* Si discuta la complessità computazionale dell’algoritmo di Bellman-Ford.
  + L’algoritmo di Bellman-Ford ha una complessità ), in quanto viene fato un numero di iterazioni pari al numero di nodi con un ciclo che esamina le etichette duali su tutti gli archi . Qualora esistesse un ciclo negativo (abbiamo aggiornato almeno un’etichetta) oppure tutte le etichette sono stabili, l’algoritmo termina la sua esecuzione.
* Si discuta la complessità computazionale dell’algoritmo di Dijkstra per il problema del cammino minimo
  + Il prof riporta: “Si consiglia di riportare i passi dell’algoritmo di Dijkstra e, quindi, discuterne la complessità come fatto nelle dimostrazione della Proprietà 10 alla fine del paragrafo 6.1 delle dispense sugli algoritmi per problemi di cammino minimo. Quindi si può approfondire con qualche dettaglio sull’influenza della scelta di opportune strutture dati per migliorare l’efficienza, come dal paragrafo 6.2 delle stesse dispense.”

Per la complessità:

* consideriamo una complessità lineare di analisi di tutti i nodi
* cerchiamo il minimo e dobbiamo trovare un minimo dipendente dalla cardinalità di tutti gli elementi; *usando una lista*, avremo
* controlliamo gli archi che escono da un solo nodo, con
* la complessità sarebbe
* calcolando invece la complessità ammortizzata (cioè, considerando tutte le iterazioni e nel caso peggiore; se le operazioni costose sono poco frequenti, compenso “ammortizzando” con operazioni poco costose in modo deterministico), consideriamo ogni volta gli archi che escono dall’etichetta ottima; controlliamo *al massimo una volta* tutti gli archi.

Avremo quindi

La complessità finale, usando e come liste è

Implementando invece e come heap, possiamo fare la ricerca di minimo in tempo costante, che ora ci costa almeno , dato che se fosse min-heap, sta nella radice.

* Per creare un heap (*create*), costerà
* Per l’inserimento (*insert*), dovremo mantenere la proprietà di heap (facendo in modo che il nodo padre sia più grande/più piccolo di tutti i nodi figli e sia bilanciato, quindi con numero minimo di figli), con costo
* Una volta trovato il minimo (*find-min*), dobbiamo anche eliminare il nodo radice e ristabilire tutto lo heap.
* Quindi, eliminare il minimo (*delete-min*) costerà
* L’aggiornamento dell’etichetta (*decrease-key*) costerà

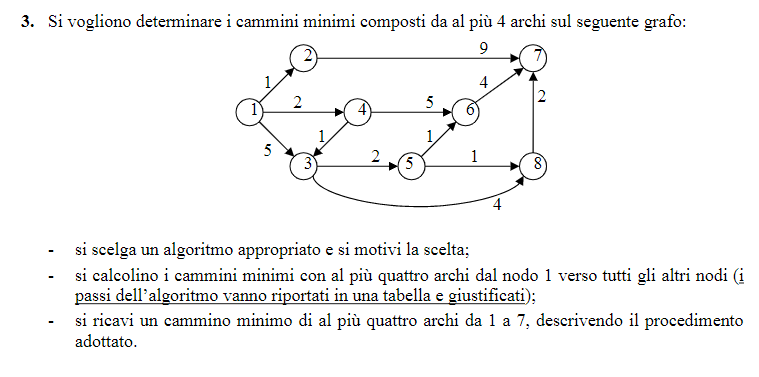
La complessità finale, usando e come heap è

Quindi, avremo ; tuttavia:

* se avessimo a che fare con un grafo completo/denso, meglio implementare con le liste;
* se avessimo a che fare con un grafo sparso, meglio implementare con gli heap.

Se usassimo un Fibonacci heap, si dimostra che la complessità sarà .

Simulazione Esame 2018/2019 – Esercizio 3 Grafi Raccolta “Esercizi vari”



1. Nella scelta dell’algoritmo, notiamo che ci viene dato un massimo numero di archi da dover rispettare, pertanto si può applicare solo l’algoritmo di Bellman – Ford, l’unico che dà la possibilità di calcolo dei cammini minimi sulla base di un massimo numero di archi. Applicheremo Bellman – Ford fermandoci alla quarta iterazioni, con un numero di archi e iterazioni pari a
2. Ora, il calcolo dei cammini minimi, riportati in tabella (controllando ad ogni passo miglioramenti rispetto alla riga precedente e segnando in colonna apposita gli aggiornamenti effettuati; i predecessori sono segnati come semplici pedici senza parentesi):

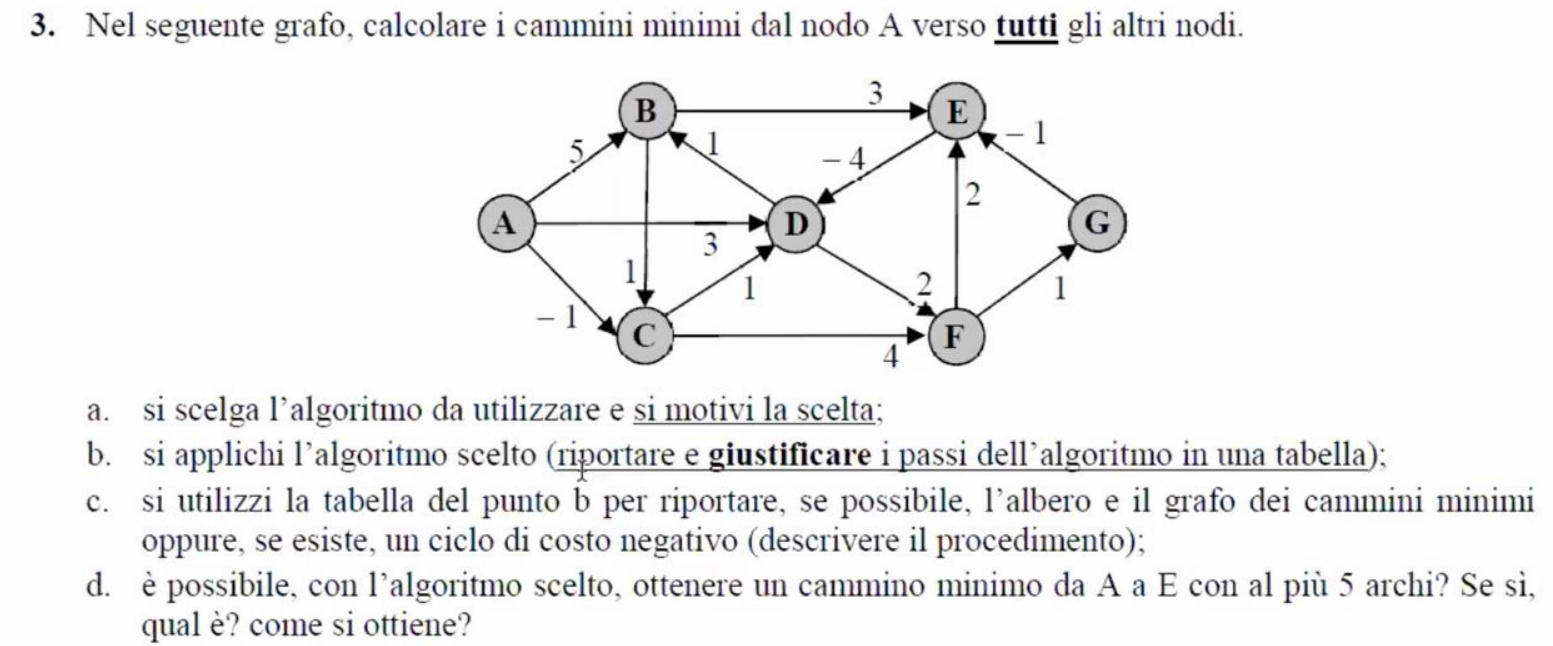
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iterazione | Nodo 1 | Nodo 2 | Nodo 3 | Nodo 4 | Nodo 5 | Nodo 6 | Nodo 7 | Nodo 8 | Aggiornamenti |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

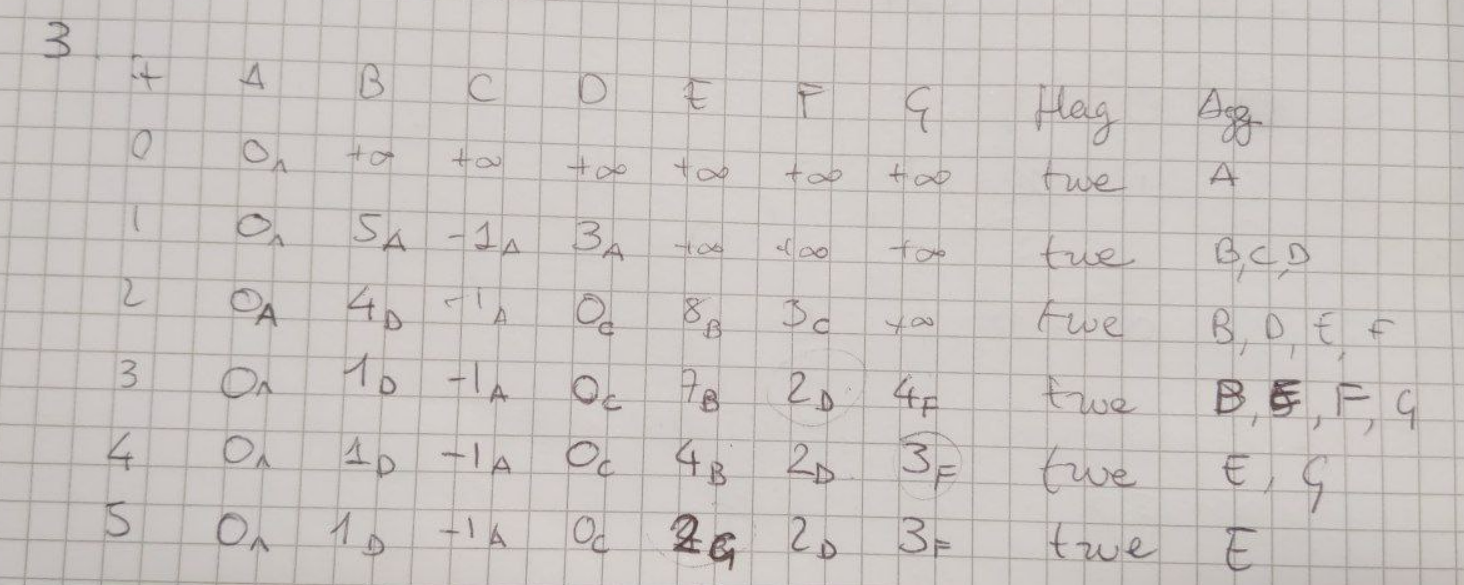
Le etichette di una riga sono ottenute controllando i vincoli duali su tutti gli archi uscenti dai nodi “aggiornati” della riga (iterazione) precedente secondo la dove è uno degli archi uscenti da un nodo aggiornato all’iterazione precedente, l’etichetta corrente (sulla riga corrente) del nodo , è l’etichetta del nodo all’iterazione (riga) precedente e è il costo dell’arco .

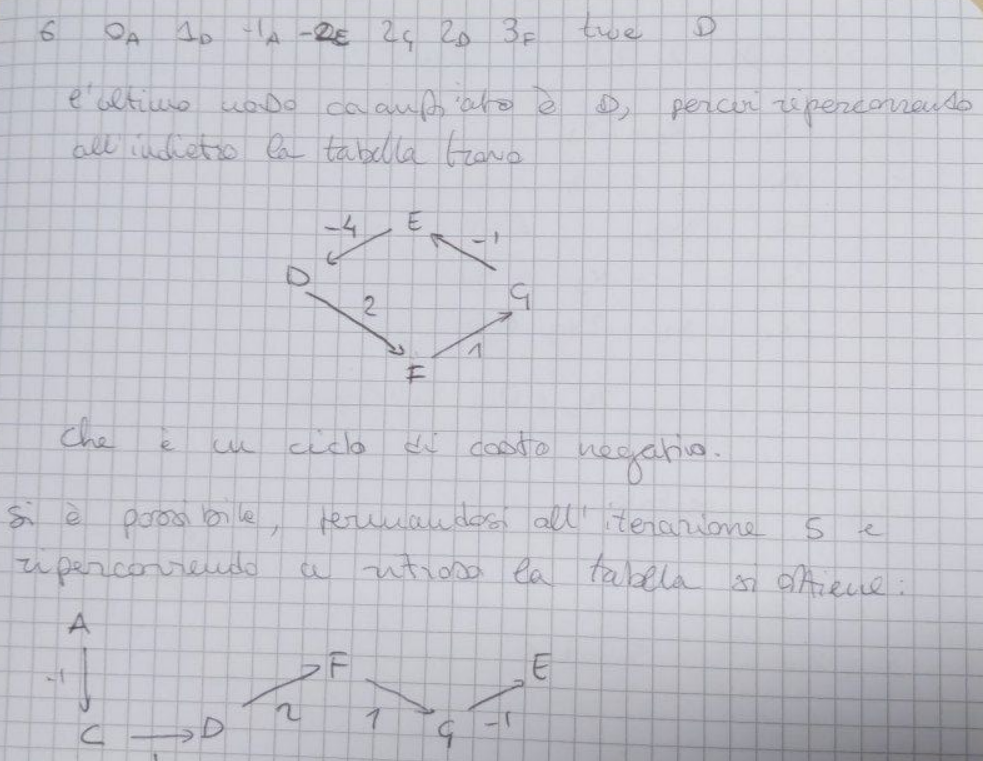
Grazie al fatto di utilizzare l’etichetta del nodo precedente, viene assicurata la scelta del cammino minimo con archi (qua si può fare un qualsiasi esempio numerico dove si va a scegliere, prendendo ad esempio l’ultima iterazione, qui un cammino che ha costo migliore considerando l’etichetta precedente e non quella corrente, qui , dimostrando la validità di quanto fatto).

1. Un cammino minimo con al massimo archi da viene fatta seguendo la catena dei predecessori, quindi, partendo dal nodo (ultimo nodo in generale) con si considera quindi:

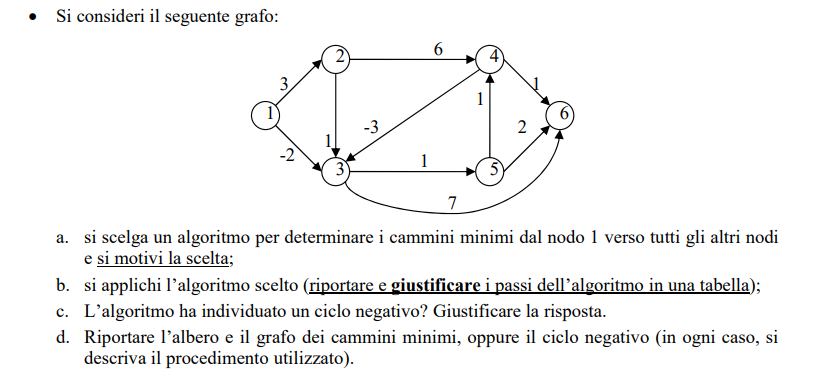
Questo equivale al percorso , con costo 9.







Carrellata di Esercizi da Raccolta “Esercizi vari”



a) In questo grafo, utilizziamo Bellman-Ford, in quanto esistono archi con costi ridotti negativi.

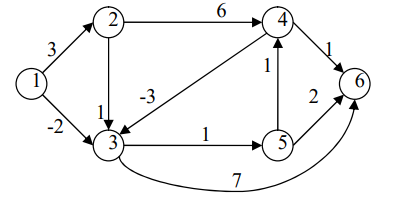
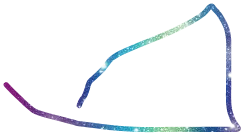
b) Ora, il calcolo dei cammini minimi, riportati in tabella (controllando ad ogni passo miglioramenti rispetto alla riga precedente e segnando in colonna apposita gli aggiornamenti effettuati; i predecessori sono segnati come semplici pedici senza parentesi). Dato che non siamo limitati dai max-hop, andremo a creare la tabella iterando un numero pari di volte al numero di nodi (quindi, facendoli tutti):

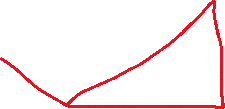
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Iterazione | Nodo 1 | Nodo 2 | Nodo 3 | Nodo 4 | Nodo 5 | Nodo 6 | Aggiornamenti |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | // |
|  |  |  |  |  |  |  | // |

c) L’algoritmo non ha individuato un ciclo negativo in quanto non ha terminato con *flag\_aggiornato=true*, cioè non ha aggiornato fino alla fine delle iterazioni.

d) Per individuare l’albero dei cammini minimi, si percorre a ritroso la catena dei predecessori partendo dall’ultima iterazione, quindi, con costo . Si evidenzia in rosso il cammino minimo.

Per individuare il grafo dei cammini minimi, composto da *tutti* i cammini minimi, esamino il grafo e scrivendo le etichette ottime, verifico se esistono altri cammini minimi. In questo caso, albero e grafo coincidono, non avendo altre etichette di costo minimo.

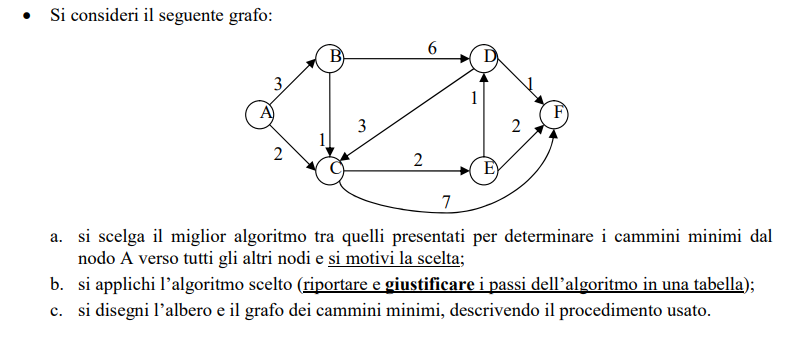




Legenda:

* Rosso – Albero
* Arcobaleno - Grafo





a) Il miglior algoritmo in questo contesto è Dijkstra, in quanto più efficiente e usabile in quanto tutti i costi ridotti sono positivi.

b) Ora, il calcolo dei cammini minimi, riportati in tabella. Si ricorda che:

* rappresenta l’etichetta minima di ogni iterazione
* rappresentano le etichette ancora da fissare
* Il segno \* rappresenta l’etichetta fissata
* Il segno – rappresenta l’etichetta controllata ma non aggiornata
* Il segno x rappresenta l’etichetta non controllata perché il nodo è già fissato
* Gli spazi vuoti servono per indicare che non considero più l’etichetta nelle varie iterazioni in quanto fissata
* L’algoritmo termina quando non ci sono più nodi in

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iterazione | Nodo | Nodo | Nodo | Nodo | Nodo | Nodo |  |  |
|  |  |  |  |  |  |  |  |  |
|  | \* |  |  |  |  |  |  |  |
|  |  | - | \* |  |  |  |  |  |
|  |  | \* |  |  | - |  |  |  |
|  |  |  |  | - | \* | - |  |  |
|  |  |  |  | \* |  | - |  |  |
|  |  |  |  |  |  | \* |  |  |

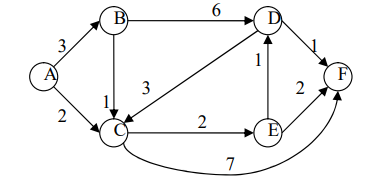
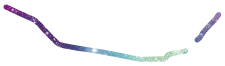
Ad ogni iterazione, percorriamo il grafo e scegliamo il percorso con costo minore. Ad ogni iterazioni, scegliamo e fissiamo un’etichetta che ha costo minore, scremando ad ogni iterazione quelle da controllare e avere sempre in mano il costo minimo. Anche in questo caso, come per gli algoritmi label correcting in caso di convergenza alla soluzione ottima, le etichette calcolate e i relativi puntatori rappresentano, rispettivamente, una soluzione duale ammissibile e i predecessori su dei cammini dall’origine ai diversi nodi. Questo funziona non avendo costi negativi.

c)

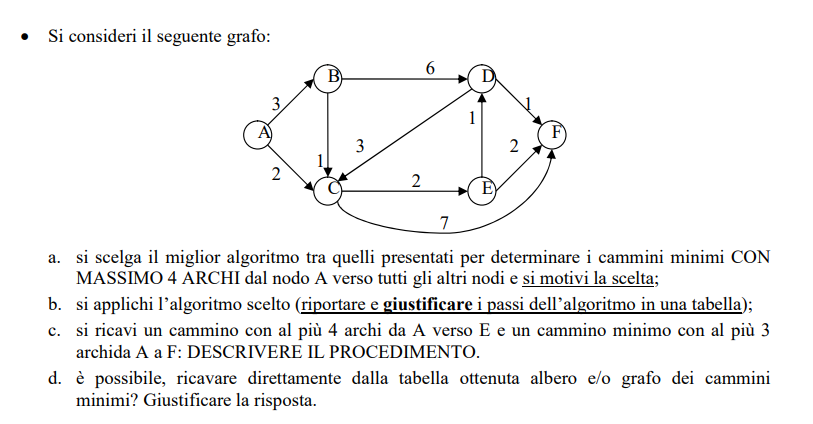
Pertanto, come abbiamo visto per gli algoritmi label correcting in caso di convergenza, è possibile derivare l’albero (risp. il grafo) dei cammini minimi attraverso i puntatori ai predecessori (risp. la verifica della saturazione dei vincoli duali sugli archi).

Concretamente, avremo che:

* In rosso riportiamo l’albero dei cammini minimi
* In arcobaleno riportiamo il grafo (composto come sempre da *tutti* i cammini minimi, quindi fissando le etichette ottime e scegliendo come cammino sia l’albero che tutte le altre etichette con costo )







1. Nella scelta dell’algoritmo, notiamo che ci viene dato un massimo numero di archi da dover rispettare, pertanto si può applicare solo l’algoritmo di Bellman – Ford, l’unico che dà la possibilità di calcolo dei cammini minimi sulla base di un massimo numero di archi. Applicheremo Bellman – Ford fermandoci alla quarta iterazioni, con un numero di archi e iterazioni pari a
2. Ora, il calcolo dei cammini minimi, riportati in tabella (controllando ad ogni passo miglioramenti rispetto alla riga precedente e segnando in colonna apposita gli aggiornamenti effettuati; i predecessori sono segnati come semplici pedici senza parentesi):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Iterazione | Nodo A | Nodo B | Nodo C | Nodo D | Nodo E | Nodo F | Aggiornamenti |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

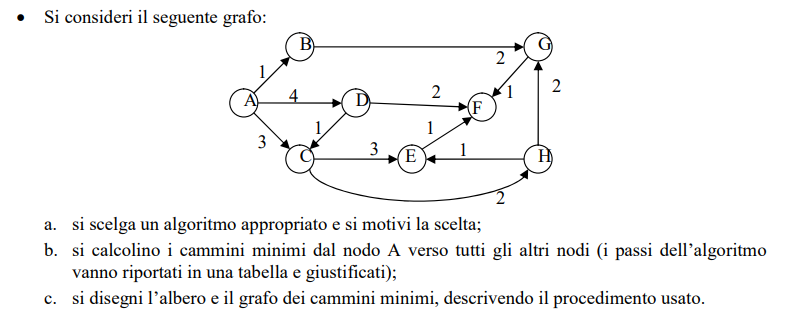
Le etichette di una riga sono ottenute controllando i vincoli duali su tutti gli archi uscenti dai nodi “aggiornati” della riga (iterazione) precedente secondo la dove è uno degli archi uscenti da un nodo aggiornato all’iterazione precedente, l’etichetta corrente (sulla riga corrente) del nodo , è l’etichetta del nodo all’iterazione (riga) precedente e è il costo dell’arco .

1. Un cammino minimo con al più archi da verso , seguendo la catena dei predecessori e considerano lo stesso , viene dato da con costo

Un cammino minimo con al più archi da verso , considerando lo stesso essendo l’ultimo nodo che è stato aggiornato, e seguendo la catena dei predecessori, viene dato da con costo . Usando solo i predecessori, non abbiamo un albero dei cammini minimi.

Per l’individuazione di entrambi i cammini minimi, seguo come detto i predecessori partendo dal nodo indicato e diminuendo di 1, considerando la diminuzione delle iterazioni ad ogni passo e guardando la riga precedente. Infatti, nel caso si risolva un problema di cammino minimo con un massimo numero di hop, non si può parlare di albero dei cammini minimi.

1. L’albero dei cammini minimi può essere ottenuto dalla tabella seguendo la catena dei predecessori (solo nel caso della prima richiesta), mentre per quanto riguarda il grafo, questo viene dato da tutti e soli i cammini minimi; in questo caso specifico, coincide con il percorso con al più 3 archi, in quanto quello da 4 non può essere considerato dati i motivi enunciati sopra.



a) Il miglior algoritmo in questo contesto è Dijkstra, in quanto più efficiente e usabile in quanto tutti i costi ridotti sono positivi.

b) Ora, il calcolo dei cammini minimi, riportati in tabella. Si ricorda che:

* rappresenta l’etichetta minima di ogni iterazione
* rappresentano le etichette ancora da fissare
* Il segno \* rappresenta l’etichetta fissata
* Il segno – rappresenta l’etichetta controllata ma non aggiornata
* Il segno x rappresenta l’etichetta non controllata perché il nodo è già fissato
* Gli spazi vuoti servono per indicare che non considero più l’etichetta nelle varie iterazioni in quanto fissata
* L’algoritmo termina quando non ci sono più nodi in

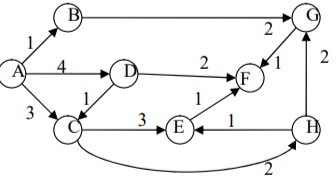
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iterazione | Nodo | Nodo | Nodo | Nodo | Nodo | Nodo | Nodo | Nodo |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  | \* |  |  |  |  |  |  |  |  |  |
|  |  | \* |  |  |  |  |  |  |  |  |
|  |  |  | \* |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  | \* |  |  |  |
|  |  |  |  |  |  | \* |  |  |  |  |
|  |  |  |  | \* |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | \* |  |  |  |  |  |

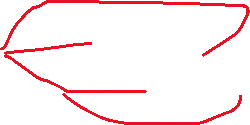
Ad ogni iterazione, percorriamo il grafo e scegliamo il percorso con costo minore. Ad ogni iterazioni, scegliamo e fissiamo un’etichetta che ha costo minore, scremando ad ogni iterazione quelle da controllare e avere sempre in mano il costo minimo. Anche in questo caso, come per gli algoritmi label correcting in caso di convergenza alla soluzione ottima, le etichette calcolate e i relativi puntatori rappresentano, rispettivamente, una soluzione duale ammissibile e i predecessori su dei cammini dall’origine ai diversi nodi. Questo funziona non avendo costi negativi.

In questo esercizio, si presentavano più etichette con lo stesso costo minimo; ho scelto in tutti i casi l’etichetta con indice inferiore.

c)

Pertanto, come abbiamo visto per gli algoritmi label correcting in caso di convergenza, è possibile derivare l’albero (risp. il grafo) dei cammini minimi attraverso i puntatori ai predecessori (risp. la verifica della saturazione dei vincoli duali sugli archi).

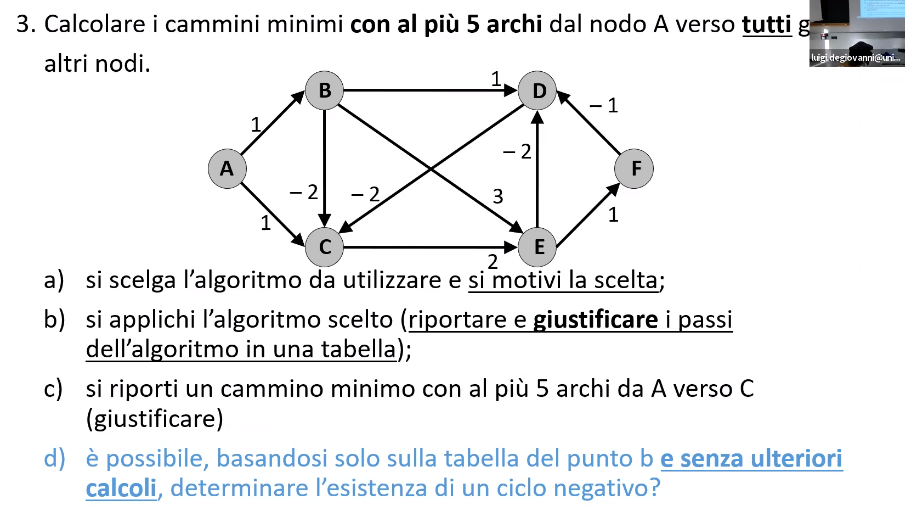


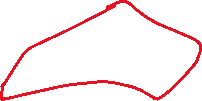


Concretamente, avremo che:

* In rosso riportiamo l’albero dei cammini minimi
* In arcobaleno riportiamo il grafo (composto come sempre da *tutti* i cammini minimi, quindi fissando le etichette ottime con lo stesso colore e scegliendo come cammino sia l’albero che tutte le altre etichette con costo ); in questo caso, grafo ed albero sono diversi.







a)

Nella scelta dell’algoritmo, notiamo che ci viene dato un massimo numero di archi da dover rispettare, pertanto si può applicare solo l’algoritmo di Bellman – Ford, l’unico che dà la possibilità di calcolo dei cammini minimi sulla base di un massimo numero di archi. Applicheremo Bellman – Ford fermandoci alla quarta iterazione, con un numero di archi e iterazioni pari a

b)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Iterazione | Nodo A | Nodo B | Nodo C | Nodo D | Nodo E | Nodo F | Aggiornamenti |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

c) Un cammino minimo (con al più 5 archi) può essere individuato seguendo la catena dei predecessori

che equivale al percorso con costo

A livello di algoritmo, si termina con *flag\_aggiornato=true*, pertanto esiste un ciclo negativo, in quanto ad ogni iterazione è stata aggiornata almeno un’etichetta.

d) Basandosi solo sulla tabella e senza ulteriori calcoli, è possibile determinare il ciclo negativo perché siamo arrivati “fino in fondo” con i calcoli.



e) È possibile disegnare albero e grafo dei cammini minimi? No, in quanto abbiamo etichette instabili (no soluzione ammissibile duale e quindi no cammini minimi).

Dimostriamo ad esempio esiste il ciclo ; come tale, partiamo dal nodo iniziale e seguiamo per i primi nodi quelli dettati dal percorso dei predecessori e poi seguiamo il percorso che vogliamo fare noi, in questo caso ciclo; come si vede, esiste un vertice che era già presente nel cammino e che, idealmente, lo migliora ad iterazioni.

Risposte con limiti di archi (al più 5 archi)

e) È possibile disegnare albero e grafo dei cammini minimi? No, in quanto abbiamo etichette instabili (no soluzione ammissibile duale e quindi no cammini minimi). Fermandosi all’iterazione 5, vedo che è stato aggiornato e, seguendo da questo la catena dei predecessori, arrivo a costruire l’albero dei cammini minimi.

Senza fare calcoli, non posso sapere se le etichette si stabilizzeranno e quindi dato che non faccio l’iterazione 6, non riesco a fare il grafo con cammini minimi.

* L'algoritmo di Dijkstra può essere utilizzato per il percorso minimo in presenza di costi negativi ma senza cicli di lunghezza negativa? GIUSTIFICARE LA RISPOSTA!

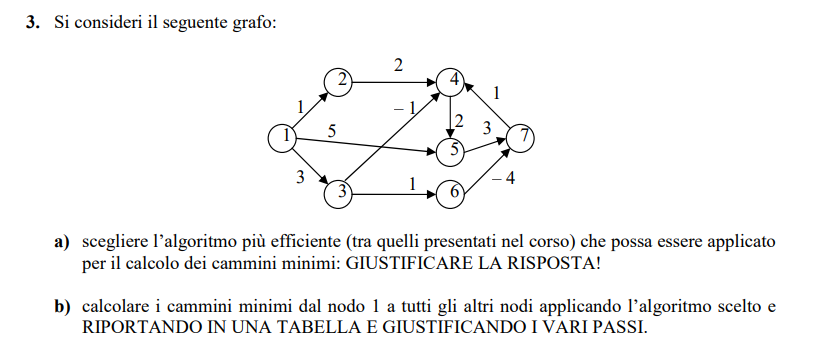
No, l'algoritmo di Dijkstra non può essere utilizzato per il cammino minimo in presenza di costi negativi perché non è stato progettato per gestire i pesi negativi dei bordi.

L'algoritmo di Dijkstra è un algoritmo di ricerca su grafo utilizzato per trovare il percorso più breve tra due nodi di un grafo. Funziona partendo da un dato nodo sorgente e rilassando ripetutamente gli spigoli del grafo per trovare la distanza minima da ogni nodo.

La caratteristica principale dell'algoritmo di Dijkstra è che sceglie sempre il percorso a distanza minima verso un dato nodo, e lo fa utilizzando una coda di priorità per memorizzare i nodi che sono stati visitati e le distanze da essi. Ciò garantisce che l'algoritmo scelga sempre il percorso a distanza minima per ogni nodo, poiché seleziona sempre per primo il nodo con la distanza minima dalla coda di priorità.

Tuttavia, questo approccio non funziona quando nel grafo sono presenti pesi negativi sui bordi. Infatti, l'algoritmo sceglierà sempre il percorso a distanza minima, anche se questo comporta l'attraversamento di un bordo negativo. Questo può far sì che l'algoritmo trovi un percorso che ha una distanza complessiva inferiore a quella del vero percorso a distanza minima, il che non è quello che vogliamo.

Pertanto, l'algoritmo di Dijkstra non può essere utilizzato per il percorso minimo in presenza di costi negativi.



a) Nella scelta dell’algoritmo, notiamo che ci viene dato un massimo numero di archi da dover rispettare, pertanto si può applicare solo l’algoritmo di Bellman – Ford, l’unico che dà la possibilità di calcolo dei cammini minimi con degli archi di costo negativo. Applicheremo Bellman – Ford fermandoci alla settima iterazione, dunque esaminando ciclicamente tutti gli archi.

b)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iterazione | Nodo 1 | Nodo 2 | Nodo 3 | Nodo 4 | Nodo 5 | Nodo 6 | Nodo 7 | Aggiornamenti |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

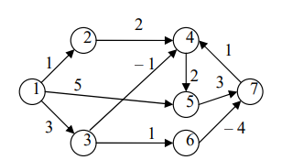
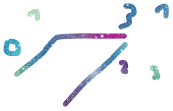


Applicazione dell’algoritmo: si utilizza una tabella che riporta una riga per ogni iterazione dell’algoritmo. Ogni colonna della tabella è dedicata ad un nodo e riporta, iterazione dopo iterazione, l’evoluzione delle rispettive etichette. L’ultima colonna riporta i nodi aggiornati nel corso dell’iterazione: all’iterazione successiva è sufficiente controllare solo gli archi uscenti da questi nodi.

Le etichette di una riga sono ottenute controllando i vincoli duali degli archi uscenti dei nodi aggiornati rispetto alla riga precedente. Non abbiamo un limite di max hop, pertanto consideriamo la lunghezza del cammino minimo alla h-esima iterazione.

Questo viene individuato seguendo la catena dei predecessori a partire dal nodo 7 con e considerando la riga precedente, partendo dal nodonell’iterazione con , essendo l’ultimo aggiornato, e procedendo a ritroso per ritrovare i predecessori.

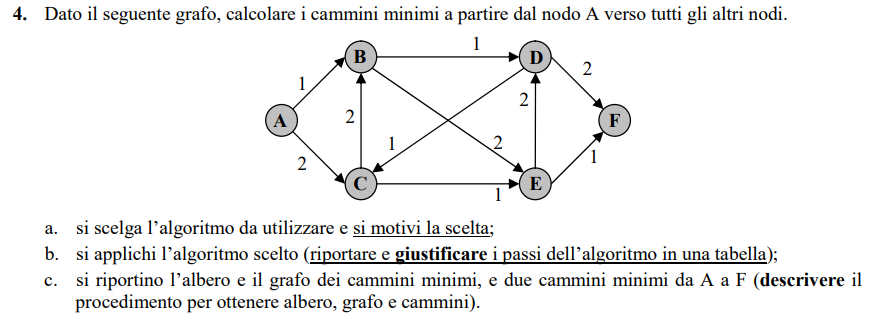
Si ottiene, come per altro disegnato, un cammino con 5 archi che ha , con costo 3.

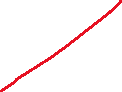




Legenda:

Albero 🡪 Rosso (seguo i predecessori)  
Grafo 🡪 A parità di archi e costi, considero quelli di costo minore/uguale al percorso che ho già





a) Il miglior algoritmo in questo contesto è Dijkstra, in quanto più efficiente e usabile in quanto tutti i costi ridotti sono positivi.

b) Ora, il calcolo dei cammini minimi, riportati in tabella. Si ricorda che:

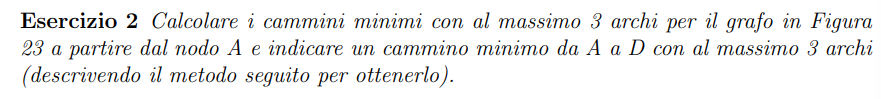
* rappresenta l’etichetta minima di ogni iterazione
* rappresentano le etichette ancora da fissare
* Il segno \* rappresenta l’etichetta fissata
* Il segno – rappresenta l’etichetta controllata ma non aggiornata
* Il segno x rappresenta l’etichetta non controllata perché il nodo è già fissato
* Gli spazi vuoti servono per indicare che non considero più l’etichetta nelle varie iterazioni in quanto fissata
* L’algoritmo termina quando non ci sono più nodi in

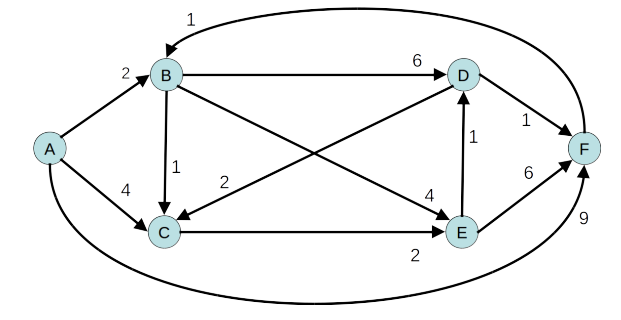
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Iterazione | Nodo | Nodo | Nodo | Nodo | Nodo | Nodo |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  | \* |  |  |  |  |  |  |
|  |  |  |  | \* |  |  |  |  |
|  |  |  | \* |  |  |  |  |  |
|  |  |  |  |  | \* |  |  |  |
|  |  |  |  |  |  | \* |  |  |

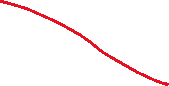
Ad ogni iterazione, percorriamo il grafo e scegliamo il percorso con costo minore. Ad ogni iterazione, scegliamo e fissiamo un’etichetta che ha costo minore, scremando ad ogni iterazione quelle da controllare e avere sempre in mano il costo minimo. Anche in questo caso, come per gli algoritmi label correcting in caso di convergenza alla soluzione ottima, le etichette calcolate e i relativi puntatori rappresentano, rispettivamente, una soluzione duale ammissibile e i predecessori su dei cammini dall’origine ai diversi nodi. Questo funziona non avendo costi negativi.

c) Come abbiamo visto per gli algoritmi label correcting in caso di convergenza, è possibile derivare l’albero (risp. il grafo) dei cammini minimi attraverso i puntatori ai predecessori (risp. la verifica della saturazione dei vincoli duali sugli archi).

Contrariamente a quanto chiede la domanda, non esistono altri cammini minimi, seguendo tutti i possibili percorsi. Albero e grafo coincidono.







Essendo che abbiamo al massimo 3 archi, possiamo usare solo Bellman-Ford.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Iterazione | Nodo A | Nodo B | Nodo C | Nodo D | Nodo E | Nodo F | Aggiornamenti |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

Non possiamo parlare di albero dei cammini minimi qualora si parla di massimo numero di archi. Possiamo però identificare due cammini minimi:

