CoursMagistral_6

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 6

- Ce cours traite des algorithmes gloutons.
- Ce notebook sera concis, comparé aux précédents.

3.1 Rendu de monnaie

- Voir https://en.wikipedia.org/wiki/Change-making_problem ou https://fr.wikipedia.org/wiki/Probl%C3%A8me_du_rendu_de_monnaie

```
In [27]: def binary_coin_change(x, R):
             """Coin change

             :param x: table of non negative values
             :param R: target value
             :returns bool: True if there is a non negative linear combination
             of x that has value R
             :complexity: O(n*R)
             """
             if int(R) != R:  # we work with 1/100
                 R = int(R * 100)
                 x = [int(xi * 100) for xi in x]
             b = [False] * (R + 1)
             b[0] = True
             for xi in x:
                 for s in range(xi, R + 1):
                     b[s] |= b[s - xi]
             return b[R]
```

```
In [37]: def constructive_coin_change(values_of_coins, sum_to_find):
             """Coin change

             :param values_of_coins: table of non negative values
             :param sum_to_find: target value
             :returns bool: True if there is a non negative linear combination
             of x that has value R
             :complexity: O(n*R)
             """
             with_cents = False
             if int(sum_to_find) != sum_to_find:  # we work with 1/100
                 with_cents = True
                 sum_to_find = int(sum_to_find * 100)
                 values_of_coins = [int(pi * 100) for pi in values_of_coins]
             n = len(values_of_coins)
             number_of_coins = [0] * n
             values_of_coins = sorted(values_of_coins, reverse=True)
             current_sum = sum_to_find
             for i, pi in enumerate(values_of_coins):
                 assert pi > 0, "Error: a coin with value zero."
                 if pi > current_sum:
                     continue  # coin is too large, we continue
                 how_much_pi, rest = divmod(current_sum, pi)  # x // y, x % y
                 number_of_coins[i] = how_much_pi
                 print("For current sum = {}, coin = {}, was used {} times, now sum = {}.".forr
```

```
            current_sum = rest
        if current_sum != 0:
            raise ValueError("Could not write {} in the coin system {} with greedy method
        if with_cents:
            values_of_coins = [round(pi / 100, 2) for pi  in values_of_coins]
        return number_of_coins, values_of_coins
```

Avec les pièces des euros :

```
In [39]: billets = [500, 200, 100, 50, 20, 10, 5]
         pieces = [2, 1, 0.5, 0.2, 0.1, 0.05, 0.02, 0.01]
         euros = billets + pieces

         binary_coin_change(euros, 16.12)
         constructive_coin_change(euros, 16.12)
```

```
Out[39]: True
```

```
For current sum = 1612, coin = 1000, was used 1 times, now sum = 612.
For current sum = 612, coin = 500, was used 1 times, now sum = 112.
For current sum = 112, coin = 100, was used 1 times, now sum = 12.
For current sum = 12, coin = 10, was used 1 times, now sum = 2.
For current sum = 2, coin = 2, was used 1 times, now sum = 0.
```

```
Out[39]: ([0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0],
          [500.0,
           200.0,
           100.0,
           50.0,
           20.0,
           10.0,
           5.0,
           2.0,
           1.0,
           0.5,
           0.2,
           0.1,
           0.05,
           0.02,
           0.01])
```

```
In [41]: billets = [500, 200, 100, 50, 20, 10, 5]

         binary_coin_change(billets, 16)
         constructive_coin_change(billets, 16)
```

```
Out[41]: False
```

```
For current sum = 16, coin = 10, was used 1 times, now sum = 6.
For current sum = 6, coin = 5, was used 1 times, now sum = 1.
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-41-2f08541109fc> in <module>
  2
  3 binary_coin_change(billets, 16)
----> 4 constructive_coin_change(billets, 16)


<ipython-input-37-d17dafd42583> in constructive_coin_change(values_of_coins, sum_to_fi
 26         current_sum = rest
 27     if current_sum != 0:
---> 28         raise ValueError("Couldnot write {} in the coin system {} with greedy metho
 29     if with_cents:
 30         values_of_coins = [round(pi / 100, 2) for pi  in values_of_coins]


ValueError: Couldnot write 16 in the coin system [500, 200, 100, 50, 20, 10, 5] with gr
```

Avec un autre système de pièce :

```
In [ ]: billets = [19, 13, 7]
        pieces = [3, 2]
        weird = billets + pieces

        if binary_coin_change(weird, 47):
            constructive_coin_change(weird, 47)
        if binary_coin_change(weird, 49):
            constructive_coin_change(weird, 49)
        if binary_coin_change(weird, 50):
            constructive_coin_change(weird, 50)
```

```
For current sum = 47, coin = 19, was used 2 times, now sum = 9.
For current sum = 9, coin = 7, was used 1 times, now sum = 2.
For current sum = 2, coin = 2, was used 1 times, now sum = 0.
```

```
Out[ ]: ([2, 0, 1, 0, 1], [19, 13, 7, 3, 2])
```

```
For current sum = 49, coin = 19, was used 2 times, now sum = 11.
For current sum = 11, coin = 7, was used 1 times, now sum = 4.
For current sum = 4, coin = 3, was used 1 times, now sum = 1.
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-43-7bf348b1b994> in <module>
      6     constructive_coin_change(weird, 47)
      7 if binary_coin_change(weird, 49):
----> 8     constructive_coin_change(weird, 49)
      9 if binary_coin_change(weird, 50):
     10     constructive_coin_change(weird, 50)


<ipython-input-37-d17dafd42583> in constructive_coin_change(values_of_coins, sum_to_fi
     26         current_sum = rest
     27     if current_sum != 0:
---> 28         raise ValueError("Couldnot write {} in the coin system {} with greedy metho
     29     if with_cents:
     30         values_of_coins = [round(pi / 100, 2) for pi  in values_of_coins]


ValueError: Couldnot write 49 in the coin system [19, 13, 7, 3, 2] with greedy method.
```

Cette méthode gourmande ne marche pas pour tous les systèmes !

3.2 Structure “Union-Find”

3.2.1 Naïve

```
In [49]: class UnionFind:
             """Maintains a partition of {0, ..., n-1}
             """
             def __init__(self, n):
                 self.up_bound = list(range(n))

             def find(self, x_index):
                 """
                 :returns: identifier of part containing x_index
                 :complex_indexity: O(n) worst case, O(log n) in amortized cost.
                 """
                 if self.up_bound[x_index] == x_index:
                     return x_index
                 self.up_bound[x_index] = self.find(self.up_bound[x_index])
                 return self.up_bound[x_index]

             def union(self, x_index, y_index):
```

```
        """
        Merges part that contain x and part containing y
        :returns: False if x_index, y_index are already in same part
        :complexity: O(n) worst case, O(log n) in amortized cost.
        """
        repr_x = self.find(x_index)
        repr_y = self.find(y_index)
        if repr_x == repr_y:       # already in the same component
            return False
        self.up_bound[repr_x] = repr_y
        return True
```

Par exemple avec $S = \{0, 1, 2, 3, 4\}$ et les unions suivantes :

```
In [50]: S = [0,1,2,3,4]
         U = UnionFind(len(S))

In [51]: U.up_bound
         U.union(0, 2)
         U.up_bound

Out[51]: [0, 1, 2, 3, 4]

Out[51]: True

Out[51]: [2, 1, 2, 3, 4]

In [52]: U.up_bound
         U.union(2, 3)
         U.up_bound

Out[52]: [2, 1, 2, 3, 4]

Out[52]: True

Out[52]: [2, 1, 3, 3, 4]

In [53]: for i in S:
             U.find(i)

Out[53]: 3

Out[53]: 1

Out[53]: 3

Out[53]: 3

Out[53]: 4
```

Cela représente la partition $\{\{0, 2, 3\}, \{1\}, \{4\}\}$.

3.2.2 Avec compression de chemin

```
In [54]: class UnionFind_CompressedPaths:
             """Maintains a partition of {0, ..., n-1}
             """
             def __init__(self, n):
                 self.up_bound = list(range(n))
                 self.rank = [0] * n

             def find(self, x_index):
                 """
                 :returns: identifier of part containing x_index
                 :complex_indexity: O(inverse_ackerman(n))
                 """
                 if self.up_bound[x_index] == x_index:
                     return x_index
                 self.up_bound[x_index] = self.find(self.up_bound[x_index])
                 return self.up_bound[x_index]

             def union(self, x_index, y_index):
                 """
                 Merges part that contain x and part containing y
                 :returns: False if x_index, y_index are already in same part
                 :complexity: O(inverse_ackerman(n))
                 """
                 repr_x = self.find(x_index)
                 repr_y = self.find(y_index)
                 if repr_x == repr_y:       # already in the same component
                     return False
                 if self.rank[repr_x] == self.rank[repr_y]:
                     self.rank[repr_x] += 1
                     self.up_bound[repr_y] = repr_x
                 elif self.rank[repr_x] > self.rank[repr_y]:
                     self.up_bound[repr_y] = repr_x
                 else:
                     self.up_bound[repr_x] = repr_y
                 return True
```

Par exemple avec $S = \{0, 1, 2, 3, 4\}$ et les unions suivantes :

```
In [55]: S = [0,1,2,3,4]
         U = UnionFind_CompressedPaths(len(S))

In [56]: U.up_bound
         U.union(0, 2)
         U.up_bound

Out[56]: [0, 1, 2, 3, 4]

Out[56]: True
```

```
Out[56]: [0, 1, 0, 3, 4]

In [57]: U.up_bound
         U.union(2, 3)
         U.up_bound

Out[57]: [0, 1, 0, 3, 4]

Out[57]: True

Out[57]: [0, 1, 0, 0, 4]

In [58]: for i in S:
             U.find(i)

Out[58]: 0

Out[58]: 1

Out[58]: 0

Out[58]: 0

Out[58]: 4
```

Cela représente la partition $\{\{0,2,3\},\{1\},\{4\}\}$.

3.3 Algorithme de Kruskal

On utilise une des implémentations de la structure Union-Find, et le reste du code est très simple.

```
In [59]: def kruskal(graph, weight):
             """Minimum spanning tree by Kruskal

             :param graph: undirected graph in listlist or listdict format
             :param weight: in matrix format or same listdict graph
             :returns: list of edges of the tree
             :complexity: ``O(|E|log|E|)``
             """
             # a UnionFind with n singletons { {0}, {1}, ..., {n-1} }
             u_f = UnionFind(len(graph))
             edges = [ ]
             for u, _ in enumerate(graph):
                 for v in graph[u]:
                     # we add the edge (u, v) with weight w(u,v)
                     edges.append((weight[u][v], u, v))
             edges.sort()  # sort the edge in increasing order!
             min_span_tree = [ ]
```

```
    for w_idx, u_idx, v_idx in edges:  # O(|E|)
        if u_f.union(u_idx, v_idx):
            # u and v were not in the same connected component
            min_span_tree.append((u_idx, v_idx))
            # we add the edge (u, v) in the tree, now they are in the same connected
    return min_span_tree
```

3.4 Algorithme de Prim

3.4.1 File de priorité min

On peut utiliser les opérations `heappush` et `heappop` du module `heapq`. Ou notre implémentation maison des tas, qui permet d'avoir une opération `update` pour efficacement mettre à jour la priorité d'un élément.

```
In [60]: from heapq import heappop, heappush
```

```
In [61]: from heap_operations import OurHeap
```

3.4.2 Prim

```
In [97]: def prim(graph, weight, source=0):
             """Minimum spanning tree by Prim

             - param graph: directed graph, connex and non-oriented
             - param weight: in matrix format or same listdict graph
             - assumes: weights are non-negative

             - param source: source vertex
             - returns: distance table, precedence table

             - complexity: O(|S| + |A| log|A|)
             """
             n = len(graph)
             assert all(weight[u][v] >= 0 for u in range(n) for v in graph[u])
             prec = [None] * n
             cost = [float('inf')] * n
             cost[source] = 0
             # the difference with Dijsktra is that the heap starts with all the nodes!
             heap = OurHeap([])
             is_in_the_heap = [False for u in range(n)]
             for u in range(n):
                 heap.push((cost[u], u))
                 is_in_the_heap[u] = True
             while heap:
                 dist_node, node = heap.pop()       # Closest node from source
                 is_in_the_heap[node] = False
```

```
            # and there is no color white/gray/black
            # the node is always visited!
            for neighbor in graph[node]:
                if is_in_the_heap[neighbor] and cost[neighbor] >= weight[node][neighbor]:
                    old_cost = cost[neighbor]
                    cost[neighbor] = weight[node][neighbor]
                    prec[neighbor] = node
                    heap.update((old_cost, neighbor), (cost[neighbor], neighbor))
        # now we need to construct the min_spanning_tree
        edges = [ ]
        for u in range(n):
            if u != prec[u]:
                edges.append((u, prec[u]))
        return edges  # cost, prec
```

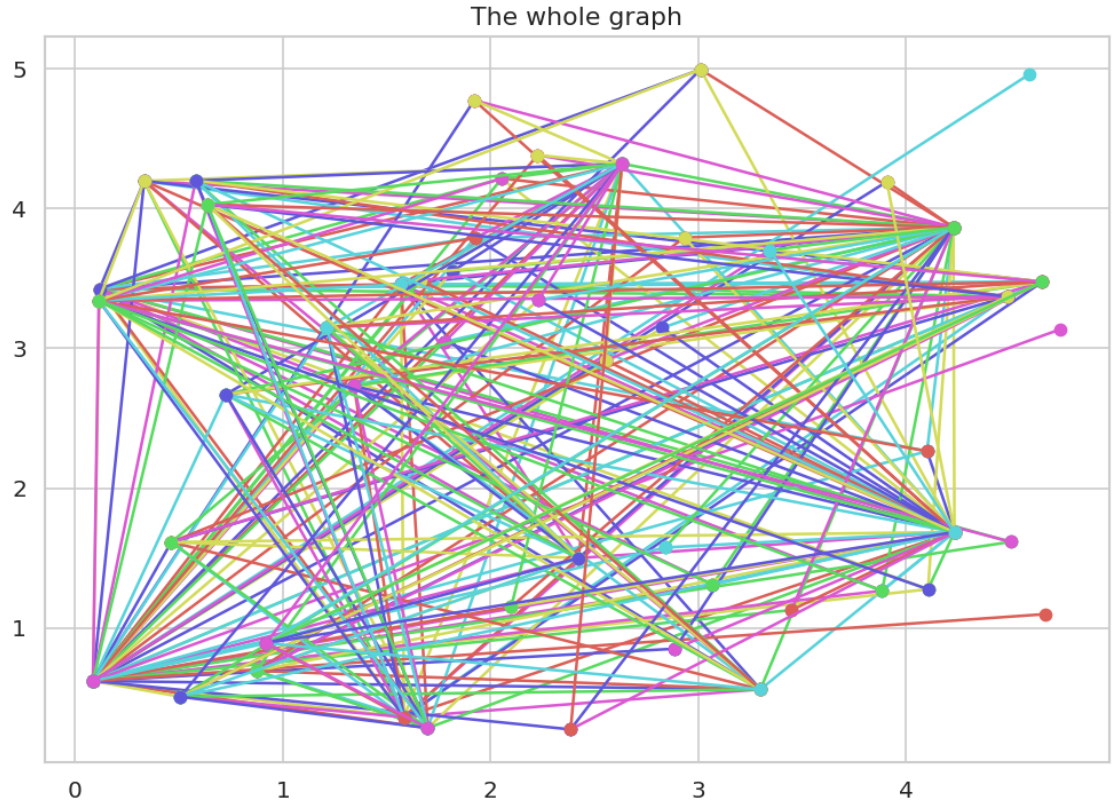
3.5 Illustrations

```
In [88]: import random
         import math

In [89]: def dist(a, b):
             """
             distance between point a and point b
             """
             return math.sqrt(sum([(a[i] - b[i]) * (a[i] - b[i]) for i in range(len(a))]))

In [90]: import matplotlib as mpl
         mpl.rcParams['figure.figsize'] = (10, 7)
         mpl.rcParams['figure.dpi'] = 120
         import matplotlib.pyplot as plt

         import seaborn as sns
         sns.set(context="notebook", style="whitegrid", palette="hls", font="sans-serif", font_

In [91]: N = 50
         points = [[random.random() * 5, random.random() * 5] for _ in range(N)]
         weight = [[dist(points[i], points[j]) for j in range(N)]
                   for i in range(N)]
         graph = [[j for j in range(N) if i != j] for i in range(N)]

In [92]: min_span_tree_kruskal = kruskal(graph, weight)

In [98]: min_span_tree_prim = prim(graph, weight)

In [94]: plt.figure()
         for u in points:
             for v in points:
```

```
            if u > v: break
            xu, yu = u
            xv, yv = v
            _ = plt.plot([xu, xv], [yu, yv], 'o-')
            # print("{} -- {}".format(points[u_idx], points[v_idx]))
        plt.title("The whole graph")
        plt.show()
```

Out[94]: <Figure size 1200x840 with 0 Axes>

Out[94]: Text(0.5, 1.0, 'The whole graph')

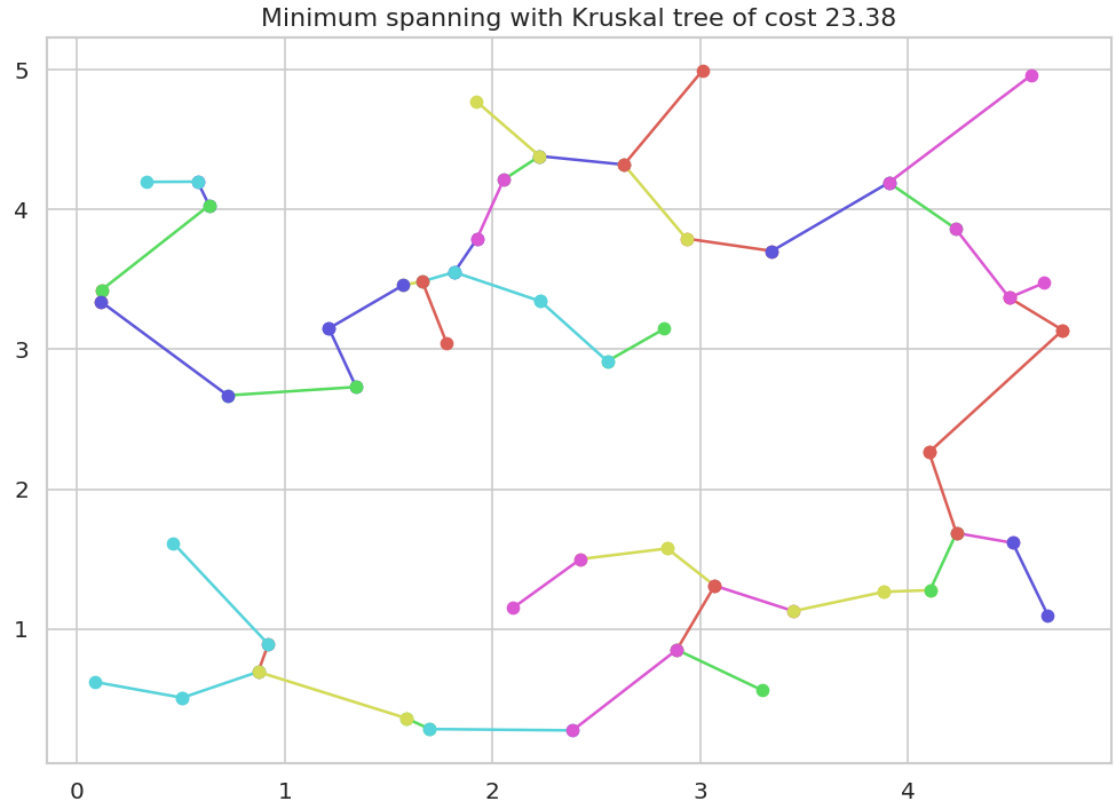


In [95]:
```
plt.figure()
val = 0
for u_idx, v_idx in min_span_tree_kruskal:
    val += weight[u_idx][v_idx]
    xu, yu = points[u_idx]
    xv, yv = points[v_idx]
    _ = plt.plot([xu, xv], [yu, yv], 'o-')
    # print("{} -- {}".format(points[u_idx], points[v_idx]))
print(val)
plt.title("Minimum spanning with Kruskal tree of cost {}".format(round(val, 2)))
plt.show()
```

Out[95]: <Figure size 1200x840 with 0 Axes>

23.37978298568072

Out[95]: Text(0.5, 1.0, 'Minimum spanning with Kruskal tree of cost 23.38')



In [96]:
```
plt.figure()
val = 0
for u_idx, v_idx in min_span_tree_prim:
    val += weight[u_idx][v_idx]
    xu, yu = points[u_idx]
    xv, yv = points[v_idx]
    _ = plt.plot([xu, xv], [yu, yv], 'o-')
    # print("{} -- {}".format(points[u_idx], points[v_idx]))
print(val)
plt.title("Minimum spanning with Kruskal tree of cost {}".format(round(val, 2)))
plt.show()
```

Out[96]: <Figure size 1200x840 with 0 Axes>

```
        ValueError                                Traceback (most recent call last)

        <ipython-input-96-da7c635e3b93> in <module>
          1 plt.figure()
          2 val = 0
    ----> 3 for u_idx, v_idx in min_span_tree_prim:
          4     val += weight[u_idx][v_idx]
          5     xu, yu = points[u_idx]


        ValueError: too many values to unpack (expected 2)


<Figure size 1200x840 with 0 Axes>
```

3.6 Autres

On en écrira plus tard !

3.7 Conclusion

C'est bon pour aujourd'hui !