Laboratorio di Algoritmi e Strutture Dati

Esercitazioni del 22 Gennaio 2013

Esercizio 4: Algoritmo di Dijkstra

Si richiede di implementare l'algoritmo di Dijkstra per i cammini minimi su un grafo orientato pesato $G = (V, E)$, la cui funzione peso attribuisce agli archi solo pesi interi non negativi:

$$w \colon E \to \mathbb{N} \,.$$

Prima parte

Si modifichi l'implementazione di grafi orientati `directedgraph.c` implementata come soluzione dell'esercizio 1, in modo che a ogni vertice `v` corrisponda una struttura del tipo (potete personalizzarla, questo è solo un suggerimento):

```
typedef struct vertex {
    edge *adj;
    int d;
    int pred;
    int priorityindex;
} vertex;
```

dove `adj` è la testa della lista di `edge` che rappresenta gli archi `(v, w)` uscenti dal vertice `v` stesso, mentre gli altri campi si utilizzano per calcolare e memorizzare il cammino minimo.

In particolare, ogni struttura `edge` è del tipo

```
typedef struct edge {
    int vertexlabel;
    int weight;
    struct edge *next;
} edge;
```

dove `vertexlabel` è l'etichetta `w` del vertice in cui entra l'arco `(v, w)` stesso, `weight` è il suo peso, e infine `next` punta al prossimo arco uscente da `v`.

Modificate le funzioni `creategraph, readgraph, vertexinsert` in modo tale che `readgraph` legga la specifica del grafo da un file il cui formato è il seguente

```
v  e

v_11  v_12  w_1
v_21  v_22  w_2
...
v_e1  v_e2  w_e
```

dove `v` ed `e` indicano rispettivamente il numero di vertici e di archi del grafo, e ogni riga successiva specifica un arco tra il vertice `vi1` e il vertice `vi2` di peso `wi`.

Seconda parte

Modificando la coda di max-priorità per interi implementata per l'algoritmo `heapsort`, visto in un'esercitazione precedente, si implementino alcune operazioni di una coda di min-priorità sul valore del campo `d` di una `struct vertex` (Questo valore conterrà in ogni momento il costo di un cammino (il migliore trovato fino a quel momento) tra un vertice sorgente fissato e il vertice associato alla `struct vertex` stessa).

In particolare: si realizzino, adattando le analoghe funzioni della coda di max-priorità per interi, le funzioni: `heapify`, `buildheap`, `extractminheap`, `decreasevalue`.

NOTA: La coda può essere implementata fisicamente ancora come un array di interi, ma:

- Il valore intero contenuto in ogni elemento è l'indice di un vertice nell'array che rappresenta i vertici del grafo.
- La chiave su cui organizzare lo heap per priorità minima è il campo `d` del vertice del grafo.

NOTA: E' importante mantenere un collegamento diretto tra gli elementi della coda e i vertici del grafo in entrambe le direzioni: per il collegamento da vertice a elemento nella coda, si usi il campo `priorityindex` di `struct vertex`.

NOTA: Può essere utile manipolare la coda attraverso una struttura che contenga, oltre all'indirizzo iniziale dell'array dinamico che memorizza gli elementi della coda stessa, anche un campo intero che memorizzi in ogni momento quanti elementi sono rappresentati nella coda stessa (Ogni operazione `extractminheap` accorcia la coda di un elemento).

Terza parte

In questa parte si richiede di implementare sulla struttura delineata l'algoritmo di Dijkstra vero e proprio.

In particolare l'algoritmo avrà in input un grafo G e un suo vertice s, e istanzierà i campi `d` e `pred` di ogni vertice di G in modo che essi codifichino un *albero di cammini minimi*, in cui ogni cammino semplice da s a qualunque altro vertice v raggiungibile da s è un *cammino minimo* da s a v.

Inizializzazione:

Per ogni vertice del grafo diverso da s si deve porre `d` $= \infty$. Mentre il campo `d` di s deve essre posto a 0.

Per ogni vertice del grafo si deve porre `pred` = indefinito.

In entrambi i casi si può porre $\infty = indefinito = -1$, avendo l'accortezza di *modificare* l'implementazione della coda di min-priorità in modo che un confronto di un valore x con -1 dichiari sempre -1 come il maggiore dei due.

Si usi `buildheap` per costruire la coda di priorità contenente tutti i vertici. Dato che solo il campo `d` di s è 0 e tutti gli altri sono ∞ (cioè, -1), s sarà l'elemento in testa alla coda.

Rilassamento:

L'algoritmo di Dijkstra consiste in una serie di *rilassamenti* della stima `d` del cammino minimo da s a un altro vertice v, attraverso la considerazione dell'informazione fornita da un arco (u, v) entrante in v.

In particolare, sia $w(u, v)$ il peso di un siffatto arco e siano $d(v)$ e $d(u)$ i campi `d` di u e v; analogamente sia $pred(v)$ il campo `pred` relativo al vertice v. Allora:

$$\text{Se } d(v) > d(u) + w(u, v)$$

possiamo *rilassare* l'arco (u, v) ponendo

$$d(v) = d(u) + w(u, v)\,, \qquad pred(v) = u\,.$$

In tal modo abbiamo migliorato la stima per $d(v)$ e indicato che l'ultimo arco da percorrere per arrivare a v è (u, v).

Il ciclo principale:

Dopo aver eseguito l'inizializzazione, l'algoritmo procede in questo modo:

Itera fino a quando la coda non diventa vuota i seguenti passi:

- Estrae la testa u dalla coda di min-priorità
- Per ogni vertice v adiacente al vertice estratto u, esegue il rilassamento dell'arco (u, v).

NOTA: Se il rilassamento modifica il campo `d` di v, allora bisogna effettuare una `decreasevalue` sull'elemento corrispondente a v, impostando il nuovo valore di `d`.

Stima dei costi

Sia dato il grafo $G = (V, E)$.

L'inizializzazione comporta $O(|V|)$ assegnamenti e la costruzione tramite `buildheap` della coda di priorità: questo costa $O(|V|)$ operazioni.

Nel ciclo principale i vertici vengono estratti uno alla volta, e ogni arco è considerato una volta sola.

L'estrazione di un vertice dalla coda costa $O(\log |V|)$, mentre il rilassamento di un arco comporta $O(1)$ più il costo di una `decresevalue`, che costa $O(\log |V|)$.

Dunque il costo totale dell'algoritmo di Dijkstra, basato sulle code di min-priorità è: $O((|V|+|E|) \log |V|)$.

Quarta parte

Si richiede di realizzare un programma che lanciato come

`dijkstra` $nomefile$

legga la specifica del grafo orientato pesato dal file di nome $nomefile$, esegua l'algoritmo di Dijkstra sul file stesso e stampi per ogni vertice v di indice > 0 un cammino minimo da 0 a v, oppure un messaggio che segnali la non esistenza di un cammino siffatto. Il cammino si deve visualizzare come la sequenza ordinata delle coppie:

$$(0, v_1)(v_1, v_2) \cdots (v_k, v)\,.$$

Per effettuare la stampa, si noti che alla fine dell'esecuzione dell'algoritmo, ogni vertice v diverso dalla sorgente $s = 0$, contiene nel suo campo `pred` l'indice del vertice *precedente* nel cammino minimo.