica- le tendría que realizar a su solución para que se adapte a este nuevo escenario?

3.1. Suponga ahora que los portales son bidireccionales.

Al hacer uso del algoritmo de Dijkstra podemos concluir que no supone un gran reto que los portales sean bidireccionales, sin embargo, el programa sí debería ser modificado, pues para asignarle el peso a un movimiento de portal se tiene en cuenta que la salida del portal sea un piso superior.

3.2. Se le pide ahora calcular el número de rutas que existen.

Para realizar eso haría falta rediseñar el algoritmo, pues no haría falta tener en cuenta el mayor aspecto de la estructura actual, la energía de moverse de un cuarto a otro y significaría que adaptar la solución sería reescribirla basándose en un método dinámico de conteo de caminos.

3.3. Se le pide ahora calcular la ruta optima en la cual el estudiante puede visitar todos los cuartos y finaliza en (n,m)

Para realizar este escenario hay que tener en cuenta que no en todos los casos es posible, ya que pueden existir pisos a los cuales no se puede acceder mediante un

portal, lo cual causaría que no fuera posible revisar todos los cuartos. Por otro lado, un escenario más plausible seria revisar todos los cuartos que fueran posible, de tal forma que podría visitar todos los cuartos a los cuales se puede llegar con el uso de un portal. Adicionalmente, el reto de este problema es que en muchas ocasiones tendría que moverse sobre cuartos ya visitados lo cual aumentaría el costo de llegada. Para esto haría falta rediseñar nuestra solución de tal forma que revise todos los cuartos posibles e intente llegar a la posición del escenario original.