Es 2:

Per ricercare i *bridge* all’interno di un grafo non diretto è stato utilizzato il seguente algoritmo:

Dato un grafo G, e considerando un generico *spanning tree* di G (ad esempio quello restituito dalla DFS\_Completeper grafi connessi), tutti gli archi non inclusi nello *spanning tree* non sono archi *bridge*, inoltre aggiungendo uno di essi si ha la formazione di uno o più cicli.

Perciò l’analisi degli archi *bridge* ricade solo su quelli appartenenti allo *spanning tree*.

Preso un generico arco (u,v) dell’albero, quest’ultimo è un *bridge* se il nodo verso cui è diretto, nodo *V*, non ha archi incidenti che sono di tipo *back* e se i discendenti (ai livelli inferiori) del nodo *V* non hanno archi *back* connessi a nodi antenati: cioè se l’arco non appartiene a cicli e quindi tutti i percorsi per raggiungere *V* percorrono l’arco (u,v).

Per ottenere uno *spanning tree*, la funzione bridge(G) invoca la:

binary\_tree\_from\_dfs(G)

Tale funzione esegue una DFS\_Complete sul grafo *G*, il dizionario restituito dalla DFS è poi utilizzato da due funzioni aggiuntive per trasformare il risultato in un LinkedBinaryTree:

make\_tree\_dict(discovered, node\_list, tree\_dict)

from\_tree\_dict\_to\_bin\_tree(p, tree, bt)

La complessità temporale di queste due funzioni è pari a *O (n)* dovendo solo iterare dizionari di lunghezza *n*.

La complessità della binary\_tree\_from\_dfs invece, dovendo eseguire una DFS, è pari a *O (n+m).*

Una volta ottenuto l’albero della DFS come LinkedBinaryTree, la funzione bridge(G) controlla gli archi appartenenti all’albero.

Per verificare tutti gli archi, vi è la necessità di calcolare quattro parametri per ogni nodo all’interno dell’albero:

* discovery number (N)
* descendants number (DN)
* lower reachable (LR)
* higher reachable (HR)

Il *discovery number* è assegnato progressivamente durante la visita in postorder (o preorder, in tal caso la condizione sarà diversa) indicando il “tempo” in cui il nodo è stato scoperto durante la visita in postorder, il *descendants number* indica il numero di discendenti del nodo più il nodo stesso, *lower* e *higher reachable* indicano il nodo raggiungibile con il *discovery number* rispettivamente più basso e più alto utilizzando oltre agli archi interni all’albero anche gli archi *back*.

**Un arco *u*->*v* sarà *bridge* se e solo se per il nodo *v* sarà possibile raggiungere soltanto i nodi radicati nel sottoalbero con radice *v*.**

Perciò una volta calcolati questi valori, preso un nodo *v*, se quest’ultimo avrà:

HR <= N AND LR > N – DN

Allora l’arco diretto verso di lui nell’albero sarà un arco *bridge*.

Tale condizione corrisponde a dire che sia HR sia LR sono compresi all’interno del sottoalbero con radice *v* ({n-dn+1, n-dn+2, …, n}).

E’ facile verificare che data la visita in postorder tutti i nodi nel sottoalbero radicato in *v* avranno *discovery number* minore, per cui se HR è maggiore di N il nodo può raggiungere nodi a livelli più alti e quindi l’arco soprastante non è un arco *bridge*.

Allo stesso modo data la visita in postorder il numero più piccolo raggiungibile da *v* nel sottoalbero radicato in *v* è pari proprio a N-DN, per cui se LR è minore di N-DN allora da quel nodo si possono raggiungere nodi non radicati nel sottoalbero originato da *v* e quindi l’arco soprastante non è un arco *bridge*.

La complessità finale dell’algoritmo è pari a *O (n+m).*