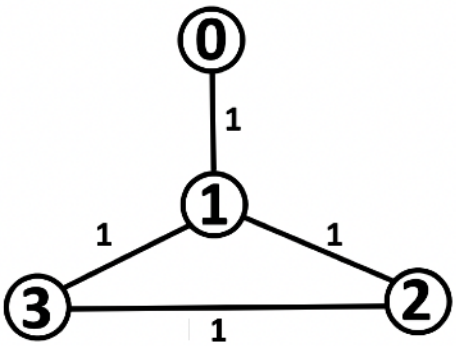
Documento a cura di Simone Remoli.

https://github.com/SimoneRemoli/Dijkstra-s-algorithm_New.git

Algoritmo di Dijkstra

Redigo due annotazioni riguardanti l'impiego dell'algoritmo di Dijkstra. Tale algoritmo trae gran beneficio dai fondamenti dell'informatica teorica, in special modo dal concetto di pila (o stack), essenziale nella ricostruzione dei cammini minimi. Sarà *deliberatamente omesso* il **cuore operativo dell'algoritmo**, che si occupa di determinare i soli costi dei percorsi minimi, con particolare riguardo alla computazione del vettore dei costi. È altresì sottinteso che il vettore dei nodi "conosciuti" sia **integralmente impostato a vero**, poiché ci poniamo in un contesto in cui l'algoritmo ha già ultimato le sue iterazioni relative alla determinazione dei costi di percorrenza.

Dunque, esaminando il caso di un grafo composto da soli quattro nodi, la disposizione attuale dei due array si presenta come segue:



Cost = [0, 1, 2, 2]

Know = [1, 1, 1, 1]

Se si apre il codice implementato è possibile notare esattamente questo output.

Fine premessa.

```
125             if(cost[adj_temp] > cost[nodo_partenza] + matriceAdiacenza[nodo_partenza][adj_temp])
126             {
127                 cost[adj_temp] = cost[nodo_partenza] + matriceAdiacenza[nodo_partenza][adj_temp];
                    precedente[adj_temp] = nodo_partenza;
129             }
```

Poniamo particolare attenzione ad una istruzione all'interno del codice, ossia l'istruzione che assegna all'indice del nodo adiacente del vettore precedente, il nodo di partenza.

Inizialmente, il nodo di partenza 0 ha come nodo adiacente il nodo 1, e quindi il nodo 1 registra come nodo precedente il nodo 0.
Successivamente, il nodo 1 viene impostato come nuovo nodo di partenza, e di conseguenza tutti i suoi nodi adiacenti, ovvero i nodi 2 e 3, avranno come nodo precedente il nodo 1. Il nodo 0, pur essendo adiacente al nodo 1, è già stato contrassegnato come "conosciuto". È importante notare che l'istruzione mostrata nella figura viene eseguita soltanto se il nodo di partenza non è già stato contrassegnato come "conosciuto".
Pertanto, ci ritroveremo con un array dei nodi precedenti configurato nel modo seguente:

Prec = [-1, 0, 1, 1]

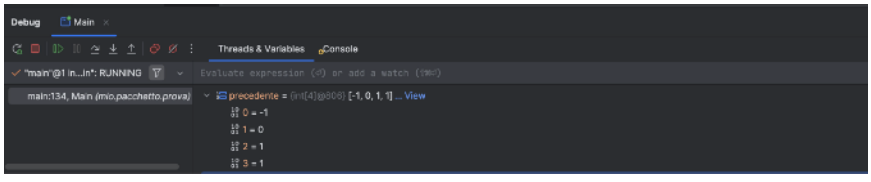
Il precedente di 0 non è nessuno.

Il precedente di 1 è 0.

Il precedente di 2 è 1.

Il precedente di 3 è 1.

Notiamo che noi vorremmo ricostruire un cammino per tutti questi nodi.
Ricordiamo che il costo minimo già lo abbiamo.
Ma quali sono i cammini che permettono di mostrare graficamente come ricostruire quel costo minimo?



```
140         System.out.println("Path.");
141         for (int i = 0; i < matriceAdiacenza.length; i++)   matriceAdiacenza: int[4][]@802
142         {
143             System.out.print("Nodo " + i + ": Costo = " + cost[i] + ", Percorso = ");
144             a.stampaPercorso(i, precedente);
145             System.out.println();
146         }
147     }
148 }
```

(Immagine che mostra il contenuto di prec da processo Debugging).

Dunque, come si può ben notare, ci apprestiamo a compiere l'operazione cardinale di tutto l'algoritmo, in cui si itererà un numero di volte pari al numero dei nodi del grafo, ovvero quattro volte. Verrà eseguita la stampa dell'array dei costi (del quale non ci dilungheremo) e si visualizzerà il percorso necessario per giungere a ciascun nodo; tale operazione sarà realizzata mediante la funzione "stampaPercorso(i:int, precedente:array)".

La funzione è così siffatta:

```
64      public void stampaPercorso(int nodo, int[] precedente) {  1 usage  new *
65          Stack<Integer> percorso = new Stack<>();
66          while (nodo != -1) {
67              percorso.push(nodo);
68              nodo = precedente[nodo];
69          }
70          while (!percorso.isEmpty()) {
71              System.out.print(percorso.pop() + " ");
72          }
73      }
```

Ora spieghiamo come è composta.

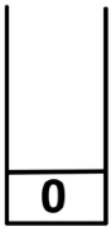
Internamente c'è dichiarata uno stack, pronto ad immagazzinare informazioni intere, in questo caso i nodi.

1. PASSO.

Nodo = 0;

A ritroso voglio trovare il percorso breve.

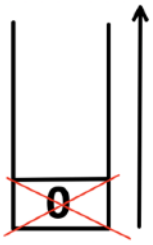
Push 0 sullo stack.



Nodo = prec[0], ossia Nodo prende il valore del precedente di 0, ossia nessuno, quindi **Nodo = -1;**

Ora viene nuovamente eseguito il primo ciclo while ma questa volta è pari a -1 il nodo, quindi si esce dal primo ciclo while e si procede verso il secondo ciclo while.

A questo punto lo stack è non vuoto quindi si effettua un pop da esso e ciò che viene estratto rappresenta proprio il cammino:



Per il nodo = 0, il cammino minimo è quello passante per 0.

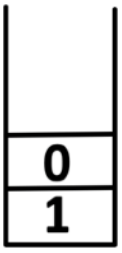
2. PASSO.

Nodo = 1;

Nodo = prec[1] = 0.

0 != -1.

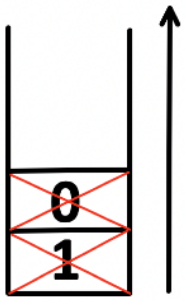
Push 0 on stack.



Nodo = prec[0] = -1;

-1 = -1, 1 ciclo while stop.

Svuoto lo stack.



Per il nodo = 1, il cammino minimo è quello passante per 0,1.

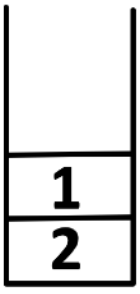
3. PASSO.

Nodo = 2;

Nodo = prec[2] = 1.

1 != -1.

Push 1 on stack.



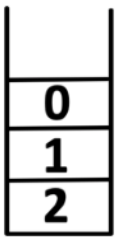
Nodo = prec[1] = 0;

0 != -1.

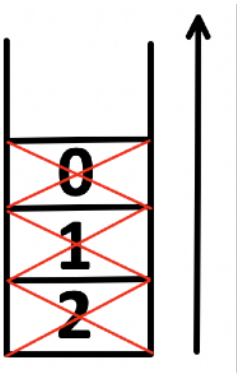
Push 0 on stack.

Nodo = prec[0] = -1;

-1 = -1, 1 ciclo while stop.



Svuoto lo stack.



Per il nodo = 1, il cammino minimo è quello passante per 0,1,2.

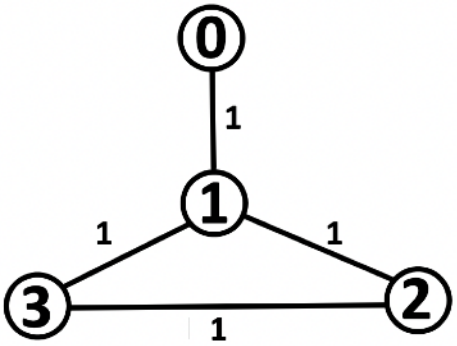
Il terzo passaggio opera in maniera del tutto analoga. L'algoritmo di Dijkstra funziona mediante una ricorsione applicata su un PDA (Push Down Automata). Si rammenti che **l'impiego dello stack** serve *unicamente per visualizzare i percorsi minimi* dal nodo di partenza (impostato, per impostazione predefinita, a 0) verso tutti gli altri nodi. Questo approccio non si preoccupa della gestione del vettore dei costi.

La gestione del vettore dei costi in questa trattazione sarà del tutto scontata essendo tra virgolette "la parte facile" dell'algoritmo.

Concludo affermando che, avendo a disposizione un vettore dei precedenti, per ogni nodo (da 0 a N, dove N è il numero dei nodi), registro nello stack, tramite operazioni di *push*, la catena dei precedenti. Iniziando dal nodo desiderato, inserisco progressivamente ogni suo predecessore fino a raggiungere il valore -1, indicante l'assenza di ulteriori predecessori. Successivamente, svuotando lo stack, ottengo in ordine il percorso desiderato.

Risultato finale.

```
Nodo 0: Costo = 0, Percorso = 0
Nodo 1: Costo = 1, Percorso = 0 1
Nodo 2: Costo = 2, Percorso = 0 1 2
Nodo 3: Costo = 2, Percorso = 0 1 3
```



Cost = [0, 1, 2, 2]

Know = [1, 1, 1, 1]

Prec = [-1, 0, 1, 1]

Queste informazioni e queste tecniche algoritmiche sono state pensate ispirandosi a 迪克斯特拉算法 - https://github.com/Lijiale96.
Lo ringrazio molto.