Giorno 8

1 Cammini minimi da sorgente unica

Sia $G = (V, E)$ un grafo, e sia $w : E \to \mathbb{R}$ una funzione che associa ad ogni arco un *peso*. Un grafo associato ad una funzione peso si dice *pesato*.

Def. 1.1. Il *peso di un cammino* è la somma dei pesi degli archi che lo compongono.

Def. 1.2. Il cammino di peso minimo tra due nodi di un grafo pesato si dice *cammino minimo*

Def. 1.3. La distanza tra due nodi $\delta(u, v)$ è il peso del cammino minio tra i due nodi.

In questa sezione analizziamo il problema di individuare i cammini minimi tra un nodo s che chiamiamo *sorgente* e tutti gli altri nodi del grafo. Alcuni problemi simili possono essere ricondotti a questo e risolti utilizzando gli stessi metodi:

1. **Cammini minimi a destinazione unica**: Si invertono le direzioni degli archi, si tratta la destinazione come sorgente;
2. **Cammini minimi tra una coppia di nodi**: Sia uno dei due nodi la sorgente, il problema è risolto da un algoritmo che trova i cammini minimi da sorgente unica.
3. **Cammini minimi tra tutte le coppie di nodi**: Può essere risolto iterando un algoritmo che trova i cammini minimi da sorgente unica, ma esiste una soluzione migliore.

1.1 Algoritmo di Dijkstra

Ad ogni nodo v sono associati due campi:

- $v.d$, stima per eccesso del peso del cammino tra s e v: $\quad v.d \geq \delta(s, v)$
- $v.p$, alla fine dell'esecuzione è il precedente di v nel cammino tra s e v.

1. Inizialmente tutti i nodi tranne la sorgente hanno $v.p = \text{nil}$, $v.d = \infty$, mentre $s.d = 0$.
2. Si inseriscono tutti i nodi in una coda di min-priorità con priorità $v.d$.
3. Si estrae ripetutamente, finché la CDP non è vuota, il nodo v la cui stima è minima, che alla prima iterazione sarà la sorgente, e per ogni elemento u di $Adj[v]$, se $u.d > v.d + w(v, u)$:
   (a) Si pone $u.d = v.d + w(v, u)$;
   (b) Si pone $u.p = v$;
   (c) Si diminuisce, nella coda di priorità, il valore della chiave di u, ponendola $= v.d + w(v, u)$

   Queste tre operazioni sono dette "di rilassamento".

1.2 Complessità

```
Dijkstra(g, s, w):
    PQ = new min-priority queue
    s.d = 0;
    for(v ∈ g.V):
        v.d = ∞
        v.p = nil;
        insert(PQ, v);
    while(PQ ≠ ∅):
        v = extractMin(PQ);
        for(u ∈ Adj[v]):
            if(u.d > v.d + w(v, u)):
                u.d = v.d + w(v, u);
                decreaseKey(PQ, u, v.d + w(
                    v, u))
```

L'algoritmo esegue $\Theta(|V|) \cdot T(insert)$ operazioni di inizializzazione, e il while costa $\Theta(|E|) \cdot (T(extractMin) + T(decreaseKey))$.
Assumendo di usare un min-heap come coda di priorità:

1. costruire un Min-Heap costa $O(|V|)$
2. extractMin costa $O(\log |V|)$
3. decreseKey costa $O(\log |V|)$ (+ tempo ricerca, ma teniamo riferimenti tra pos heap e nodi)

Quindi il costo totale è $O(\log |V| \cdot (|V| + |E|))$

1.2.1 Correttezza dell'algoritmo

Teorema 1.4. Alla fine dell'esecuzione si ha che $\forall v \in V : v.d = \delta(s, v)$

Dimostrazione.

($\star$) Notiamo che per definizione $v.d \geq \delta(s, v)$;

Sia S l'insieme dei vertici estratti dalla coda. Dimostriamo per induzione sul numero di vertici in S.
Hp. Ind: Per ogni $v \in S : v.d = \delta(s, v)$.

- (Base) Il primo vertice estratto è s, la cui distanza da s è zero, ok.
- (Passo)
  1. Sia v il prossimo vertice rimosso dalla coda. Sia P il cammino minimo tra s e v.
  2. Sia (x, y) il primo arco di P ad uscire da S (y potrebbe essere v, se il sottocammino $s \rightsquigarrow x$ fosse interamente in S).
  3. Dimostriamo che $y.d = \delta(s, y)$:
     Per ipotesi induttiva sappiamo che $x.d = \delta(s, x)$.
     All'estrazione di x il vertice y è stato "rilassato", quindi:
     $$y.d \leq x.d + w(x, y) = \delta(s, x)w(x, y) = \delta(s, y)$$
     Quindi $y.d \leq \delta(s, y)$. Per ($\star$) abbiamo $y.d \geq \delta(s, y) \implies y.d = \delta(s, y)$
  4. $$v.d \underset{(\star)}{\geq} \delta(s, v) \underset{\text{pesi non neg}}{\geq} \delta(s, y) \underset{(3)}{=} y.d \underset{v \text{ estr. prima di } y}{\geq} v.d$$
     Quindi abbiamo $v.d \geq \delta(s.v) \geq v.d$, quindi $v.d = \delta(s, v)$.

■