Teoría de Algoritmos – TP2

“Viaje en Tren (programación dinámica)”

Materia: 75.29 Teoría de Algoritmos

Alumno: Juan Sebastián Goldberg

Padrón: 82078

Mail: sebas.goldber@gmail.com

Fecha: 10-NOV-2014

Contenido

Contenido 2

Análisis y Diseño 3

Planteo inicial [O(n\*\*2)] 3

Resolución [O(n\*p\*log(p) + p\*\*2)] 3

Búsqueda del óptimo [O(p)] 4

Ejemplo de Resolución 4

Planteo del problema 4

Resolución 4

Condición inicial 5

Análisis de cómo llegar a la ciudad X en k=1 tramo 5

Ciudad 1 5

Ciudad 2 5

Ciudad 3 6

Resultados obtenidos para k=1 6

Análisis de cómo llegar a la ciudad X en k=2 tramos 6

Ciudad 1 6

Ciudad 2 6

Ciudad 3 7

Resultados obtenidos para k=2 7

Obtención del óptimo 7

Reconstrucción de la solución óptima 7

Diagrama de clases 8

Justificación de órdenes del lenguaje y librerías 8

Código Fuente 8

tp2.py 9

escenario.py 10

optimo.py 15

tramo.py 16

lista\_ordenada.py 16

Análisis y Diseño

Planteo inicial [O(n\*\*2)]

n: Cantidad de ciudades.

Para resolver el problema planteado lo que se hizo fue utilizar una resolución similar al problema de la mochila.

Para comenzar se construye una matriz donde cada componente de la misma es un listado ordenado de soluciones posibles. Cada fila representa la cantidad de tramos necesarios para llegar a una determinada ciudad. Cada columna representa una ciudad.

Por lo tanto, la componente en la fila i, columna j, representa todas las soluciones que demoran menos para llegar a la ciudad j utilizando i tramos desde cualquier otra ciudad. Con “demoran menos”, nos referimos a todas las soluciones que llegan a j en el menor tiempo posible por cada tren que llega a j.

Luego, inicialmente nuestra matriz M la cargamos de la siguiente forma:

* M[0][ciudad\_origen] = [ Optimo( horario\_llegada=hora\_inicial, tramo=Tramo( tren=None, hora\_desde=hora\_inicial, hora\_hasta=hora\_inicial, ciudad\_desde=ciudad\_origen, ciudad\_hasta=ciudad\_origen ), tiempo\_total=0), ]
* M[0][x] = []; donde x<>ciudad\_origen y x pertenece al conjunto de ciudades.

Como vemos, utizando 0 tramos, solo podemos llegar a la ciudad origen a la hora hora\_inicial (dato) sin utilizar ningún tren. Vemos que en el listado de cada componente guardamos instancias de Optimo ordenando por el horario\_llegada a ciudad\_hasta. Optimo, en realidad no representa los Optimos del problema sino las soluciones que “demoran menos” mencionadas con anterioridad. Un Optimo se compone del horario de llegada a la ciudad analizada, el tramo utilizado para llegar y el tiempo total insumido en los k pasos.

Resolución [O(n\*p\*log(p) + p\*\*2)]

n: Cantidad de ciudades; p: Cantidad de tramos

Luego resolver el problema simplemente consiste en resolver cada componente de la matriz, insertando las soluciones que “demoran menos” en el listado de cada componente.

Por ejemplo, sea M nuestra matriz, si estamos analizando la cantidad de tramos k, para llegar a la ciudad x:

* Por cada tramo t que llega a x:
  + Intentamos obtener de M[k-1][t.ciudad\_desde] (el listado de soluciones que llegan a t.ciudad\_desde utilizando k-1 tramos), la solución O que tenga horario de llegada a t.ciudad\_desde más próximo a t.horario\_desde.
  + Si no existe tal solución, continuamos analizando el siguiente tramo t
  + Si existe significa que encontramos una solución para llegar a la ciudad x utilizando k tramos, por lo tanto insertamos dicha solución al listado M[k][x]:
    - tiempo\_total = O.tiempo\_total + t.horario\_llegada – O.tramo.horario\_llegada
    - Y la solución a insertar sería: Optimo( horario\_llegada=t.horario\_llegada, tramo=t, tiempo\_total=tiempo\_total)

Búsqueda del óptimo [O(p)]

p: Cantidad de tramos

Una vez resuelta cada componente de M simplemente debemos buscar la solución que tiene horario de llegada más temprano y en caso de empate la que tenga menor duración.

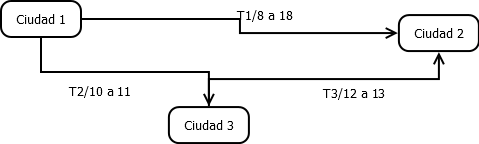
Esto lo hacemos recorriendo los listados M[k][ciudad\_destino] para cada k desde 0 a (n-1). Donde n es la cantidad de ciudades y ciudad\_destino es la ciudad a donde se desea ir.

Ejemplo de Resolución

Planteo del problema

Supongamos que tenemos el siguiente esquema:

* Las ciudades: 1, 2 y 3.
* Los tramos:
  + El tramo de la ciudad 1 a la 2 con salida 8:00 hs y llegada 18:00 hs.
  + El tramo de la ciudad 1 a la 3 con salida 10:00 hs y llegada 11:00 hs.
  + El tramo de la ciudad 3 a la 2 con salida 12:00 hs y llegada 13:00 hs.
* Hora inicial: 8:00 hs.
* Ciudad origen: 1
* Ciudad destino: 2



Resolución

Vamos a proceder como indicamos en las secciones anteriores.

Inicialmente tenemos una matriz M como la que sigue:

|  |  |  |  |
| --- | --- | --- | --- |
| **# Tramos\Ciudad** | **1** | **2** | **3** |
| **0** | [] | [] | [] |
| **1** | [] | [] | [] |
| **2** | [] | [] | [] |

Las columnas representan las ciudades.

Las filas representan la cantidad de tramos utilizados para llegar a una determinada ciudad.

Cada componente de la matriz es un listado de ítems de solución, que para simplificar, cada ítem lo representaremos como una tupla con la siguiente información:

(<hora\_de\_llegada\_a\_la\_ciudad\_x>, <tiempo\_acumulado>, <puntero\_al\_item\_anterior>)

Condición inicial

Como comentamos la condición inicial estaría representado por la salida de la ciudad origen a la ciudad origen con una hora de llegada con valor de la hora inicial y tiempo acumulado nulo: (8:00, 0, NULL)

Luego la matriz quedaría como sigue:

|  |  |  |  |
| --- | --- | --- | --- |
| **# Tramos\Ciudad** | **1** | **2** | **3** |
| **0** | [I1(8:00, 0, NULL)] | [] | [] |
| **1** | [] | [] | [] |
| **2** | [] | [] | [] |

Análisis de cómo llegar a la ciudad X en k=1 tramo

La idea es buscar el óptimo para llegar a cada ciudad en 1 tramo.

Para hacer el análisis tomamos cada ciudad, y por cada ciudad analizamos cada tramo que llega a la misma.

Ciudad 1

Para la ciudad 1 vemos que no llega ningún tramo, por lo tanto vamos con la siguiente ciudad.

Ciudad 2

A la ciudad 2 vemos que llegan 2 tramos, “T1/8 a 18” y “T3/12 a 13”.

Tramo t=T1:

* “Intentamos obtener de M[k-1][t.ciudad\_desde], la solución O que tenga horario de llegada a t.ciudad\_desde más próximo a t.horario\_desde.” En nuestro caso es M[0][1] y vemos que la solución (8:00, 0, NULL) es la que tiene horario de llegada a 1 (8:00) más próximo al horario de salida del tramo T1 (8:00).
* Como existe significa que encontramos una solución para llegar a la ciudad 2 utilizando k=1 tramos, por lo tanto insertamos dicha solución al listado M[k=1][x=2]: I2(18:00, 10, I1)

Tramo t=T3:

* “Intentamos obtener de M[k-1][t.ciudad\_desde], la solución O que tenga horario de llegada a t.ciudad\_desde más próximo a t.horario\_desde.” En nuestro caso es M[0][3] y vemos que no hay ninguna solución en M[0][3]. Por lo tanto descartamos el tramo.

Ciudad 3

A la ciudad 3 vemos que llega 1 tramo, “T2/10 a 11”.

Tramo t=T2:

* “Intentamos obtener de M[k-1][t.ciudad\_desde], la solución O que tenga horario de llegada a t.ciudad\_desde más próximo a t.horario\_desde.” En nuestro caso es M[0][1] y vemos que la solución I1(8:00, 0, NULL) es la que tiene horario de llegada a 1 (8:00) más próximo al horario de salida del tramo T1 (10:00).
* Como existe significa que encontramos una solución para llegar a la ciudad 3 utilizando k=1 tramos, por lo tanto insertamos dicha solución al listado M[k=1][x=3]: I3(11:00, 1, I1)

Resultados obtenidos para k=1

Vemos que la matriz tiene el siguiente estado:

|  |  |  |  |
| --- | --- | --- | --- |
| **# Tramos\Ciudad** | **1** | **2** | **3** |
| **0** | [I1(8:00, 0, NULL)] | [] | [] |
| **1** | [] | [I2(18:00, 10, I1)] | [I3(11:00, 1, I1)] |
| **2** | [] | [] | [] |

Análisis de cómo llegar a la ciudad X en k=2 tramos

La idea es buscar el óptimo para llegar a cada ciudad en 2 tramos.

Para hacer el análisis tomamos cada ciudad, y por cada ciudad analizamos cada tramo que llega a la misma.

Ciudad 1

Para la ciudad 1 vemos que no llega ningún tramo, por lo tanto vamos con la siguiente ciudad.

Ciudad 2

A la ciudad 2 vemos que llegan 2 tramos, “T1/8 a 18” y “T3/12 a 13”.

Tramo t=T1:

* “Intentamos obtener de M[k-1][t.ciudad\_desde], la solución O que tenga horario de llegada a t.ciudad\_desde más próximo a t.horario\_desde.” En nuestro caso es M[1][1] y vemos que no hay ninguna solución en M[1][1]. Por lo tanto descartamos el tramo.

Tramo t=T3:

* “Intentamos obtener de M[k-1][t.ciudad\_desde], la solución O que tenga horario de llegada a t.ciudad\_desde más próximo a t.horario\_desde.” En nuestro caso es M[1][3] y vemos que la solución I3(11:00, 2, I1) es la que tiene horario de llegada a 3 (11:00) más próximo al horario de salida del tramo T3 (12:00).
* Como existe significa que encontramos una solución para llegar a la ciudad 2 utilizando k=2 tramos, por lo tanto insertamos dicha solución al listado M[k=2][x=2]: I4(13:00, 3, I3)

Ciudad 3

A la ciudad 3 vemos que llega 1 tramo, “T2/10 a 11”.

Tramo t=T2:

* “Intentamos obtener de M[k-1][t.ciudad\_desde], la solución O que tenga horario de llegada a t.ciudad\_desde más próximo a t.horario\_desde.” En nuestro caso es M[1][1] y vemos que no hay ninguna solución en M[1][1]. Por lo tanto descartamos el tramo.

Resultados obtenidos para k=2

Vemos que la matriz tiene el siguiente estado:

|  |  |  |  |
| --- | --- | --- | --- |
| **# Tramos\Ciudad** | **1** | **2** | **3** |
| **0** | [I1(8:00, 0, NULL)] | [] | [] |
| **1** | [] | [I2(18:00, 10, I1)] | [I3(11:00, 2, I1)] |
| **2** | [] | [I4(13:00, 3, I3)] | [] |

Obtención del óptimo

Finalmente solo resta para cada k de 0 a 2 recorrer la columna de la ciudad destino (en nuestro caso la ciudad 2).

Tomaremos el ítem solución que tenga el menor horario de llegada. En caso de existir dos ítems con igual horario simplemente elegiremos el que tenga menor tiempo total.

En nuestro caso elegiremos a I4 el cual indica que se puede llegar a las ciudad 2 a las 13 hs invirtiendo tan solo 3 horas.

Reconstrucción de la solución óptima

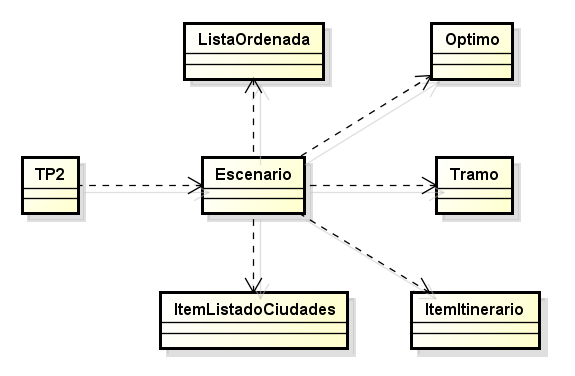
Luego de elegido el ítem solución, simplemente será seguir los punteros a la solución anterior.

La solución en nuestro caso será:

I3 -> I4

Donde cada ítem solución contiene el tramo utilizado.

Diagrama de clases



Justificación de órdenes del lenguaje y librerías

El lenguaje utilizado para la implementación del trabajo práctico es python.

Los órdenes referentes a operaciones realizadas con las estructuras provistas por el lenguaje y librerías se obtuvieron a partir de la siguiente documentación (en todos los casos se trabajo con los casos pesimistas):

* Órdenes para estructuras de datos nativas:
  + <https://wiki.python.org/moin/TimeComplexity>
* Documentación de módulo estándar de colas de prioridad:
  + <https://docs.python.org/2/library/heapq.html>
* Documentación de módulo estándar para realizar búsquedas binarias e inserción ordenada:
  + <https://docs.python.org/2/library/bisect.html>

Código Fuente

Para acceder a la totalidad del código fuente puede clonar el siguiente repositorio: <https://github.com/sebasgoldberg/7529-tp2>

A continuación se incorpora el código fuente del trabajo práctico, y para ganar claridad y disminuir la cantidad de código se excluyeron los casos de prueba de cada módulo.

La mayoría de los métodos fueron documentados de forma de registrar el orden de los algoritmos implementados.

tp2.py

|  |
| --- |
| *#!/usr/bin/python* |
| *# coding=utf-8* |
|  |
| **from** escenario **import** Escenario |
| **import** sys |
|  |
| **class** **TP2**: |
|  |
| **def** **\_\_init\_\_**(self, filepath): |
| """ |
| O(sum(max(ni,mi+ri)\*log(ni))) |
| i: Denota el escenario i |
| n: Cantidad de ciudades |
| m: Cantidad de trenes |
| r: Cantidad de ciudades visitadas por los m trenes |
| """ |
| self.escenarios **=** [] |
| **with** **open**(filepath) **as** f: |
| cantidad\_escenarios **=** **int**(f.readline().strip()) |
| *# O(sum(max(ni,mi+ri)\*log(ni)))* |
| **for** i **in** **xrange**(cantidad\_escenarios): |
| self.escenarios.append(Escenario(f)) *# O(max(n,m+r)\*log(n))* |
|  |
| **def** **resolver**(self): |
| """ |
| O(sum(max(ni\*pi\*log(pi) + pi\*\*2, ni\*\*2))) |
| i: Denota el escenario i |
| n: Cantidad de ciudades |
| p: Cantidad de tramos (p = r-n) |
| r: Cantidad de ciudades visitadas por los m trenes |
| m: Cantidad de trenes |
| """ |
| **for** escenario **in** self.escenarios: |
| escenario.resolver() *# O(max(n\*p\*log(p) + p\*\*2, n\*\*2))* |
|  |
| **def** **imprimir\_solucion**(self): |
| """ |
| O(sum(pi\*ni)) |
| i: Denota el escenario i |
| n: Cantidad de ciudades |
| p: Cantidad de tramos (p = r-n) |
| r: Cantidad de ciudades visitadas por los m trenes |
| m: Cantidad de trenes |
| """ |
| **for** i **in** **xrange**(**len**(self.escenarios)): |
| escenario **=** self.escenarios[i] |
| **print** 'Escenario %s' **%** (i**+**1) |
| escenario.imprimir\_solucion() *# O(p\*n)* |
| **print** |
|  |
|  |
|  |
| **def** **reporte\_tp2**(): |
| **for** filepath **in** sys.argv[1:]: |
| tp2 **=** TP2(filepath) |
| tp2.resolver() |
| tp2.imprimir\_solucion() |
|  |
|  |
| **if** \_\_name\_\_ **==** '\_\_main\_\_': |
| **if** **len**(sys.argv) **==** 1: |
| unittest.main() |
| **else**: |
| reporte\_tp2() |

escenario.py

|  |
| --- |
| *#!/usr/bin/python* |
| *# coding=utf-8* |
|  |
| **from** tramo **import** Tramo |
| **from** lista\_ordenada **import** ListaOrdenada, ElementoNoEncontrado |
| **from** optimo **import** Optimo |
| **from** horario **import** **\*** |
| **import** sys |
|  |
| **class** **OptimoNoEncontrado**(**Exception**): |
| **pass** |
|  |
|  |
| **class** **ItemItinerario**: |
|  |
| **def** **\_\_init\_\_**(self, hora, ciudad): |
| """ |
| O(1) |
| """ |
| self.hora **=** hora |
| self.ciudad **=** ciudad |
|  |
| **def** **\_\_str\_\_**(self): |
| **return** '%s %s' **%** (self.hora, self.ciudad) |
|  |
| **def** **\_\_cmp\_\_**(self, other): |
| **if** self.hora **<** other.hora: |
| **return** **-**1 |
| **if** self.hora **>** other.hora: |
| **return** 1 |
| **if** self.ciudad **<** other.ciudad: |
| **return** **-**1 |
| **if** self.ciudad **>** other.ciudad: |
| **return** 1 |
| **return** 0 |
|  |
|  |
| **class** **ItemListadoCiudades**: |
|  |
| **def** **\_\_init\_\_**(self, nombre, id): |
| """ |
| O(1) |
| """ |
| self.nombre **=** nombre |
| self.id **=** **id** |
|  |
| **def** **\_\_cmp\_\_**(self, other): |
| """ |
| O(1) |
| """ |
| **if** self.nombre **<** other.nombre: |
| **return** **-**1 |
| **if** self.nombre **>** other.nombre: |
| **return** 1 |
| **return** 0 |
|  |
| **def** **\_\_str\_\_**(self): |
| **return** '(%s, %s)' **%** (self.nombre, self.id) |
|  |
|  |
| **class** **Escenario**: |
|  |
| **def** **\_\_init\_\_**(self, f**=None**): |
| """ |
| O(max(n,m+r)\*log(n)) |
| n: Cantidad de ciudades |
| m: Cantidad de trenes |
| ri: Cantidad de ciudades que visita el tren i |
| r: Cantidad de ciudades visitadas por los m trenes |
| """ |
|  |
| self.ciudades **=** [] *# O(1)* |
| self.id\_por\_ciudad **=** ListaOrdenada() *# O(1)* |
| self.tramos\_por\_ciudad\_destino **=** [] *# O(1)* |
|  |
| **if** f **is** **None**: |
| **return** |
|  |
| cantidad\_ciudades **=** **int**(f.readline().strip()) *# O(1)* |
| *# O(n\*log(n))* |
| **for** i **in** **xrange**(cantidad\_ciudades): *# n* |
| ciudad **=** f.readline().strip() *# O(1)* |
| self.add\_ciudad(ciudad) *# O(log(n))* |
|  |
| cantidad\_trenes **=** **int**(f.readline().strip()) *# O(1)* |
|  |
| *# O(m\*log(n) + sum(ri\*log(n))) = O(m\*log(n)+r\*log(n)) = O((m+r)\*log(n))* |
| **for** tren **in** **xrange**(cantidad\_trenes): *# m* |
| cantidad\_ciudades\_tren **=** **int**(f.readline().strip()) *# O(1)* |
| horario\_salida, ciudad\_origen **=** f.readline().strip().split(' ', 1) *# O(1)* |
| horario\_salida **=** normalizar\_horario(horario\_salida) *# O(1)* |
| ciudad\_origen **=** self.get\_id\_ciudad(ciudad\_origen) *# O(log(n))* |
| **for** i **in** **xrange**(cantidad\_ciudades\_tren**-**1): *# ri* |
| horario\_llegada, ciudad\_destino **=** f.readline().strip().split(' ', 1) *# O(1)* |
| horario\_llegada **=** normalizar\_horario(horario\_llegada) *# O(1)* |
| ciudad\_destino **=** self.get\_id\_ciudad(ciudad\_destino) *# O(log(n))* |
| self.add\_tramo(tren, ciudad\_origen, ciudad\_destino, |
| horario\_salida, horario\_llegada) |
| horario\_salida, ciudad\_origen **=** horario\_llegada, ciudad\_destino *# O(1)* |
|  |
| self.set\_condiciones\_iniciales( |
| horario\_inicial **=** normalizar\_horario(f.readline().strip()), *# O(1)* |
| ciudad\_origen **=** self.get\_id\_ciudad(f.readline().strip()), *# O(log(n))* |
| ciudad\_destino **=** self.get\_id\_ciudad(f.readline().strip()) *# O(log(n))* |
| ) |
|  |
| **def** **add\_ciudad**(self, ciudad): |
| """ |
| O(log(n)) |
| """ |
| self.id\_por\_ciudad.insert(ItemListadoCiudades(ciudad, **len**(self.ciudades))) *# O(log(n))* |
| self.ciudades.append(ciudad) *# O(1)* |
| self.tramos\_por\_ciudad\_destino.append([]) *# O(1)* |
|  |
| **def** **add\_tramo**(self, tren, ciudad\_origen, ciudad\_destino, |
| horario\_salida, horario\_llegada): |
| """ |
| O(1) |
| """ |
| self.tramos\_por\_ciudad\_destino[ciudad\_destino].append( |
| Tramo(tren,ciudad\_origen, ciudad\_destino, |
| horario\_salida, horario\_llegada)) *# O(1)* |
|  |
| **def** **add\_tramo\_from\_nombre\_ciudades**(self, tren, ciudad\_origen, ciudad\_destino, |
| horario\_salida, horario\_llegada): |
| """ |
| O(log(n)) |
| """ |
| ciudad\_origen **=** self.get\_id\_ciudad(ciudad\_origen) *# O(log(n))* |
| ciudad\_destino **=** self.get\_id\_ciudad(ciudad\_destino) *# O(log(n))* |
| self.tramos\_por\_ciudad\_destino[ciudad\_destino].append( |
| Tramo(tren,ciudad\_origen, ciudad\_destino, |
| horario\_salida, horario\_llegada)) *# O(1)* |
|  |
| **def** **set\_condiciones\_iniciales**(self, horario\_inicial, ciudad\_origen, ciudad\_destino): |
| """ |
| O(1) |
| """ |
| self.horario\_inicial **=** horario\_inicial |
| self.ciudad\_origen **=** ciudad\_origen |
| self.ciudad\_destino **=** ciudad\_destino |
|  |
| **def** **get\_id\_ciudad**(self, ciudad): |
| """ |
| O(log(n)) |
| """ |
| **return** self.id\_por\_ciudad.get\_item(ItemListadoCiudades(ciudad,**None**)).id |
|  |
| **def** **get\_tramos\_a\_ciudad**(self, id\_ciudad\_destino): |
| """ |
| O(1) |
| """ |
| **return** self.tramos\_por\_ciudad\_destino[id\_ciudad\_destino] |
|  |
| **def** **resolver**(self): |
| """ |
| O(max(n\*p\*log(p) + p\*\*2, n\*\*2)) |
| n: Cantidad de ciudades |
| p: Cantidad de tramos (p = r-n) |
| r: Cantidad de ciudades visitadas por los m trenes |
| self.solucion es una matriz donde la primer componente es la cantidad |
| máxima de tramos necesarios para llegar a una ciudad, la segunda |
| componente es la ciudad destino, y la tercera es la solucion optima |
| para un determinado horario de llegada. |
| """ |
|  |
| self.solucion **=** [] *# O(1)* |
|  |
| *# O(n\*\*2)* |
| **for** k **in** **xrange**(**len**(self.ciudades)): *# n* |
| self.solucion.append([]) *# O(1)* |
| **for** ciudad **in** **xrange**(**len**(self.ciudades)): *# n* |
| self.solucion[k].append(ListaOrdenada(permitir\_repetidos**=True**)) *# O(1)* |
|  |
| *# O(1)* |
| self.solucion[0][self.ciudad\_origen].insert(Optimo( |
| self.horario\_inicial, Tramo(**None**, self.ciudad\_origen, |
| self.ciudad\_origen, self.horario\_inicial, |
| self.horario\_inicial), 0)) |
|  |
|  |
| *# O(sum(p\*log(p)) + sum(pj\*p)) = O(n\*p\*log(p) + p\*\*2)* |
| **for** k **in** **xrange**(1,**len**(self.ciudades)): *# n-1* |
| *# O(sum(pj\*log(pj)) + sum(sum(pi))) = O(p\*log(p)+ pj\*p)* |
| **for** ciudad **in** **xrange**(**len**(self.ciudades)): *# n* |
| *# O(pj\*(log(pj) + sum(log(pi))), donde pi son los tramos que llegan a cada ciudad i que llega a j.* |
| **for** tramo **in** self.get\_tramos\_a\_ciudad(ciudad): *# pj (los tramos que llegan a la ciudad j)* |
| **try**: |
| """ |
| O(log(si)) (si cantidad de soluciones para la ciudad i en k-1 tramos, |
| si <= pi ya que habra como máximo tantas soluciones hacia i como tramos |
| lleguen a i (ver insert más abajo)) |
| Se obtiene la solucion más cercana al horario de salida desde la |
| ciudad origen a la destino en k-1 tramos. |
| """ |
| optimoAnterior **=** self.solucion[k**-**1][tramo.ciudad\_origen].get\_anterior\_mas\_cercano( |
| Optimo(tramo.horario\_salida, **None**, **None**)) |
| **except** ElementoNoEncontrado: |
| **continue** |
| tiempo\_total **=** optimoAnterior.tiempo\_total **+** (tramo.horario\_llegada **-** |
| optimoAnterior.tramo.horario\_llegada) *# O(1)* |
| self.solucion[k][ciudad].insert( |
| Optimo(tramo.horario\_llegada, |
| tramo, tiempo\_total, optimoAnterior)) *# O(log(sj)) = O(log(pj))* |
|  |
| self.optimos **=** [] *# O(1)* |
| tiempo\_total\_optimo **=** **None** *# O(1)* |
| horario\_llegada\_optimo **=** **None** |
| *# O(sum(pj)) = O(p)* |
| **for** k **in** **xrange**(**len**(self.ciudades)): *# n* |
| **for** solucion **in** self.solucion[k][self.ciudad\_destino].iteritems(): *# sj = pj* |
| **if** horario\_llegada\_optimo **is** **None**: *# O(1)* |
| self.optimos **=** [solucion] *# O(1)* |
| tiempo\_total\_optimo **=** solucion.tiempo\_total *# O(1)* |
| horario\_llegada\_optimo **=** solucion.horario\_llegada |
| **elif** solucion.horario\_llegada **<** horario\_llegada\_optimo: |
| self.optimos **=** [solucion] *# O(1)* |
| tiempo\_total\_optimo **=** solucion.tiempo\_total *# O(1)* |
| horario\_llegada\_optimo **=** solucion.horario\_llegada |
| **elif** solucion.horario\_llegada **==** horario\_llegada\_optimo: |
| **if** solucion.tiempo\_total **==** tiempo\_total\_optimo: *# O(1)* |
| self.optimos.append(solucion) *# O(1)* |
| **elif** solucion.tiempo\_total **<** tiempo\_total\_optimo: *# O(1)* |
| self.optimos **=** [solucion] *# O(1)* |
| tiempo\_total\_optimo **=** solucion.tiempo\_total *# O(1)* |
|  |
| **def** **get\_itinerarios\_optimos**(self): |
| """ |
| O(p\*n) |
| """ |
| itinerarios **=** [] *# O(1)* |
| **if** **len**(self.optimos) **==** 0: *# O(1)* |
| **return** itinerarios *# O(1)* |
| *# O(p\*n)* |
| **for** optimo **in** self.optimos: *# p* |
| itinerario **=** [] *# O(1)* |
| solucion **=** [] *# O(1)* |
| **while** optimo.optimoAnterior **is** **not** **None**: *# n* |
| solucion.insert(0,optimo) *# O(1)* |
| optimo **=** optimo.optimoAnterior *# O(1)* |
| itinerario.append(ItemItinerario( |
| format\_horario(solucion[0].tramo.horario\_salida), |
| self.ciudades[solucion[0].tramo.ciudad\_origen])) *# O(1)* |
| **for** optimo **in** solucion[:**-**1]: *# n* |
| itinerario.append(ItemItinerario( |
| format\_horario(optimo.tramo.horario\_llegada), |
| self.ciudades[optimo.tramo.ciudad\_destino])) *# O(1)* |
| itinerario.append(ItemItinerario( |
| format\_horario(solucion[**-**1].tramo.horario\_llegada), |
| self.ciudades[solucion[**-**1].tramo.ciudad\_destino])) *# O(1)* |
| itinerarios.append(itinerario) *# O(1)* |
| **return** itinerarios *# O(1)* |
|  |
|  |
| **def** **imprimir\_solucion**(self): |
| """ |
| O(p\*n) |
| """ |
| itinerarios **=** self.get\_itinerarios\_optimos() *# O(p\*n)* |
| **if** **len**(itinerarios) **==** 0: |
| **print** 'Sin combinaciones posibles' |
| **return** |
| **for** itinerario **in** itinerarios: |
| **print** 'Salida %s %s' **%** ( |
| itinerario[0].hora, itinerario[0].ciudad) |
| **for** item **in** itinerario[1:**-**1]: |
| **print** 'Trasbordo %s %s' **%** ( |
| item.hora, item.ciudad) |
| **print** 'Arribo %s %s' **%** ( |
| itinerario[**-**1].hora, itinerario[**-**1].ciudad) |

optimo.py

|  |
| --- |
| *#!/usr/bin/python* |
| *# coding=utf-8* |
|  |
| **class** **Optimo**: |
|  |
| **def** **\_\_init\_\_**(self, horario\_llegada, tramo, tiempo\_total, optimoAnterior**=None**): |
| self.horario\_llegada **=** horario\_llegada |
| self.tramo **=** tramo |
| self.tiempo\_total **=** tiempo\_total |
| self.optimoAnterior **=** optimoAnterior |
|  |
| **def** **\_\_cmp\_\_**(self, other): |
| **if** self.horario\_llegada **>** other.horario\_llegada: |
| **return** 1 |
| **if** self.horario\_llegada **<** other.horario\_llegada: |
| **return** **-**1 |
| **return** 0 |
|  |
| **def** **\_\_str\_\_**(self): |
| **return** '%s [Total: %s]' **%** (self.tramo, self.tiempo\_total) |

tramo.py

|  |
| --- |
| *#!/usr/bin/python* |
| *# coding=utf-8* |
|  |
| **class** **Tramo**: |
|  |
| **def** **\_\_init\_\_**(self, tren, ciudad\_origen, ciudad\_destino, |
| horario\_salida, horario\_llegada): |
| """ |
| O(1) |
| """ |
| self.tren **=** tren |
| self.ciudad\_origen **=** ciudad\_origen |
| self.ciudad\_destino **=** ciudad\_destino |
| self.horario\_salida **=** horario\_salida |
| self.horario\_llegada **=** horario\_llegada |
|  |
| **def** **\_\_str\_\_**(self): |
| """ |
| O(1) |
| """ |
| **return** '[Tren: %s] [Origen: %s] [Destino: %s] [Salida: %s] [Llegada: %s]' **%** ( |
| self.tren, self.ciudad\_origen, self.ciudad\_destino, |
| self.horario\_salida, self.horario\_llegada) |

lista\_ordenada.py

|  |
| --- |
| *#!/usr/bin/python* |
| *# coding=utf-8* |
|  |
| **import** bisect |
|  |
| **class** **ElementoNoEncontrado**(**Exception**): |
|  |
| **def** **\_\_init\_\_**(self, elemento): |
| **Exception**.**\_\_init\_\_**(self, |
| "El elemento %s no se ha encontrado en la lista" **%** elemento) |
|  |
| **class** **ListaOrdenada**(): |
|  |
| **def** **\_\_init\_\_**(self, permitir\_repetidos**=False**): |
| self.lista **=** [] |
| self.permitir\_repetidos **=** permitir\_repetidos |
|  |
| **def** **iteritems**(self): |
| """ |
| O(1) |
| """ |
| **return** **iter**(self.lista) |
|  |
| **def** **insert**(self, node): |
| """ |
| O(n\*log(n)) |
| """ |
| i **=** bisect.bisect\_left(self.lista, node) |
| **if** **not** self.permitir\_repetidos: |
| **if** i <> **len**(self.lista) **and** self.lista[i] **==** node: |
| **raise** **Exception**('El nodo %s ya existe en la lista.' **%** node) |
| **return** self.lista.insert(i, node) |
|  |
| **def** **has**(self, node): |
| """ |
| O(log(n)) |
| """ |
| i **=** bisect.bisect\_left(self.lista, node) |
| **if** i <> **len**(self.lista) **and** self.lista[i] **==** node: |
| **return** **True** |
| **return** **False** |
|  |
| **def** **get\_item**(self, item): |
| """ |
| O(log(n)) |
| """ |
| i **=** bisect.bisect\_left(self.lista, item) |
| **if** i <> **len**(self.lista) **and** self.lista[i] **==** item: |
| **return** self.lista[i] |
| **raise** ElementoNoEncontrado(item) |
|  |
|  |
| **def** **get\_anterior\_mas\_cercano**(self, x): |
| """ |
| O(log(n)) |
| """ |
| **if** **len**(self.lista) **==** 0: |
| **raise** ElementoNoEncontrado(x) |
| i **=** bisect.bisect\_left(self.lista, x) |
| **if** i **==** **len**(self.lista): |
| **return** self.lista[i**-**1] |
| **if** self.lista[i] **==** x: |
| **return** self.lista[i] |
| **if** (i**-**1) **>=** 0: |
| **return** self.lista[i**-**1] |
| **raise** ElementoNoEncontrado(x) |
|  |
|  |
| **def** **intersection**(self, other): |
| """ |
| O(len(self.lista)+len(other.lista)) = O(n1+n2) |
| """ |
| len\_self **=** **len**(self.lista) |
| len\_other **=** **len**(other.lista) |
| i\_self **=** 0 |
| i\_other **=** 0 |
| intersection **=** [] |
|  |
| **while** i\_self **<** len\_self **and** i\_other **<** len\_other: |
| x\_self **=** self.lista[i\_self] |
| x\_other **=** other.lista[i\_other] |
|  |
| **if** x\_self **<** x\_other: |
| i\_self **+=** 1 |
| **elif** x\_self **>** x\_other: |
| i\_other **+=** 1 |
| **else**: |
| intersection.append(x\_self) |
| i\_self **+=** 1 |
| i\_other **+=** 1 |
|  |
| **return** intersection |