Grapha-Nui Problem

Nome: MASSIMILIANO
Cognome: GIORDANO ORSINI
Matricola: 0124002214

Descrizione del problema:

Dietro il testo romanzato del quesito, in cui appaiono evidenti riferimenti ad una particolare struttura dati, si cela un problema sui grafi. Infatti, come possiamo notare, le isole, a cui viene associato un indice PIL, e i ponti, che compongono l'arcipelago di Grapha-Nui, rappresentano rispettivamente i vertici, dotati di un peso, e gli archi di un grafo. Tali vertici sono quindi pesati; i ponti sono percorribili in due direzioni, pertanto il grafo risulta "non orientato". Inoltre, nessuna specifica esclude grafi particolari, come ad esempio grafi ciclici.

Il problema di Grapha-Nui, quindi, è quello di determinare il percorso che massimizza gli introiti del governo che non attraversa lo stesso ponte più volte, in altre parole, il cammino di costo massimo in un grafo non orientato che ha il vincolo di non poter attraversare lo stesso arco più di una volta, data una particolare sorgente.

Descrizione della struttura dati:

Un grafo è una struttura dati, spesso utilizzata in informatica, formata da due insiemi: un insieme V di elementi detti nodi (o vertici) ed un insieme E di archi. Viene comunemente rappresentato attraverso la notazione $G = (V, E)$.

Se ogni arco, appartenente all'insieme E, è una coppia ordinata (u, v) di nodi, in cui u e v sono elementi di V, allora il grafo viene detto "orientato" o diretto, pertanto $(u, v) \neq (v, u)$. Invece, se ogni arco, appartenente all'insieme E, è una coppia non ordinata (u, v) di nodi, in cui u e v sono elementi di V, allora il grafo viene detto "non orientato", pertanto $(u, v) = (v, u)$. Se u e v sono elementi di V collegati da un arco, si dice che u è adiacente a v.

Per cammino o percorso di un grafo s'intende la sequenza di nodi $v_0, v_1, \dots, v_{n-1}$ tali che le coppie $(v_0, v_1), (v_1, v_2), \dots, (v_{n-2}, v_{n-1})$ sono archi. Un cammino semplice è un cammino in cui tutti i nodi, eccetto eventualmente il primo e l'ultimo, sono distinti. La lunghezza del percorso è il numero di nodi presenti al suo interno oppure il numero di archi attraversati. Per ciclo di un grafo s'intende un sotto-insieme di connessi in una catena chiusa.

La rappresentazione di un grafo G può essere realizzata mediante:

1. matrice di adiacenza A di dimensione $|V| \times |V|$: la posizione a_{ij} indica la presenza di un arco tra l'elemento i e l'elemento j
2. lista di adiacenza: ogni nodo è descritto da una lista contenente gli elementi ad esso adiacenti.

La struttura dati implementata prevede una rappresentazione mediante liste di adiacenza. Infatti, ogni nodo contiene la sua lista di adiacenza, l'informazione e un riferimento al padre. Tra le operazioni classiche della struttura dati, è presente la visita, la quale avviene "in ampiezza" o "in profondità".

Il grafo-oggetto del problema è quindi:

1. non orientato
2. vertex-weighted, ossia il peso è collocato sui vertici del grafo
3. possibilmente ciclico, come riportato nell'esempio del quesito.

Formato dei dati input/output:

L'utente può interagire, da tastiera, digitando un numero intero positivo, corrispondente all'operazione che vuole eseguire. Nel caso in cui l'utente dovesse inserire una cifra non valida, il programma fa ripetere l'inserimento della cifra per la scelta tra le opzioni disponibili fino a quando questa non risulta essere corretta.

All'avvio del file eseguibile, l'utente è chiamato ad inserire la stringa corrispondente al nome del file di testo in cui sono memorizzati i dati. Il primo rigo contiene due interi separati da uno spazio: il numero N delle isole (numerate da 0 ad $N-1$) ed il numero P dei ponti. I successivi N righi contengono i PIL di ciascuna isola ed i successivi P righi contengono ciascuno due numeri I_1, I_2 per indicare la coppia di isole collegate dal ponte. Il numero N delle isole è compreso tra 2 e 1000 mentre il numero P dei ponti tra 1 e 10000.

L'output dell'algoritmo è rappresentato dalla stampa a video del percorso di costo massimo, ossia la sequenza ordinata dei nodi attraversati, rappresentati da un identificatore univoco, e il relativo costo.

Descrizione dell'algoritmo:

L'algoritmo è basato su una visita in profondità a cui sono apportate delle varianti. La logica dietro la visita in profondità (di un albero) è partire da un nodo scelto arbitrariamente come radice, visitare il nodo e spostarsi al prossimo nodo adiacente non visitato fin quando è possibile, per poi ritornare al passo precedente e ripetere l'operazione per i nodi adiacenti non ancora visitati e quindi raggiungibili dal nodo sorgente.

Per semplicità, viene assegnato un colore ad un particolare stato del nodo: bianco se questo non è ancora stato visitato, grigio se la sua visita è in corso e nero se è stato completamente visitato. Nello specifico caso, sono state apportate delle modifiche agli stati associati ai nodi.

L'algoritmo proposto prevede un ulteriore stato, e quindi colore, relativo al nodo. Infatti, poiché lavoriamo con un grafo possibilmente ciclico, occorre ricordare quali nodi vi appartengono. Il colore associato a tale stato del nodo è il grigio scuro.

Sia v il nodo che stiamo visitando, ad ogni iterazione a partire dalla radice, l'algoritmo esegue un controllo sul colore di v, viene sequenzialmente memorizzato il nodo v ed assegnato il padre del nodo; esistono quindi quattro possibili casi:

1. se v è nero, la visita termina.
2. se v è grigio, allora v fa parte di un ciclo quindi dev'essere colorato di grigio scuro, di conseguenza, i nodi che vi fanno parte vengono colorati di tale colore e gli viene assegnato il costo accumulato, in quanto tali nodi possono essere raggiunti percorrendo il ciclo almeno una volta, massimizzando così il costo per ognuno. Viene dunque terminata la visita.
3. se v è grigio scuro, allora v dev'essere colorato di nero poiché, se così non fosse, ciò permetterebbe di percorrere cammini che attraversano gli stessi archi più di una volta.
4. se v è bianco, allora v deve essere colorato di grigio e calcolato il costo accumulato fino a v.

Negli ultimi due casi, viene esaminata la lista di adiacenza di v. Sia u un nodo adiacente a v:

a) se u risulta essere il padre, allora u viene ignorato poiché significherebbe ripercorrere l'arco una seconda volta
b) se u è nero e v è grigio, allora u viene ignorato in quanto gli archi collegati da/a u sono già stati attraversati almeno una volta.

Se u non ricade in questi due sotto-casi, viene chiamata ricorsivamente la procedura su u.

Una volta arrivati alle foglie è necessario verificare che il percorso fatto sia quello di costo massimo, pertanto si verifica se il costo accumulato risulta essere maggiore del costo massimo memorizzato fino a quel

momento. In caso di esito positivo, viene aggiornato il costo massimo e memorizzato tale percorso. A parità di costo, viene memorizzato come percorso di costo massimo il percorso la cui lunghezza risulta essere maggiore. Questa operazione viene effettuata poiché ai nodi di un ciclo viene assegnato il suo costo totale, pertanto potrebbe verificarsi che un percorso contenente tali nodi non sia effettivamente un percorso valido, viene quindi correttamente sostituito. Inoltre, in quanto foglie, queste vengono ricolorate di bianco per permettere di analizzare ulteriori percorsi che giungono a tali foglie.

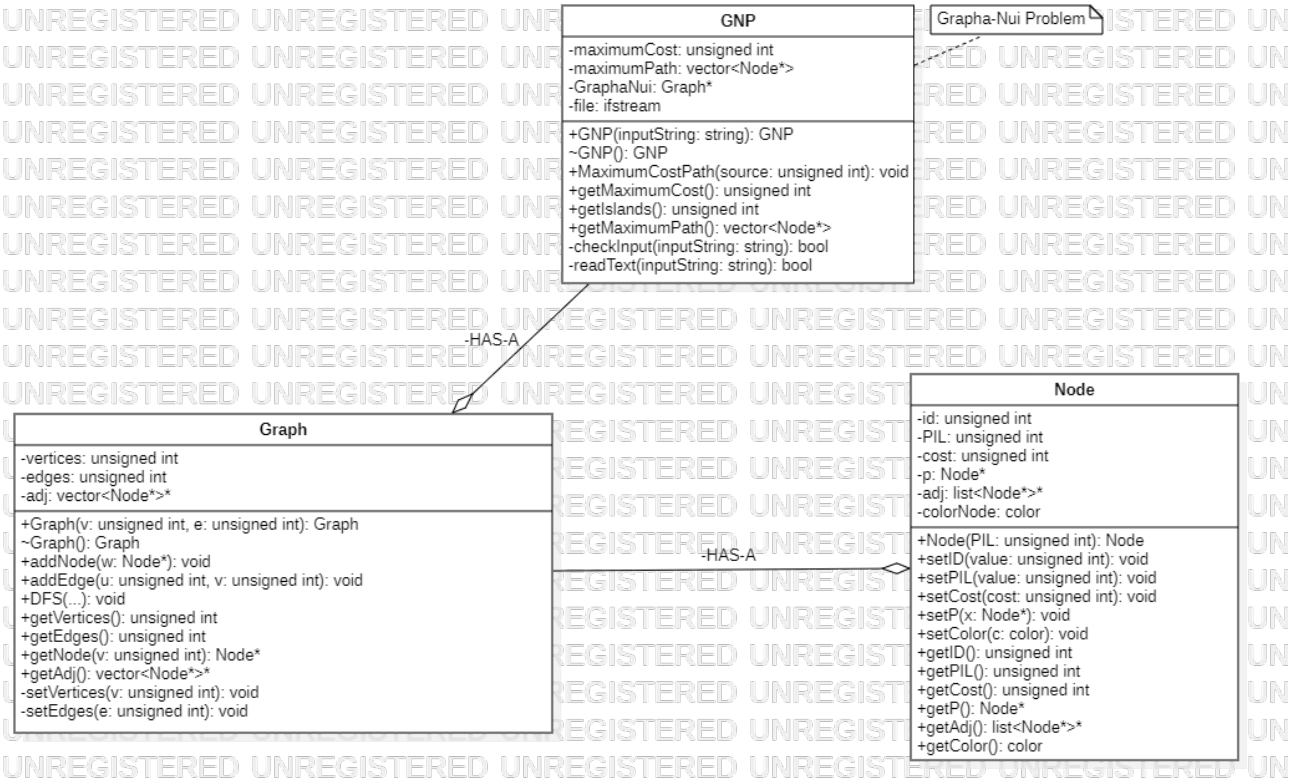
Prima di fare il passo di "backtrack", viene scartato il nodo la cui visita è appena terminata.

```
procedura DFS(G, u, parent, cost, maximumCost, maximumPath, A)

      if color[u] == BLACK              //case 1
            return

      if color[u] == LIGHTGRAY          //case 2
            cur ← parent
            color[cur] ← DARKGRAY
            cost[u] ← cost
            while cur ≠ u
                  cur ← π[cur]
                  color[cur] ← DARKGRAY
                  cost[cur] ← cost
            return

      if color[u] == DARKGRAY           //case 3
            color[u] ← BLACK

      if color[u] == WHITE              //case 4
            color[u] ← LIGHTGRAY
            cost[u] ← cost + PIL[u]

      π[u] ← parent
      push(A, u)

      for each v ∈ Adj[u]
            if v == π[u] continue
            if color[u] == BLACK and color[v] == LIGHTGRAY continue
            DFS(G, v, u, cost[u], maximumCost, maximumPath, A)

      if cost[u] > maximumCost
            maximumCost ← cost[u]
            maximumPath ← A
      else if cost[u] == maximumCost and A.size > maximumPath.size
            maximumPath ← A

      if color[u] == LIGHTGRAY
            color[u] ← WHITE
      pop(A)
```

Al termine dell'algoritmo, ogni nodo conterrà il costo massimo del percorso che parte dalla medesima sorgente e ha tale nodo come foglia, inoltre i nodi appartenenti a cicli saranno colorati di nero. Pertanto quindi vengono mostrati il percorso di costo massimo, il relativo costo e la relativa sorgente.

```
procedura MaximumCostPath(source)
      A = ∅
      maximumPath = ∅
      cost ← 0
      maximumCost ← 0
      DFS(G, source, NIL, cost, maximumCost, maximumPath, A)

      print source
      print maximumCost
      print maximumPath
```

Class Diagram:



La relazione di aggregazione di tipo “has-a” tra la classe GNP e la classe Graph deriva dal fatto che la prima utilizza un’istanza della seconda, pertanto è opportuno scegliere una relazione di questo tipo. Per lo stesso motivo, la classe Graph ha una relazione di aggregazione “has-a” con la classe Node.

Studio della complessità:

L’algoritmo si basa su una variante della visita in profondità DFS, in quanto opera solo su una sorgente e deve memorizzare il cammino di costo massimo. Ipotizzando che, la memorizzazione del cammino sia una sequenza di assegnazioni e che questo contenga al più i V nodi del grafo, possiamo attribuire a questa un costo pari a $\mathrm{O}(V)$.

Il numero di volte che viene eseguita questa operazione è situazionale e dipende dal numero delle foglie del grafo. Nel caso migliore, ossia quello in cui il percorso sia esattamente quello di costo massimo alla prima foglia raggiunta, il costo di questa operazione è quindi $\mathrm{O}(V)$. Nel caso peggiore, ossia quello in cui il percorso di costo massimo sia aggiornato foglia dopo foglia, date k foglie, l’operazione viene effettuata k volte. Consideriamo grafi generici ed assumiamo k pari ad $\mathrm{O}(V)$, il costo è pari ad $\mathrm{O}(V^2)$.

Le operazioni effettuate sullo “stack”, aggiornato ad ogni chiamata, hanno un costo medio pari a $\mathrm{O}(1)$ per il metodo dell’aggregazione.

Se il grafo è connesso, si può subito indicare come $\Omega(V + E)$ il limite inferiore, in quanto non viene generata una foresta DF ma solamente un albero DF poiché viene indicata una sorgente unica su cui applicare l'algoritmo. Se il grafo risulta essere non connesso, l'algoritmo lavora ugualmente.

E' opportuno però fare una distinzione sulla struttura del grafo, pertanto si analizza la complessità dell'algoritmo applicato su un grafo aciclico e non.

Sia G un grafo aciclico, connesso e non orientato, l'algoritmo esplora la lista di adiacenza di ogni nodo, per cui questo lavora soltanto se un vertice è bianco e quindi gli archi risultano essere tutti archi d'albero. Questa operazione viene eseguita $|Adj[v]|$ volte. Poiché

$$\sum_{v \in V} |Adj[v]| = \Theta(E)$$

il costo totale dell'algoritmo è pari a $O(V^2 + E)$, dove V è l'insieme di vertici ed E è l'insieme degli archi del grafo.

Sia G un grafo ciclico, connesso e non orientato, l'algoritmo esplora la lista di adiacenza almeno una volta per ogni nodo. Poiché i nodi che appartengono a cicli vengono ispezionati più di una volta così come le foglie, non si può definire $O(E)$ come limite superiore ma è opportuno indicare $O(E^2)$. Pertanto il costo dell'algoritmo è pari ad $O(V^2 + E^2)$, sebbene questo non sia un limite stretto.

La complessità di spazio è pari a $\Theta(V + E) + O(V)$, in quanto viene utilizzata un'ulteriore area di memoria per memorizzare il percorso.

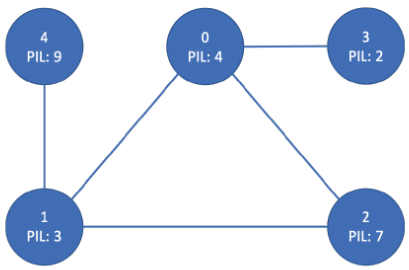
Test e risultati:

Test #1:

L'obiettivo del test è confrontare i risultati ottenuti dall'algoritmo applicato sul grafo-esempio presente all'interno della traccia e quelli presentati dalla stessa. Pertanto, all'interno del file di testo, sono presenti i medesimi valori di input. Di seguito viene riproposto l'esempio.

Esempio

input.txt	Output (partendo dall'isola 1)
5 5 4 3 7 2 9 0 1 0 2 0 3 1 2 1 4	**23** 1->2->0->1->4



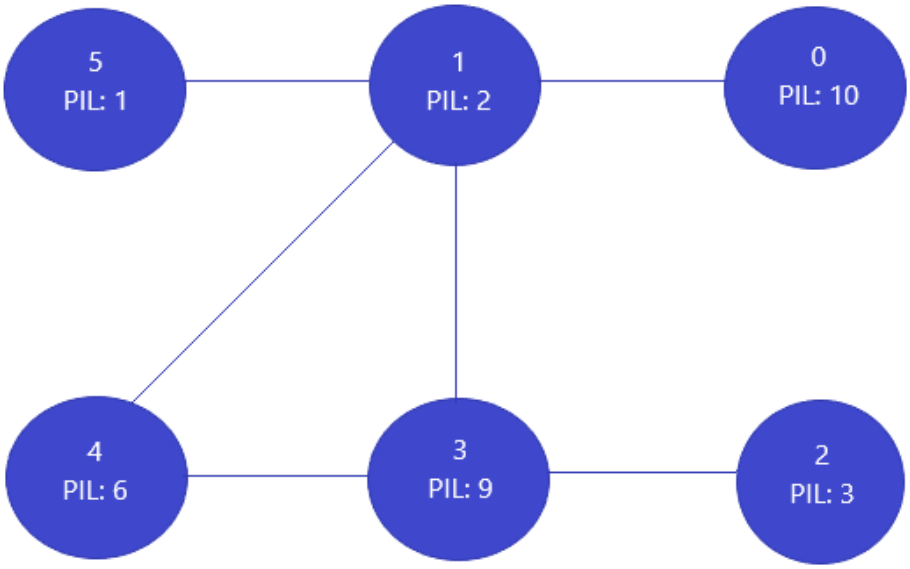
All’avvio, l’utente inserisce la stringa corrispondente al nome del file di testo; l’algoritmo mostra a video l’esito dell’apertura del file e dell’inserimento dei vertici e degli archi, fornendo le informazioni analitiche del grafo.

Successivamente, è richiesto l’inserimento di un nodo sorgente su cui applicare l’algoritmo, pertanto l’utente inserisce un numero intero positivo corrispondente all’identificatore nodo che si vuole indicare come sorgente.

L’algoritmo viene eseguito e al termine vengono mostrati i risultati, che come si può notare, corrispondono esattamente a quelli presenti nell’esempio all’interno della traccia.

```
----------GRAPHA-NUI PROBLEM-------------
Insert TextFile Name:
input.txt
FILE: input.txt
Status: Successfully opened!

V: 5    E: 5

Node 0: added! /PIL: 4
Node 1: added! /PIL: 3
Node 2: added! /PIL: 7
Node 3: added! /PIL: 2
Node 4: added! /PIL: 9

Edge (0,1): added!
Edge (0,2): added!
Edge (0,3): added!
Edge (1,2): added!
Edge (1,4): added!

Select a source from Node 0 to Node 4
1
Source: 1
MaximumCost: 23
1 -> 2 -> 0 -> 1 -> 4 ->
Premere un tasto per continuare . . .
```

Test #2:

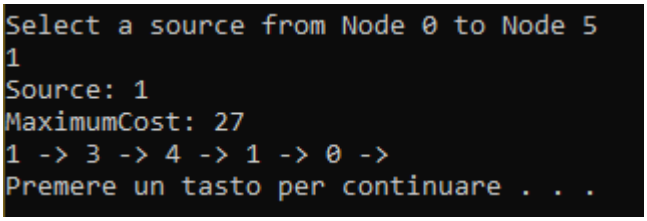


L’obiettivo del test è applicare l’algoritmo sul grafo riportato sopra e confrontare i risultati con le previsioni antecedentemente all’esecuzione. In particolare, si vuole applicare l’algoritmo specificando come sorgente il nodo 1 e successivamente il nodo 0.

Nel primo caso, il percorso di costo massimo ha costo pari a 27 ed è determinato dalla sequenza:

$$1 \rightarrow 4 \rightarrow 3 \rightarrow 1 \rightarrow 0$$

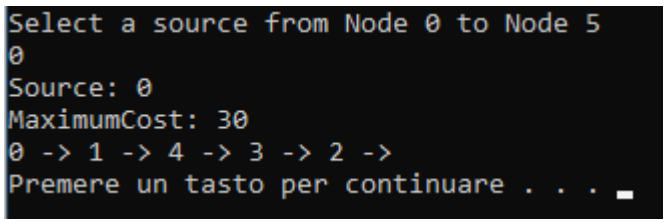
Dopo aver inserito i dati relativi al grafo e specificato il nodo 1 come sorgente, l'output mostrato dell'algoritmo è riportato qui a destra.



Sebbene questo risultato non sia esattamente identico alla previsione fatta inizialmente, è considerabile comunque corretto in quanto, per grafi ciclici, risulta determinante l'ordine di inserimento degli archi all'interno della struttura dati.

Nel secondo caso, quello in cui la sorgente sia il nodo 0, il percorso di costo massimo ha costo pari a 30 ed è determinato dalla sequenza:

$$0 \rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow 2$$



Dopo aver inserito i dati relativi al grafo e specificato il nodo 0 come sorgente, l'output mostrato dell'algoritmo è riportato qui a sinistra.

Anche in questo caso, il risultato ottenuto dall'algoritmo è coerente con la previsione fatta inizialmente.