# Intelligenza Artificiale (2020/21)

In questo elaborato ho implementato il **cutset conditioning**, un algoritmo che preso in input un grafo approssimabile ad un albero trova tale albero. Rende poi in output un nuovo grafo aciclico dopo aver effettuato un opportuno taglio al grafo iniziale. Successivamente si implementa anche il **map coloring,** il problema con vincoli.

Non avendo mai programmato in Python, ho dovuto prendere diversi spunti da siti e documentazioni online. Inizialmente ho cercato un modo per poter raffigurare i grafi e ho utilizzato i dizionari del Python dove ogni chiave (**dict.keys**) rappresenta un vertice e i valori di esse (**dict.values**) rappresentano a quali vertici sono connessi.

Dopo aver implementato le varie funzioni che mi sarebbero tornate utili al fine del progetto, ho cercato un modo per poter trovare e restituire in output tutti i cicli presenti in un grafo, tuttavia ho dovuto modificare e riadattare l'algoritmo che controllava solamente la presenza o meno di cicli (**isCyclic**). Questa è stata probabilmente la funzione più difficile da scrivere.

Dal momento che questa funzione (sopra citata) mi restituiva più percorsi come diversi output, ho raccolto tutti i cicli in una nuova funzione che crea una lista di liste dove ognuna di queste è un ciclo (**all_cycles**)

Dopo aver capito il modo per calcolare un certo percorso dato un vertice iniziale e uno finale (**find_path_cycle**), ho trovato come restituire un ciclo e ho effettuato questa operazione per ogni vertice del grafo.

A questo punto ho creato una definizione (**generate_graph**) che mi permettesse di creare casualmente grafi, dati in input il numero dei vertici. Dal numero dei vertici genero un numero di archi abbastanza alto da far sì che ci sia quasi sempre almeno un ciclo, ma tale che non superi un numero massimo di archi dato da: $\frac{(\text{n°vertici})(\text{n°vertici}-1)}{2}$

Per poter distinguere e capire meglio i vari procedimenti che il mio programma seguiva, ho cercato un modo per poter disegnare i grafi (**draw_graph**) in una nuova finestra. Questo è stato possibile grazie anche alla documentazione online sulla libreria *Python* ***NetworkX*** che permette di creare nuove finestre e rappresentare graficamente i vertici e gli archi del grafo.

Una volta trovati i cicli all'interno del grafo ho cercato di capire quale fosse il miglior vertice da tagliare considerando il numero di archi di ogni vertice del ciclo. Infatti, a seconda del numero di cicli, ho creato due funzioni (**vertex_max_edges** e **vertex_min_edges)** con le quali è possibile trovare il nodo da tagliare. Se il numero di cicli è minore di 1, calcolo il vertice con meno archi, mentre se sono presenti più cicli, taglio il vertice con più archi considerando solamente quelli appartenenti al ciclo. Per scegliere quali delle due funzioni applicare, è presente un'altra funzione (**cut_cycle**).

Successivamente, ho tagliato il grafo creandone quindi uno nuovo (**cut_graph**) che non contenesse cicli. La presenza o meno di cicli nel grafo tagliato è controllata da una funzione simile a quella precedente (**yet_cycles**). Se sono presenti, il grafo creato non è approssimabile ad un albero e quindi creo un nuovo grafo da tagliare utilizzando **generate_graph**

Infine ho implementato l'algoritmo **map coloring,** dove, tramite la ricorsione della funzione **solve_map_coloring,** è possibile sodisfare i vincoli del problema. Questa funzione chiama inoltre due altre funzioni: **add_color**, che colora i vertici assegnandogli un numero che va da 1 al numero di colori immessi in input e **checkRestriction,** che controlla se tutti i vincoli sono rispettati.

Qui sono presenti alcuni dati sperimentali che permettono di trovare il vertice giusto da tagliare dato in input un grafo approssimabile ad un albero, utilizzando **Jupyter Notebook**:

Prima di tutto è necessario importare *exercise* e *Graph,* dopodiché inizializzare il grafo che si desidera utilizzare *graph={}* e creare l'oggetto *g=Graph(graph).* Si ottiene il grafo in finestra scrivendo *%matplotlib qt* e chiamando poi la funzione *exercise.draw_graph(graph1,False)*. Per creare il grafo tagliato è necessario chiamare infine *g.isCyclic(False)*

# **Risultati**:

1. Esempio Australia :

   Qui si utilizza l'esempio classico dell'australia riportata nel libro di testo R&N 2010.

```
>>> import exercise
>>> import Graph
>>> %matplotlib qt
>>> australia = { "a" : ["b","f"], "b" : ["a","c","f"], "c" : ["b", "f","d"], "d" : ["c","e","f"], "e" : ["d","f"],
                "f" : ["a","b","c","d","e"], "g" : [] }
>>> g=Graph(australia)
>>> exercise.draw_graph(australia, False)
>>> g.isCyclic(False)
```

   Questo renderà come output il grafo primario in una finestra, successivamente quello tagliato in un'altra.

   Output:

```
The GRAPH is:
{'a': ['b', 'f'], 'b': ['a', 'c', 'f'], 'c': ['b', 'f', 'd'], 'd': ['c', 'e', 'f'], 'e': ['d', 'f'], 'f': ['a', 'b', 'c', 'd', 'e'], 'g': []}
--------------------------------------------------------------------------------------------------------------------------------
The vertex that must be cut is f
--------------------------------------------------------------------------------------------------------------------------------
The CUT GRAPH is:
{'a': ['b'], 'b': ['a', 'c'], 'c': ['b', 'd'], 'd': ['c', 'e'], 'e': ['d'], 'f': [], 'g': []}
```

2. Esempio graph1 :

>>> import exercise

[…]

>>> graph1 = { "a" : ["b","c","d"], "b" : ["a", "c","g"], "c" : ["a", "b", "d", "f"], "d" : ["a","c","e"],

"e" : ["d"], "f" : ["c"], "g" : ["b"] }

[…]

Output:

The GRAPH is:
{'a': ['b', 'c', 'd'], 'b': ['a', 'c', 'g'], 'c': ['a', 'b', 'd', 'f'], 'd': ['a', 'c', 'e'], 'e': ['d'], 'f': ['c'], 'g': ['b']}
---
The vertex that must be cut is a
---

The CUT GRAPH is:
{'a': [], 'b': ['c', 'g'], 'c': ['b', 'd', 'f'], 'd': ['c', 'e'], 'e': ['d'], 'f': ['c'], 'g': ['b']}

3. Esempio graph2 (Con la stessa sintassi descritta sopra):

>>>import…

[…]

>>> graph2 = {"a" : ["b", "f"],"b": ["a","f","c"], "c": ["b", "d", "f", "g"], "d": ["e", "c"], "e": ["d", "f"], "f": ["a", "b", "c", "e"], "g": ["c"]}

[…]

Output:

The GRAPH is:
{'a': ['b', 'f'], 'b': ['a', 'f', 'c'], 'c': ['b', 'd', 'f', 'g'], 'd': ['e', 'c'], 'e': ['d', 'f'], 'f': ['a', 'b', 'c', 'e'], 'g': ['c']}
---
The vertex that must be cut is f
---
The CUT GRAPH is:
{'a': ['b'], 'b': ['a', 'c'], 'c': ['b', 'd', 'g'], 'd': ['e', 'c'], 'e': ['d'], 'f': [], 'g': ['c']}

Il problema del map coloring viene applicato sempre a grafi casuali tramite la funzione **generate_graph(***numero vertici del grafo che vogliamo creare***,** *numero di colori che vogliamo utilizzare***)**. È necessario introdurre il numero dei vertici e il numero di colori. Se il numero dei vertici è troppo grande il tempo di risoluzione del programma è molto ampio.

Comunque è possibile applicare il map coloring a qualsiasi grafo attraverso la funzione **map_coloring(***grafo, numero colori, vertice da tagliare***)**. Dato che in questo progetto il map coloring si applica tramite il cutset conditioning è necessario dargli in input il vertice da tagliare.

### A. Per l'esempio australia

Dopo aver trovato l'output sopra descritto si può chiamare la funzione conoscendo il grafo "australia", il numero dei colori che possiamo mettere a piacimento e il vertice da tagliare:

**>>>** exercise.map_coloring (australia, 3, "f")

Output: {'a': 1, 'b': 2, 'c': 1, 'd': 2, 'e': 1, 'f': 3, 'g': 1}

### B. Per l'esempio graph1

E' sempre necessario conoscere il grafo e il vertice da tagliare:

>>> exercise.map_coloring (graph1, 3, "a")

Output: {'a': 1, 'b': 2, 'c': 3, 'd': 2, 'e': 1, 'f': 1, 'g': 1}

### C. Per l'esempio graph2

E' sempre necessario conoscere il grafo e il vertice da tagliare:

>>> exercise.map_coloring (graph2, 3, "f")

Output: {'a': 1, 'b': 2, 'c': 1, 'd': 2, 'e': 1, 'f': 3, 'g': 2}

## Osservazioni finali

Il programma funziona in un tempo limitato solo se con un numero di archi minori di 20. Per poter generare grafi casuali e applicare a scelta il cutset conditioning o il map coloring, decidendo il numero di vertici che il nostro grafo deve avere, possiamo utilizzare la funzione **exercise.generate_graph (***numero vertici, False o True***)**, dove se è False non applica map coloring ma solo cutset conditioning, se è True applica entrambi. Per esempio:

>>> exercise.generate_graph (7, *False*)

Output: Applica solo il cutset conditioning a un grafo casuale di 7 vertici

>>> exercise.generate_graph (7, *True*)

Output: Applica il map coloring tramite il cutset conditioning a un grafo casuale di 7 vertici.