Esame I.A. 10 settembre 2020

Soluzioni

1 Esercizio 1

Trovare limiti superiori e inferiori per la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} T\left(\frac{1}{10}n\right) + T\left(\frac{5}{6}n\right) + T\left(\frac{1}{16}n\right) + n & n > 1 \\ 1 & n \leq 1 \end{cases}$$

Soluzione È facile vedere che $T(n) = \Omega(n)$ a causa della parte non ricorsiva. Proviamo quindi a dimostrare che $T(n) = O(n)$:

- Caso base: per $n = 1, T(1) = 1 \leq c - b$, ovvero $c \geq b + 1$;
- Ipotesi induttiva: $T(n') \leq cn', \forall n' < n$;
- Passo induttivo:

$$\begin{aligned} T(n) &= T\left(\frac{1}{10}n\right) + T\left(\frac{5}{6}n\right) + T\left(\frac{1}{16}n\right) + n \\ &\leq \frac{1}{10}cn + \frac{5}{6}cn + \frac{1}{16}cn + n \\ &= \frac{24 + 200 + 15}{240}cn \\ &= \frac{239}{240}cn \\ &\leq cn \end{aligned}$$

L'ultima disequazione è vera per $c \geq 240$, abbiamo quindi dimostrato che $T(n) = O(n)$. Ne consegue che $T(n) = \Theta(n)$.

2 Esercizio 2

Siete stati appena assunti nel settore IT dell'università, con il compito di monitorare lo stato dei servizi offerti dall'ateneo. Purtroppo, c'è in corso una pandemia ed il perfetto funzionamento dei servizi informatici è fondamentale per consentire agli studenti di seguire i corsi da remoto.

Il servizio di classi virtuali e di videoconferenza è stato realizzato utilizzando un grande numero di macchine interconnesse tra di loro, al fine di migliorare l'accessibilità al servizio. Purtroppo, il software utilizzato per implementare tale servizio è tale per cui se una macchina non risulta raggiungibile sulla rete interna, l'intero servizio non funziona.

Realizzate un programma in python, il più efficiente possibile, che permetta, dato lo stato della rete interna, di determinare se una o più macchine non sono raggiungibili, a causa ad esempio di un collegamento interrotto. La rete può essere rappresentata come un grafo pesato indiretto, in cui i pesi sugli archi rappresentano la banda passante del collegamento. Un peso pari a zero indica un collegamento non funzionante.

Nel caso in cui il vostro programma identifichi la presenza di un problema, restituire un insieme di collegamenti che è possibile riparare per consentire al sistema di tornare a funzionare.

Soluzione Il sistema funziona se tutti i nodi sono raggiungibili, quindi se il grafo è tale per cui esiste un'unica componente connessa—un arco con peso zero è un arco "non percorribile" nella costruzione delle componenti connesse.

Pertanto, si può applicare l'algoritmo di ricerca delle componenti connesse su un grafo. Se il numero di componenti connesse è maggore di 1, allora c'è un partizionamento nella rete. In questo caso, occorre concentrarsi su tutti gli archi che sono stati incontrati che hanno peso zero—si noti che non viene richiesto l'insieme minimo di archi, bensì un generico insieme di archi.

Una possibile implementazione in python è:

```
class Graph:
    def __init__(self):
        self.nodes = {}

    def V(self):
        return self.nodes.keys()

    def __len__(self):
        return len(self.nodes)

    def adj(self, u):
        if u in self.nodes:
            return self.nodes[u]

    def weight(self, u, v):
        return self.nodes[u][v]

    def insertNode(self, u):
        if u not in self.nodes:
            self.nodes[u] = {}

    def insertEdge(self, u, v, w=0):
        self.insertNode(u)
        self.insertNode(v)
        self.nodes[u][v] = w
        self.nodes[v][u] = w

    def ccDFS(self, components, comp, u, add_edges):
        components[u] = comp
        for v in graph.adj(u):
            if components[v] == 0:
                if graph.weight(u, v) == 0:
```

```
                    add_edges.append([u, v])
                else:
                    graph.ccDFS(components, comp, v, add_edges)

    def isConnected(self):
        component = 0
        components = {}
        add_edges = []

        for u in graph.V():
            components[u] = 0

        for u in graph.V():
            if components[u] == 0:
                component = component + 1
                graph.ccDFS(components, component, u, add_edges)

        if component == 1:
            return True, None
        return False, add_edges
```

Questa implementazione può essere testata su due grafi, di cui uno solo connesso, come segue:

```
# Connected
graph = Graph()
for u, v, w in [('a', 'b', 3), ('a', 'd', 2), ('b', 'c', 5), ('c', 'd', 9), ('c'
                                        , 'e', 1), ('d', 'e', 1)]:
    graph.insertEdge(u, v, w)

connected, edges = graph.isConnected()
if not connected:
    print("Links to repair: ", edges)

# Not connected
graph = Graph()
for u, v, w in [('a', 'b', 0), ('a', 'd', 0), ('b', 'c', 5), ('c', 'd', 9), ('c'
                                        , 'e', 0), ('d', 'e', 0)]:
    graph.insertEdge(u, v, w)

connected, edges = graph.isConnected()
if not connected:
    print("Links to repair: ", edges)
```

Poiché il programma proposto si basa su una DFS, il suo costo sarà $O(V + E)$.

3 Quiz

Si consideri il seguente programma per individuare l'elemento minimo all'interno di una lista singolarmente concatenata. Quale dei seguenti confronti deve essere inserito all'interno del programma, per completarlo?

```
class Node:
    def __init__(self, v, n=None):
        self.val = v
        self.next = n
```

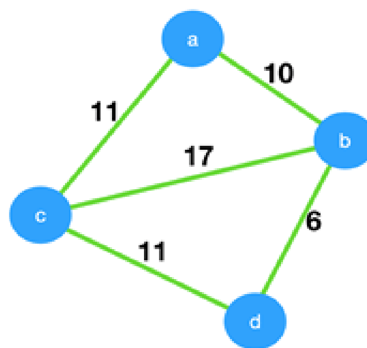
```python
def get_min(head):
    curr = head
    minimum = head.val

    while curr.next is not None:
        curr = curr.next
        if _____________:
            minimum = curr.val

    return minimum
```

- curr < minimum
- ✓ curr.val < minimum
- curr.val <= minimum
- curr.val > minimum

Un uomo vuole andare in diversi posti nel mondo. Li ha elencati tutti, ma ci sono alcuni posti che vuole visitare prima di altri posti. Quale algoritmo può usare per determinare il corretto ordine in cui visitare tutti i posti?

- Depth First Search
- Breadth First Search
- Algoritmo di Dijikstra
- ✓ Ordinamento topologico

Dato il seguente grafo, qual è il peso del minimo albero ricoprente se si utilizza l'algoritmo di Prim a partire dal nodo a?



- ✓ 27
- 39
- 38
- 28