Mathias HATEAU n°29
Corentin DOMENICHINI n°18

L'UTILISATION DE MODÈLES INFORMATIQUES DANS LA GESTION DU TRAFIC AÉRIEN MONDIAL

I. Pourquoi ce sujet ?
II. Déroulé du projet
III. Bilan et conclusions

I. Pourquoi ce sujet ?

I. Pourquoi ce sujet ?

https://www.shutterstock.com/it/image-vector/world-travel-map-airplanes-flight-routes-1531773191

II. Déroulé du projet

```
[mat@mat TIPE]$ /bin/python /home/mat/Documents/MP2I/I
Temps pour  5 sommets
Temps de génération aléatoire  7.379376173019409
starting update
graph time Bellman-Ford : 1.3532824516296387
graph time Dijkstra adj_list : 1.2525100708007812
graph time Dijkstra fib_heap : 0.4953649044036865
Temps pour  6 sommets
Temps de génération aléatoire  10.425267219543457
starting update
graph time Bellman-Ford : 2.32540225982666
graph time Dijkstra adj_list : 1.6937296390533447
graph time Dijkstra fib_heap : 0.628594160079956
Temps pour  7 sommets
Temps de génération aléatoire  12.501460075378418
starting update
graph time Bellman-Ford : 3.5870883464813232
graph time Dijkstra adj_list : 2.129509210586548
graph time Dijkstra fib_heap : 0.7727963924407959
Temps pour  8 sommets
Temps de génération aléatoire  16.36824917793274
starting update
graph time Bellman-Ford : 5.3444085121154785
graph time Dijkstra adj_list : 2.6681642532348633
graph time Dijkstra fib_heap : 0.9333209991455078
Temps pour  9 sommets
Temps de génération aléatoire  20.92163395881653
starting update
graph time Bellman-Ford : 7.652991533279419
graph time Dijkstra adj_list : 3.1685922145843506
graph time Dijkstra fib_heap : 1.1082944869995117
Temps pour  10 sommets
Temps de génération aléatoire  26.055927515029907
starting update
graph time Bellman-Ford : 10.6537926197052
graph time Dijkstra adj_list : 3.8571767807006836
graph time Dijkstra fib_heap : 1.3171381950378418
total: 144.59491610527039
```

Python > way.py > Graph > dijkstra

```python
class Graph():

    def __init__(self, V):
        self.V = V
        self.graph = defaultdict(list)
    def addEdge(self, src, dest, weight):
        newNode = [dest, weight]
        self.graph[src].insert(0, newNode)
        newNode = [src, weight]
        self.graph[dest].insert(0, newNode)
    def dijkstra(self, src):
        V = self.V
        dist = []
        minHeap = Heap()
        for v in range(V):
            dist.append(1e7)
            minHeap.array.append( minHeap.newMinHeapNode(v, dist[v]))
            minHeap.pos.append(v)
        minHeap.pos[src] = src
        dist[src] = 0
        minHeap.decreaseKey(src, dist[src])
        minHeap.size = V
        while minHeap.isEmpty() == False:
            newHeapNode = minHeap.extractMin()
            u = newHeapNode[0]
            for pCrawl in self.graph[u]:
                v = pCrawl[0]
                if (minHeap.isInMinHeap(v) and
                        dist[u] != 1e7 and \
                    pCrawl[1] + dist[u] < dist[v]):
                        dist[v] = pCrawl[1] + dist[u]
                        minHeap.decreaseKey(v, dist[v])

        printArr(dist,V)

class Graph_list:
    def __init__(self, size):
        self.adj_matrix = [[0] * size for _ in range(size)]
        self.size = size
        self.vertex_data = [''] * size

    def add_edge(self, u, v, weight):
```

III. Analyse de la complexité temporelle

Dijkstra (tas de fibonacci) : $\Theta((E + V) \log(V))$
Dijkstra (tableau simple) : $\Theta(V^2)$
Bellman-Ford : $\Theta(E\ V)$

III. Résultats expérimentaux

Nombre de sommets / Algorithmes	5	6	7	8	9	10
Bellman-Ford	1.36045002 s 13,60 µs	2.30745053 s 23,07 µs	3.70744323 s 37,07 µs	5.39601778 s 53,96 µs	7.67675113 s 76,77 µs	10.5696604 s 105,6 µs
Dijkstra tas de fibonacci	1.25878286 s 12,59 µs	2.37394905 s 23,73 µs	2.14571332 s 21,46 µs	2.61112093 s 26,11 µs	3.18913388 s 31,89 µs	3.83882474 s 38,39 µs
Dijkstra tableau	0.47695326 s 4,77 µs	0.62835907 s 6,28 µs	0.77189493 s 7,72 µs	0.92861390 s 9,29 µs	1.09547448 s 10,95 µs	1.29672789 s 12,97 µs