Illustrare il problema di Albero ricoprente di peso minimo, dimostrare le proprietà di una soluzione ottima e illustrare due algoritmi noti per risolvere il problema che si basano su queste proprietà.

Un **albero ricoprente** di un grafo connesso è un albero che collega tutti i nodi del grafo.

Il problema dell’Albero ricoprente a costo minimo consiste nell’individuare un albero ricoprente tale che la somma dei pesi degli archi appartenenti all’albero sia minimizzata.

In tali problemi, è possibile formulare 2 condizioni di ottimalità:

* Condizioni di ottimalità sui tagli (Su cui si basa l’algoritmo di Prim-Dijkstra)
* Condizione di ottimalità sui cammini (Su cui si basa l’algoritmo di Kruskal)

**Condizioni di ottimalità sui tagli**

**Teorema:** Sia *T\** un albero ricoprente di costo minimo, un arco *e* apparterrà a *T\** se e solo se esiste un taglio fondamentale in *G* tale che l’arco *e* minimizzi il costo degli archi tagliati.

**Dimostrazione Sufficienza:** Supponiamo per assurdo che esista un arco *e* che minimizza un taglio, ma che tale arco non appartenga all’albero ricoprente di costo minimo. Aggiungendo all’albero ricoprente l’arco *e* si otterrà un ciclo fondamentale, e ci sarà almeno un altro arco che appartiene al taglio, sia questo arco *f*. Varrà la condizione *c*f *> c*e, quindi rimuovendo *f* dall’albero di costo minimo si avrebbe un albero a costo inferiore segue l’assurdo.

**Dimostrazione Necessità:** Dato *T\** dobbiamo mostrare come sia possibile costruire un taglio tale che l’arco *e* sia l’arco di costo minimo tra gli archi appartenenti al taglio. A tal fine consideriamo nell’insieme *S* una delle due componenti connesse che si verrebbe a generare a seguito della rimozione dell’arco *e*. L’arco *e* quindi appartiene ad un taglio e necessariamente sarà il minimo in quel taglio altrimenti l’albero *T\** non sarebbe una soluzione ottima.

**Condizione di ottimalità sui cammini**

**Teorema:** Un albero ricoprente *T\* è* minimo se e solo se soddisfa *ce* <= *cf* per ogni arco *nontree f* di *G* e per ogni arco *e* contenuto nel cammino che connette i due nodi terminali di *f*.

**Dimostrazione Necessità:** Supponiamo per assurdo che *T\** sia ottimo e che esista un arco *e* nel cammino tra i nodi terminali dell’arco *f* che non soddisfi la condizione. Se vale *ce* >= *cf* allora introdurre l’arco *f* nell’albero ricoprente al posto dell’arco *e* produrrebbe un albero a costo inferiore di *T\**, contraddicendo l’ipotesi iniziale.

**Dimostrazione Sufficienza:** Sia *e* un *tree arc* e siano *S* e *S* generati dal taglio legato alla rimozione di *e*, sia l’arco *f* nel taglio [*S, V* − *S*]. Visto che *T\** contiene un unico cammino che unisce le estremità di *f*, tale cammino dovrà passare per l’arco *e*. L’ipotesi implica *ce* <= *cf* . Considerando che questa condizione deve essere valida per ogni *nontree* arc nel taglio [*S, S*], allora *T\** soddisfa le condizioni di ottimalità dei tagli da cui *T\** deve essere un albero a costo minimo.

🡪Algoritmo di Kruskal:

0) L’algoritmo parte con un albero ricoprente vuoto

1) Si ordinano gli archi in ordine crescente di peso in una lista

2) Si scorre la lista *prendendo* ad ogni passo l’arco con peso minore e aggiungendolo a T se non genera un ciclo.

3) Alla fine si ottiene un albero ricoprente a costo minimo.

🡪Algoritmo di Prim-Dijkstra

0) Albero ricoprente vuoto

Ad ogni iterazione:

1) Si considera un taglio fondamentale differente

2) Si cerca l’arco che minimizza il peso tra gli archi appartenenti

al taglio corrente

3) Si aggiorna il taglio corrente con l’arco selezionato

4) Si inserisce l’arco selezionato nell’albero ricoprente

L’iterazione viene ripetuta finché non si è costruito l’intero albero.

Definire il problema di cammino minimo, illustrare un algoritmo noto per risolverlo nel caso in cui

siano presenti archi di peso negativo e dimostrare il teorema di Floyd-Warshall

Il problema di cammino minimo in un **grafo orientato e pesato** consiste del trovare un camminotale che sia minimizzato il suo costo. Esistono diverse tipologie di cammino minimo:

* Cammino minimo tra due nodi;
* Cammini minimi a partire da un nodo;
* Cammino minimo tra tutte le coppie di nodi.

Per risolvere il problema di cammino minimo può essere usato l’algoritmo di Floyd–Warshall il quale può essere applicato anche a grafi che abbiano **pesi negativi sugli archi**, ammesso sempre che non esistano cicli a peso negativo. Infatti in presenza di cicli a peso negativo nel grafo l’algoritmo di Floyd–Warshall individua uno dei cicli e termina l’esecuzione senza aver risolto il problema.

L’algoritmo si basa sul seguente **teorema:**

Si consideri la matrice dei costi *C* ottenuta dall’esecuzione dell’operazione triangolare sul nodo *k.*

Per ogni *i, j* in *V,* il valore *c*[*i, j*] rappresenta il costo del cammino minimo da *i* a *j* nel sottografo indotto dall’insieme di vertici {1*, …, k*} U {*i, j*}.

**operazione triangolare:** per ogni coppia di nodi *i, j* si controlla se per andare da *i* a *j* conviene passare per il nodo *k* ovvero: Se (*c*[*i, k*] + *c*[*k, j*] < *c*[*i, j*]) allora *c*[*i, j*] = *c*[*i, k*] + *c*[*k, j*]

**Dimostrazione:**

Si procede per induzione sul nodo *k*

**Passo base dell’induzione:**

Per *k* = 0, la matrice *C* coincide con la matrice iniziale dei costi degli archi, e quindi *c*[*i, j*] rappresenta effettivamente il costo del cammino minimo da *i* a *j* nel sottografo indotto da {*i, j*}

**Passo induttivo:**

Supponiamo che la proprietà sia verificata all’iterazione *k* -1.

Si consideri un cammino minimo *Pij* da *i* a *j* nel sottografo indotto da {*1, …, k*} U {*i, j*} e sia *c*(*Pij*) il suo costo A questo punto si possono verificare due casi:

1. *Pij* non passa dal vertice *k*, allora *c*(*Pij*) = *c*[*i, j*] per l’ipotesi induttiva;
2. *Pij* passa dal vertice *k*, allora *Pij = Pik* U *Pkj.*

*Pik* (*Pkj*) = cammino di costo minimo da *i* a *k* (da *k* a *j*) nel sottografo indotto da {*1, …, h-1*} U {*i,k*} ({*k, j*})

Per l’ipotesi induttiva si ha che *c*(*Pik*) = *c*[*i, k*] e *c*(*Pkj*) = *c*[*k, j*] da cui *c*(*Pij*) = *c*[*i, k*] + *c*[*k, j*]

Unendo i due casi si ha: *c*(*Pij*) = min {*c*[*i, j*], *c*[*i, k*] + *c*[*k, j*]} ossia dopo l’operazione triangolare su *k* si dimostra *c*[*i, j*] = *c*(*Pij*).

Definire il problema di Massimo Flusso, illustrare un algoritmo noto per risolverlo e dimostrare il

teorema di Ford-Fulkerson.

Il problema del massimo flusso consiste nell’individuare un flusso da *s* (nodo sorgente)a *t* (nodo pozzo) nella rete tale che tutte le capacità sugli archi non siano violate e che tale flusso totale sia massimizzato. Gli archi (*i, j*) nella rete hanno una *capacità* non negativa *kij* >= 0

A differenza di altri problemi visti le etichette associate agli archi sono delle *capacità massime* che non possono essere superate e non delle distanze o dei costi che devono essere minimizzati.

Ci sono 2 tipi di vincoli che devono essere rispettati:

1. i vincoli sulla *capacità del flusso* che scorre negli archi;
2. i vincoli sul *bilanciamento dei flussi* entranti e uscenti nei/dai nodi.

Prima categoria di vincoli:

in ogni soluzione ammissibile, il flusso (eventualmente nullo) *xij* che scorre nell’ arco (*i, j*) non potrà mai superare il massimo flusso che può attraversare l’arco *kij*, ovvero non potrà mai eccedere la sua capacità.

Seconda categoria di vincoli:

Questi vincoli implicano che la somma dei flussi sugli archi entranti ad ogni nodo (tranne *s* e *t*) deve essere uguale alla somma dei flussi sugli archi uscenti ovvero in ogni nodo non si può ne generare ne perdere del flusso:

Un flusso si dice *ammissibile* se per ogni arco e per ogni nodo sono rispettati i 2 tipi di vincoli.

Un algoritmo per risolvere il problema di massimo flusso è quello di Ford-Fulkerson. Ad ogni iterazione bisogna cercare un cammino aumentante ovvero un cammino da S a P in cui gli archi diretti non sono saturi e gli archi inversi non sono scarichi.

• Arco diretto: (i,j) con capacità residua Kij - Xij.

• Arco inverso: (i,j) con capacità residua Xij.

Il massimo flusso δ che può scorrere nel cammino aumentante sarà dato dalla più piccola capacità tra gli archi attraversati, ovvero δ = min{kij : (i, j) in P} > 0.

* Attraversare un arco diretto con un flusso δ comporta nel grafo originale G l’incremento del flusso su tale arco (xij = xij + δ).
* Attraversare un arco inverso con un flusso δ comporta nel grafo originale G la diminuzione del flusso su tale arco (xij = xij − δ).

Se non esiste un cammino aumentante, esiste un taglio che separa la sorgente dal pozzo.

**Teorema:** Un flusso ammissibile x è il massimo flusso in una rete G se e solo se non esiste un cammino aumentante tra il nodo sorgente s e il nodo pozzo t nel grafo G di G.

**Dimostrazione:** Sia φ il valore del flusso in G. Se t è raggiungibile da s in G allora esiste un cammino aumentante P, sia δ = min{kij : (i, j) in P} > 0 il massimo valore del flusso aumentante.

Allora è possibile ottenere un flusso in G di valore φ + δ semplicemente aumentando il flusso sugli archi diretti xij + δ e diminuendo quello sugli archi inversi xij – δ.

Supponiamo che non esista un cammino aumentante in G, allora deve esistere un taglio [S, S] nella rete incrementale con flusso nullo, φ(S) = 0. Nella rete originale G allora tale taglio sarà composto solo da archi saturi (gli archi uscenti dal taglio) e scarichi (gli archi entranti nel taglio). Ne consegue che per tale taglio vale:

Dato che φ(S) <= k[S, S] si ha che tale flusso è ottimo.

Definire il problema di Flusso di costo minimo, illustrare un algoritmo noto per risolverlo e

dimostrare che una base della matrice dei coefficienti del problema in forma standard corrisponde a

un albero ricoprente della rete di flusso.

Il problema di flusso di costo minimo consiste nel far giungere il “prodotto” realizzato nei nodi sorgente ai nodi destinazione facendolo viaggiare attraverso la rete e cercando di spendere il meno possibile per il trasporto. Ciò che viene prodotto nei nodi sorgente è esattamente pari a ciò che viene consumato nei nodi destinazione.

Un algoritmo noto per risolverlo è il simplesso su reti. Imposto il duale:

Trovo u risolvendo: uj-ui = Cij, per ogni (i,j) in base e fissando u1 = 0.

Verifico l’ammissibilità duale di u: uj-ui <= Cij per ogni (i,j) fuori base.

Se uj-ui > Cij allora xij (flusso nell’arco (i,j)) entra in base e esce l’arco discorde con il minimo flusso presente nel ciclo formato da xij entrante. Se nel ciclo non è presente nessun arco discorde il problema è illimitato.

Fase 1 (problema artificiale)

* Aggiungo un nodo artificiale
* Archi artificiali entranti nei pozzi uscenti dagli archi
* Base = archi artificiali + archi originali per avere un albero ricoprente
* Costi = 1 archi artificiali, 0 archi originali
* Quando tutti gli archi artificiali hanno flusso = 0 mi posso fermare e passare alla fase 2
* Trovo soluzione ottima per problema artificiale 🡪 Soluzione ammissibile per primale

Fase 2

* Elimino nodo artificiale e i suoi archi
* Se rimane albero ricoprente continuo
* Se non rimane albero ricoprente lo ricostruisco con archi originali di costo 0

**Dimostrazione:**

1)Se gli (n-1) archi formano un albero ricoprente 🡪 Formano Base

Visito l’albero numerando i nodi da 0 a n-1 e gli archi da 1 a n nell’ordine in cui vengono visitati. Se ordiniamo le n-1 colonne e le n righe della matrice A associata all’abero, si ottiene una matrice (n-1)x n. Se scartiamo la prima riga della matrice, resta una sottomatrice di A che per costruzione è triangolare superiore ed ha tutti gli elementi sulla diagonale principale diversi da 0. Questo implica che la matrice A ha determinante diverso da 0 e quindi che le colonne associate ad ogni albero ricoprente sono sempre linearmente indipendenti fra loro. Questo dimostra che le colonne di A associate ad un albero ricoprente sono sempre basi di A.

2) Se gli (n-1) archi non formano un albero ricoprente (ovvero formano un ciclo)🡪Non formano una base

E’ sufficiente dimostrare che le colonne di A associate ad un ciclo sono linearmente dipendenti.

I coefficienti della combinazione lineare possono ottenersi semplicemente percorrendo il ciclo in un verso qualsiasi e fissando il coefficiente dell’arco (i,j) del ciclo:

• 1 se l’arco è concorde al verso di percorrenza del ciclo;

• -1 se l’arco è discorde al verso di percorrenza.

Sommando le colonne si ottiene una colonna nulla, il che dimostra che queste tre colonne sono linearmente dipendenti e quindi non possono far parte contemporaneamente di una base di A.