Appunti del corso

Cammini minimi e algoritmo di Dijkstra

Nome Cognome

Autore	**Nome Cognome**
Corso	**Algoritmi e strutture dati**
Ultimo aggiornamento	**2 novembre 2026**

Come leggere questo documento

Questo documento raccoglie le slide della lezione 04 sui cammini minimi e gli appunti a mano della lezione del 12 marzo. La struttura segue le slide; gli appunti sono riportati nei riquadri `mynotes`, senza aggiunte. Le soluzioni e le osservazioni aggiunte da chi scrive sono sempre segnalate in modo esplicito.

DEFINIZIONE

Esempio di definizione

Una definizione raccoglie in una forma precisa il significato di un concetto.

ESEMPIO

Esempio di esempio

Un esempio svolto passo per passo mostra come si applica un concetto.

OSSERVAZIONE

Esempio di osservazione

Un'osservazione chiarisce un punto delicato o segnala un errore nelle fonti.

DAI MIEI APPUNTI

Esempio di appunto: riportato così com'è, senza aggiunte.

PUNTO CHIAVE

Il punto chiave è ciò che il lettore deve ricordare.

ESERCIZIO

Esempio di esercizio

Quanto vale $1 + 1$?

Soluzione proposta. 2.

Indice

Come leggere questo documento iii

4 Cammini minimi: l'algoritmo di Dijkstra 1

4.1 Il problema 1
4.2 L'idea 1
4.3 Un esempio 2
4.4 Lo pseudocodice 3
4.5 Costo 4
4.6 Pesi negativi 5
4.7 Riepilogo 5

Capitolo 4

Cammini minimi: l'algoritmo di Dijkstra

Su una rete di collegamenti vogliamo spesso sapere qual è il percorso più economico da un punto di partenza verso tutti gli altri. L'algoritmo di Dijkstra risolve questo problema quando tutti i pesi sono non negativi, con un'idea semplice: visitare i vertici in ordine di distanza crescente.

Slide 04 · Appunti a mano, lezione del 12 marzo

4.1 Il problema

Sia dato un grafo orientato $G = (V, E)$, in cui ogni arco $(u, v) \in E$ ha un *peso* $w(u, v) \geq 0$, e un vertice $s \in V$ detto **sorgente**. Vogliamo calcolare, per ogni vertice v, la distanza minima $d(s, v)$ da s a v. La distanza di un cammino è la somma dei pesi dei suoi archi.

DEFINIZIONE

Distanza minima

Per ogni vertice $v \in V$, la *distanza minima* da s a v è

$$d(s, v) = \min\{\, w(p) \;:\; p \text{ è un cammino da } s \text{ a } v \,\},$$

dove $w(p)$ è la somma dei pesi degli archi del cammino p.

4.2 L'idea

L'algoritmo mantiene, per ogni vertice v, una stima $d[v]$ della distanza da s: all'inizio $d[s] = 0$ e $d[v] = \infty$ per ogni altro vertice. Ad ogni passo *estrae* il vertice u non ancora visitato con $d[u]$

minima, e ne **rilassa** gli archi uscenti: se passando per u si arriva a v con un costo minore di $d[v]$, aggiorniamo la stima.

DEFINIZIONE

Rilassamento

L'operazione di *rilassamento* sull'arco (u, v) è

$$\textsc{Relax}(u,v): \quad \text{se } d[u] + w(u,v) < d[v], \\ \text{allora } d[v] \leftarrow d[u] + w(u,v) \text{ e } pred[v] \leftarrow u.$$

DAI MIEI APPUNTI

Una volta estratto u, $d[u]$ non cambia più!! (è definitiva). Con pesi negativi → un arco dopo potrebbe abbassare una distanza già "chiusa". Il prof dice: pensare a $d[v]$ come a un "limite superiore" che scende.

PUNTO CHIAVE

Un vertice estratto ha già la distanza giusta, e non va più aggiornato. La correttezza dell'algoritmo si basa su questo fatto, che richiede pesi non negativi (vedi sezione 4.6).

4.3 Un esempio

Consideriamo il grafo di fig. 4.1, con gli archi $s \to a$ di peso 4, $s \to b$ di peso 1, $b \to a$ di peso 2, $a \to c$ di peso 1 e $b \to c$ di peso 5. Il cammino più economico verso a non è l'arco diretto, ma passa per b.

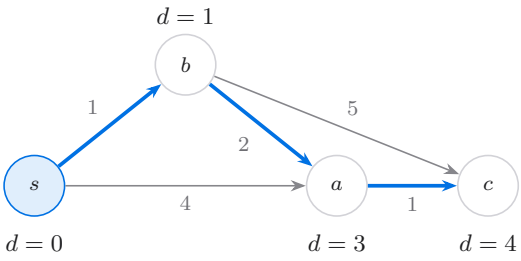


Figura 4.1. Grafo dell'esempio delle slide (disegnato di nuovo, Slide 05). Il cammino più economico da s ad a non è l'arco diretto $s \to a$ di peso 4, ma $s \to b \to a$ di peso 3. I valori sotto e sopra i vertici sono le distanze finali d.

Figura 4.1 mostra il risultato. Il box seguente segue l'esecuzione passo per passo.

ESEMPIO

Esecuzione passo per passo

Partiamo con $d[s] = 0$ e $d[a] = d[b] = d[c] = \infty$. Nella colonna di destra, lo stato indica i valori $(d[s], d[a], d[b], d[c])$.

Passo	Cosa succede	Stato
1	Estraiamo s ($d = 0$) e rilassiamo $s \to a$ e $s \to b$	$(0, 4, 1, \infty)$
2	Estraiamo b ($d = 1$) e rilassiamo $b \to a$ (scende $d[a]$ a 3) e $b \to c$	$(0, 3, 1, 6)$
3	Estraiamo a ($d = 3$) e rilassiamo $a \to c$ (scende $d[c]$ a 4)	$(0, 3, 1, 4)$
4	Estraiamo c ($d = 4$), che non ha archi uscenti	$(0, 3, 1, 4)$

Al termine $d[s] = 0$, $d[b] = 1$, $d[a] = 3$ e $d[c] = 4$, come nelle slide.

4.4 Lo pseudocodice

ALGORITMO

Algoritmo di Dijkstra

```
1  function Dijkstra(G, s) returns distanze d e predecessori pred
2      for each v ∈ V do
3          d[v] ← ∞, pred[v] ← nil
4      d[s] ← 0
5      Q ← coda di priorità con tutti i vertici, ordinata per d
6      while Q ≠ ∅ do
7          u ← Extract-Min(Q)                    // d[u] è ora definitiva
8          for each arco (u, v) uscente da u do
9              Relax(u, v)
10     return d, pred
```

Una possibile implementazione in Python è riportata sotto. Non è nelle slide: l'abbiamo scritta noi seguendo lo pseudocodice, usando la libreria `heapq` per la coda di priorità. Il grafo è un dizionario che associa a ogni vertice la lista delle coppie (vicino, peso).

PYTHON

Dijkstra (implementazione nostra)

```python
import heapq

def dijkstra(graph, s):
    d = {v: float("inf") for v in graph}
    pred = {v: None for v in graph}
    d[s] = 0
    Q = [(0, s)]                        # priority queue of (distance, vertex)
    done = set()
    while Q:
        _, u = heapq.heappop(Q)         # EXTRACT-MIN
        if u in done:                   # stale entry, already extracted
            continue
        done.add(u)
        for v, w in graph[u]:           # RELAX(u, v) for each edge (u, v)
            if d[u] + w < d[v]:
                d[v] = d[u] + w
                pred[v] = u
                heapq.heappush(Q, (d[v], v))
    return d, pred
```

Con il grafo dell'esempio, `dijkstra(graph, s)` restituisce le distanze $d(s,s) = 0$, $d(s,b) = 1$, $d(s,a) = 3$ e $d(s,c) = 4$.

4.5 Costo

Il costo dipende da come è realizzata la coda di priorità. Chiamiamo V il numero di vertici ed E il numero di archi.

Coda di priorità	Estrazione del minimo	Costo totale
Heap binario	$O(\log V)$	$O((V+E)\log V)$
Array	$O(V)$	$O(V^2)$

OSSERVAZIONE

Refuso nelle slide

La nota a lezione segnala che la slide 6 riporta $O(E \log E)$ per la realizzazione con array. È un refuso: con l'array ogni estrazione del minimo scorre i vertici e costa $O(V)$, quindi il costo totale è $O(V^2)$, come nella tabella sopra.

4.6 Pesi negativi

Il punto debole dell'algoritmo sono i pesi negativi. Se un arco ha peso negativo, la stima di un vertice già estratto può ancora scendere, mentre Dijkstra la considera definitiva. Per grafi con pesi negativi si usa quindi un altro algoritmo, quello di Bellman-Ford.

DAI MIEI APPUNTI

Controesempio visto in aula: $s \to a$ (2), $s \to b$ (3), $b \to a$ (-2) ...

PUNTO CHIAVE

Dijkstra richiede pesi non negativi. Con pesi negativi un vertice già estratto può ricevere una distanza minore, e l'algoritmo non se ne accorge.

4.7 Riepilogo

Tecnica	Idea	Costo
Dijkstra con heap binario	Estrae il minimo e rilassa gli archi	$O((V+E)\log V)$
Dijkstra con array	Stessa idea, minimo cercato in un array	$O(V^2)$

PUNTO CHIAVE

Dijkstra trova le distanze minime da una sorgente quando tutti i pesi sono non negativi: ogni vertice estratto ha distanza definitiva. Con pesi negativi serve un altro algoritmo, come Bellman-Ford.

ESERCIZIO

Esercizio dalle slide

Perché Dijkstra può fallire con pesi negativi? Fornire un controesempio.

Soluzione proposta. Consideriamo il grafo con gli archi $s \to a$ di peso 2, $s \to b$ di peso 3, $b \to a$ di peso -2 e $a \to c$ di peso 1. Dijkstra estrae prima s, poi a con $d[a] = 2$, e rilassa $a \to c$, quindi $d[c] = 3$. Poi estrae b con $d[b] = 3$ e rilassa $b \to a$: $3 - 2 = 1 < 2$, quindi $d[a]$ scende a 1. Ma a è già stato estratto, e i suoi archi non vengono più rilassati, quindi $d[c]$ resta 3. Il cammino $s \to b \to a \to c$ ha invece peso $3 - 2 + 1 = 2$: la distanza vera è $d(s, c) = 2$, non 3.