INGEGNERIA DEGLI ALGORITMI

PROGETTO N°2

A cura di

Simone Grillo	0258129
Michele Granatiero	0253496
Ivan Palmieri	0259244

13 Gennaio 2018

INDICE:

1. TRACCIA

Algorithm 1 Checks whether a Graph has a cycle

```
procedure HASCYCLEUF(Graph G) → bool
    uf ← UnionFind
    for all unvisited (u, v) ∈ E(G) do
        if uf.find(u) == uf.find(v) then cycle detected
        else uf.union(u, v)
    return no cycle present
```

Dato un grafo G connesso, non orientato e non pesato, implementare due algoritmi per determinare se G ha almeno un ciclo:

• hasCycleUF(G) che usa la struttura dati UnionFind come descritto in Algorithm 1 e un iterator per scandire tutti gli edge del grafo

• hasCycleDFS(G) che usa l'approccio della visita in profondit`a (Depth-FirstSearch)

Dopo aver scelto l'implementazione di UnionFind (QuickFind, QuickUnion, ecc) che minimizza il tempo di esecuzione di hasCycleUF(G), confrontare il tempo di esecuzione dei due algoritmi al variare della dimensione del grafo di input. I dati degli esperimenti devono essere scritti su un file da un decorator di Python (applicato alle funzioni dei due algoritmi) con il formato n; t su ogni riga dove n `e il numero di archi del grafo e t il tempo di esecuzione dell'algoritmo. Inoltre implementare un algoritmo che genera grafi con o senza cicli, da usare nella fase di testing.

2. RICHIAMI TEORICI: L'ALGORITMO di Kruskal e la visita in profondità (DFS)

- L'algoritmo di Kruskal

L'algoritmo di Kruskal durante la sua esecuzione mantiene una foresta di alberi chiamata "Foresta di Alberi Blu", la quale sarà inizialmente costituita da $[n]$ alberi disgiunti, ciascuno con un singolo vertice, considerando gli archi del grafo G in ordine decrescente.

L'algoritmo effettua una verifica su ogni albero, controllando se ha gli estremi nello stesso albero blu; in tal caso lo colora di rosso, altrimenti lo colora di blu, restituendo a fine esecuzione l'albero formato dagli archi blu, cioè il minimo albero ricoprente.

La sua esecuzione può essere riassunta in questo modo:

"Se l'arco ha gli estremi nello stesso albero blu applica la regola del ciclo, colorandolo di rosso, altrimenti applica la regola del taglio, colorandolo di blu".

Il suo tempo di esecuzione è dato da:

- Verifica se l'arco ha gli stessi estremi nello stesso albero blu in tempo $O(n)$.
- La verifica è estesa e tutti gli $[m]$ archi ottenendo un tempo pari a $O(mn)$.

L'algoritmo può essere perfezionato grazie all'implementazione mediante Union-Find in questo modo, gli alberi blu possono essere rappresentati con degli insiemi disgiunti ai quali applicare le operazioni di Union-Find:

- Union permette di unire due alberi blu.
- Find verifica se gli estremi appartengono allo stesso albero blu.

Grazie a questa implementazione l'algoritmo di Kruskal viene eseguito nel caso peggiore in tempo $O(mlog(n))$.

- La visita in profondità (DFS)

La visita in profondità (DFS) esamina i vertici di un grafo generando un albero T chiamato "Albero DFS", procedendo in profondità sul grafo a partire dai vertici incontrati. Ogni volta che verrà esaminato un nuovo vertice esso sarà marcato come "esplorato". Il processo di esplorazione continua fin quando tutti i vertici del grafo non sono stati esaminati, per un tempo di esecuzione $O(m + n)$.

3. GLI ALGORITMI graphGeneratorNotCycle(n) e graphGeneratorCycle(n)

L'algoritmo graphGeneratorNotCycle(n) genera un grafo non orientato di dimensione n, dove n è il numero di nodi. Viene inizializzato un grafo $graph$ vuoto e due liste d'appoggio $dump\_node$ e $connected\_node$, anch'esse vuote. Successivamente vengono aggiunti, uno alla volta, n nodi al grafo ed a $dump\_node$.

A questo punto viene eseguito un ciclo $while$ fino a svuotare la lista $dump\_node$ nel seguente modo:

- Se la lista $connected\_node$ è vuota, prende un elemento casuale da $dump\_node$ e lo sposta in $connected\_node$.
- Se la lista $connected\_node$ non è vuota, crea un arco tra un elemento casuale di $connected\_node$, e un elemento casuale di $dump\_node$, quest'ultimo viene poi spostato da $dump\_node$ a $connected\_node$.

È facile dimostrare per induzione che l'algoritmo restituisce un grafo connesso senza cicli poiché gli archi che vengono creati avvengono sempre tra un nodo che appartiene a un grafo connesso aciclico e un nodo che non è connesso al grafo.

L'algoritmo graphGeneratorCycle(n) genera tramite graphGeneratorNotCycle(n) un grafo aciclico non orientato di dimensione n. A questo punto viene inserito almeno un ciclo creando archi tra elementi successivi tramite un ciclo for e creando un arco tra l'ultimo elemento e il primo.

4. GLI ALGORITMI hasCycleUF(G) e hasCycleDFS(G)

- hasCycleUF(G)

L'algoritmo hasCycleUF(G), utilizzando una struttura dati Union-Find, verifica se, dato un grafo come argomento, esso ha un ciclo, restituendo un valore booleano.

Utilizzando un'implementazione simile all'algoritmo di Kruskal, viene inizializzato un insieme UF di tipo Union-Find a cui vengono aggiunti tutti i nodi. Successivamente, tramite un iteratore*, vengono scanditi tutti gli archi e, per ogni arco, viene confrontata la testa e la coda dell'arco stesso: se la radice dei coincide significa che esiste un ciclo, quindi viene restituito il valore "True", altrimenti, se non è mai verificata questa condizione, verrà restituito il valore "False".

*L'iteratore utilizzato è quello generato internamente tramite la struttura $for \dots in < collection >$

- hasCycleDFS(G)

L'algoritmo hasCycleDFS(G), sfruttando una modifica del metodo per la creazione di un albero DFS ($graph.dfsCycle()$), verifica se dato come argomento un grafo, esso ha un ciclo restituendo un valore booleano.

Viene scelto un nodo casuale da cui far partire una visita DFS: vengono marcati tutti i vertici come "inesplorati", successivamente viene marcato il vertice di partenza come "aperto" e si inizializzano l'albero DFS (formato dal solo vertice di partenza) e una pila vuota.

Si inserisce ($push$) nella pila il vertice di partenza, a questo punto viene eseguito un ciclo $while$ che procede nel modo seguente fino a quando la pila non risulta vuota:

- Viene estratto l'ultimo vertice u dalla pila (pop) e segnato come "esplorato"
- Vengono osservati tutti i nodi adiacenti ad u, se almeno uno di essi è marcato come "esplorato" significa che esiste un ciclo, quindi viene restituito il valore "True", altrimenti, se non è mai verificata questa condizione, verrà restituito il valore "False"

5 VALUTAZIONE RISULTATI SPERIMENTALI

Come è ben visibile dal grafico dell'andamento dei tempi d'esecuzione, vediamo come l'algoritmo hasCycleDFS(G) è molto più veloce dell'algoritmo hasCycleUF(G)