CoursMagistral_8

October 18, 2019

1 Table of Contents

2 ALGO1 : Introduction à l'algorithmique

- Page du cours : https://perso.crans.org/besson/teach/info1_algo1_2019/
- Magistère d'Informatique de Rennes - ENS Rennes - Année 2019/2020
- Intervenants :
  - Cours : Lilian Besson
  - Travaux dirigés : Raphaël Truffet
- Références :
  - Open Data Structures

3 Cours Magistral 8

- Ce cours traite du problème du flot maximal.

3.1 Algorithme de Ford-Fulkerson

- On va utiliser l'exemple donné sur la page Wikipédia :
- Cf. la page Wikipédia.

3.1.1 Etape de base : augmente un chemin

```
In [14]: def augment(graph, capacity, flow, val, u, target, visit):
             """ Find an augmenting path from u to target with value at most val."""
             visit[u] = True
             if u == target:
                 return val
             for v in graph[u]:
                 cuv = capacity[u][v]
                 if not visit[v] and cuv > flow[u][v]:  # reachable arc
                     res = min(val, cuv - flow[u][v])
                     delta = augment(graph, capacity, flow, res, v, target, visit)
                     if delta > 0:
                         flow[u][v] += delta            # augment flow
                         flow[v][u] -= delta
                         return delta
             return 0
```

3.1.2 Etape de base : ajoute un arc retour

```
In [15]: def add_reverse_arcs(graph, capac=None):
             """ Utility function for flow algorithms that need for every arc (u,v),
             the existence of an (v,u) arc, by default with zero capacity.
             graph can be in adjacency list, possibly with capacity matrix capac.
             or graph can be in adjacency dictionary, then capac parameter is ignored.

             :param capac: arc capacity matrix
             :param graph: in listlist representation, or in listdict representation,
             in this case capac is ignored
             :complexity: linear
             :returns: nothing, but graph is modified
             """
             for u, _ in enumerate(graph):
                 for v in graph[u]:
                     if u not in graph[v]:
                         if type(graph[v]) is list:
                             graph[v].append(u)
                             if capac:
                                 capac[v][u] = 0
                         else:
                             assert type(graph[v]) is dict
                             graph[v][u] = 0
```

3.1.3 Calcul du flot maximum

```
In [16]: def ford_fulkerson(graph, capacity, s, t):
             """Maximum flow by Ford-Fulkerson
```

```
    :param graph: directed graph in listlist or listdict format
    :param capacity: in matrix format or same listdict graph
    :param int s: source vertex
    :param int t: target vertex

    :returns: flow matrix, flow value
    :complexity: `O(|V|*|E|*max_capacity)`
    """
    add_reverse_arcs(graph, capacity)
    n = len(graph)
    flow = [[0] * n for _ in range(n)]
    INF = float('inf')
    while augment(graph, capacity, flow, INF, s, t, [False] * n) > 0:
        pass                            # work already done in _augment
    return (flow, sum(flow[s]))         # flow network, amount of flow
```

3.1.4 Exemple

Un exemple de graphe, comme celui utilisé dans la page Wikipédia :

```
In [17]: A, B, C, D = 0, 1, 2, 3
         graph = [
             [B, C], # A
             [C, D], # B
             [D], # C
             [], # D
         ]
         capacity = [
             [0, 1000, 1000, 0], # A
             [0, 0, 1, 1000], # B
             [0, 0, 0, 1000], # C
             [0, 0, 0, 0], # D
         ]

In [18]: ford_fulkerson(graph, capacity, A, D)

Out[18]: ([[0, 1000, 1000, 0],
           [-1000, 0, 0, 1000],
           [-1000, 0, 0, 1000],
           [0, -1000, -1000, 0]],
          2000)
```

On voit que le résultat est le même que celui donné dans l'exemple sur Wikipédia :

3.2 Conclusion

C'est bon pour aujourd'hui !