Relazione per il progetto del Laboratorio di Algoritmi e Stutture Dati
A.A. 2013-2014
Progetto [*Avanzato*]
Competizione [*Si*]

Alessio Addimando, 0000693233
Edoardo Antonini, 0000691494
Alberto Giunta, 0000691428

1. Definizione del problema

Dato un grafo orientato e pesato(con pesi strettamente positivi), calcolare tutti i cammini minimi tra tutte le coppie di vertici nel grafo. Stampare i cammini minimi.

2. Scelte progettuali

L'algoritmo che abbiamo sviluppato si basa su una versione modificata dell'algoritmo di Dijkstra. Vengono inoltre usate 4 strutture dati: Grafo, Pila, Padri, Vertex
Queste sono le strutture dati utilizzate:

```
VERTEX {
        int id
        int dist
        int padre
        VERTEX *next
}
```

```
PADRI {
        int id
        PADRI *next
        PADRI *prec
}
```

```
PILA {
        int id
        PADRI *vettPadri
        PILA *next
}
```

```
GRAPH{
        VERTEX **AdjList    // Lista di adiacenza
        int n               // Numero di vertici nel grafo
        PADRI **p           // Array di liste che conterrà i padri di ogni nodo
        int *d              // Distanza
        int *inPila         // flag in pila
}
```

Per il nostro algoritmo usiamo Dijkstra, modificato in maniera tale da avere per ogni nodo la lista dei padri relativi ai cammini minimi: in questo modo siamo in grado, attraverso una procedura di stampa particolare di trovare tutti i cammini minimi anche se doppi (Dijkstra normalmente a parità di costo considera solo un cammino minimo).

Esempio Dijkstra modificato:
0 -> 1 -> 2 costo 3
0 -> 2 costo 3

Esempio Dijkstra:
0 -> 1 -> 2 costo 3

Per stampare questi cammini usiamo una procedura ricorsiva chiamata stampaCamminiMinimi che per ogni sorgente (da 0 a 99), parte da tutti i nodi (esclusa ovviamente la sorgente corrente) e risale tutti i diversi cammini minimi fino alla sorgente (tenendo ogni volta in considerazione il fatto di poter avere anche più padri che creano quindi diversi cammini).

3. Pseudocodice e analisi del costo computazionale

Come si può notare dalla struttura GRAPH, per i padri dei nodi viene usato un array di liste (conterrà infatti tutti i padri dei nodi che portano a un cammino minimo).

In Dijkstra la modifica è stata fatta alla funzione Relax: ora infatti quando trova un arco che se rilassato crea un cammino minore rispetto a quello precedentemente trovato, cancella tutti gli elementi della lista dei padri del nodo corrente e aggiunge a questa lista il nodo di partenza dell'arco appena rilassato (che sarà appunto il padre del nodo). Se invece la Relax rilassando un arco trova un cammino di peso uguale a quello precedente allora aggiunge il nodo di partenza dell'arco alla lista dei padri del nodo di arrivo dell'arco stesso.

```
procedure Relax (padre, figlio,G)
        if(G.d[figlio] > (G.d[padre] + G.AdjList[padre].dist)          //trovato nuovo cammino minimo
                G.d[figlio] ← G.d[padre] + G.AdjList[padre].dist)      //assegno la nuova distanza minima
                if (G.p[figlio].next) = NIL)                          //se il nodo aveva solo un padre gli do il nuovo id
                        G.AdjList[padre].id ← padre;
                else
                        eliminaCorrispondenze(G.p[figlio]);           //se il nodo aveva più padri li elimino tutti
                        Alloc(temp)                                   //tipo dato (PADRI)
                        temp.id ←  padre
                        temp.next ← NIL
                        temp.prec ← NIL
                        G.p[figlio] ← temp                            //inserisco il nuovo padre in lista
                end else
        else if (G.d[figlio] = (G.d[padre] + G.AdjList[padre].dist)  //se trovo un cammino di peso uguale ai precedenti
                Alloc(temp)                                           //tipo dato (PADRI)
                temp.id ← padre;
                temp.next ← G.p[figlio]
                temp.prec ← NIL
                G.p[figlio] ← temp
                G.p[figlio].next.prec  ←  G.p[figlio]                 //inserisco il nuovo padre in testa alla lista
        end else
end procedure

procedure eliminaCorrispondenze (list)
        Alloc(elimina)                                                //tipo dato PILA
        tempTestaLista ← list
        while (temp != NULL)
                elimina ← tempTestaLista
                Delete(elimina)
                tempTestaLista ← tempTestaLista.next
        end while
end procedure
```

La procedura ricorsiva stampaCamminiMinimi prende in input il grafo, il nodo d'arrivo del cammino, la sorgente attuale (ovvero il nodo di partenza del cammino), una copia dell'ultimo nodo (solo per non stampare l'ultima freccia nel cammino), la testa della pila, e un flag di avvenuto scorrimento di una qualsiasi lista padri durante la procedura. Inizialmente si parte dal nodo e si risale fino alla sorgente attraverso il primo della lista dei padri di ogni nodo. Risalendo la ricorsione dalla sorgente, quando si incontra il primo nodo con più padri si eseguono diversi controlli per vedere se fare o meno una push del nodo e della sua lista di padri, la lista viene poi scorsa e impostato il flagScorrimento. Per tutto il resto della ricorsione non potrà essere scorsa nessun'altra lista e non si potranno fare pop dalla pila.
Solo quando risalendo ricorsione non si scorrono liste di padri (poiché arrivate al loro ultimo elemento) e si incontrano nodi la cui lista di padri è in pila (flag inPila = TRUE) allora si ripristina la lista di padri originale di quel nodo tramite una pop della suddetta lista.

```
procedure stampaCamminiMinimi(G, nodo, sorg, ultimoNodo, pila, flagScorrimento)
   if (nodo = sorg)          //caso base, si è arrivati alla fine della ricorsione
         Visit(nodo)
         return
   end if
   stampaCamminiMinimi(G, G.p[nodo].id, sorg, ultimoNodo, pila, flagScorrimento)
   Visit(nodo)

  if (G.p[nodo].next != NULL)                          //se il nodo ha più di un padre
         if (flagScorrimento = FALSE)                  //se non sono già state scorse altre liste

                  if (G.inPila[nodo] = FALSE)          //se il nodo e la sua lista di padri non è già in pila
                        push(pila, nodo, G.p[nodo])    //si esegue una push del nodo e della sua lista di padri
                        G.inPila[nodo] = TRUE
                  end if

        G.p[nodo] ← G.p[nodo].next                     //si scorre la lista dei padri
         FlagScorrimento = TRUE
        end if
  end if
     else if (G.inPila[nodo] = TRUE && flagScorrimento = FALSE)
         G.p[nodo] ← pop(pila, nodo)                   //si ripristina la lista di padri del nodo tramite pop
         G.inPila[nodo] = FALSE
     end else
end procedure

procedure push (testaPila, nodo, listaPadri)
         Alloc(tempPila)                               //tipo dato PILA
         tempPila.id ← nodo
         tempPila.next ←  testaPila
         tempPila.vettPadri. ← listaPadri
         testaPila ← tempPila                          //si inserisce il nuovo elemento in cima alla pila
end procedure
```

```
procedure pop(testaPila,nodo)
        if (testaPila = NIL) ERRORE
        temp ← testaPila
        prec ← testaPila
        foreach (temp(id) != nodo)              //si cerca l'elemento nella pila e lo si elimina
                prec ← temp                     //con elementari operazioni su lista
                temp ← (next)
        stop
        if (prec =testaPila)
                testaPila → (next)
        else prec(next) ← temp(next)
                restituisci ← temp(vettPadri)
                elimina ← temp
                testaPila ← temp(next)
                Delete(elimina)
                return restituisci
end procedure
```

La procedura PrintAllSP utilizza l'algoritmo di Dijkstra a partire da tutte le sorgenti e la procedura stampaCamminiMinimi che come precedentemente detto stampa i cammini minimi da ogni sorgente verso ogni nodo.

```
procedure PrintAllSP(G)
   Alloc(V)
   Alloc(testaPila)
   for (scorriSorgenti = 0 to  G.n)
        sorg ← scorriSorgenti
        Dijkstra(G, sorg, V)

        for (nodo = 0 to G.n)
                if (nodo = sorg) nodo ← nodo+1
                if (nodo = G.n) break
                do
                     flagScorrimento = FALSE;
                    stampaCamminiMinimi (G, nodo, sorg, nodo, &testaPila, &flagScorrimento)
                while (flagScorrimento != 0)
     end for
end procedure
```

Procedura di cancellazione del grafo e di tutto ciò che è in esso contenuto.

```
procedure EraseGraph(G)

        //cancello la lista dei padri
        PADRI cancPadre;
        for(i = 0 to G.n)
                while(G.p[i] != NULL)
                        cancPadre ← G.p[i]              //Assegno a cancPadre il valore dell' elemento
                        G.p[i] ← G.p[i].next            //Faccio scorrere la testa
                        Delete(cancPadre)               //Dealloco l'elemento
                end while
        end for
        G.p→NIL
        Delete(G.p)

        //cancello la lista di adiacenza
        VERTEX* cancVert;
         for(i = 0 to G.n)
                while(G.AdjList[i] != NULL)
                        cancVert ← G.AdjList[i]         //Assegno a canc il valore dell'elemento
                        G.AdjList[i] ← G.AdjList[i].next //Faccio scorrere la testa
                        Delete(cancVert)                //Dealloco l'elemento
                end while
        end for

        //dealloco le singole celle e la struttura del grafo (compreso il puntatore al grafo)
        G.AdjList ← NIL
        Delete(G.AdjList)
        G.d ← NIL
        Delete(G.d)
        G.inPila ← NIL
        Delete(G.inPila)
        G ← NIL
 end procedure
```

Analisi del costo computazionale

L'algoritmo tradizionale di Dijkstra costa notoriamente $O(n^2)$, dove n è il numero di nodi. La modifica da noi effettuata a questo algoritmo nella procedura Relax non varia in ordine di grandezza il costo totale.

La procedura di stampa scorre tutti i padri di tutti i nodi (che hanno più di un padre). Nel caso peggiore, tutti gli n nodi avranno n-1 padri ($n * (n-1) = n^2$). Da questo si ha che la procedura costa $O(n^2)$.
Le funzioni push e pop non influiscono in ordine di grandezza sul costo totale della procedura di stampa.

Dato che Dijkstra e la procedura stampaCamminiMinimi costano entrambi $O(n^2)$, e sono ripetuti "numero di nodi" volte (in PrintAllSP) si può dire che entrambi costano $O(n^3)$, per un totale di $2(n^3)$, che in ordine di grandezza è $O(n^3)$.