**Grafo Bipartito**

Un grafo bipartito è un grafo non orientato in cui si ha un insieme di vertici partizionato, ossia che non si intersecano e che coprono l'intero insieme di vertici in più sottoinsiemi, chiamati ad esempio A e B, e ho che gli archi possono andare soltanto tra A e B, quindi possono solo congiungere un nodo di A o un nodo di B. Potrei ad esempio colorare tutti i nodi di A di blu e tutti i nodi di B di rosso. A questo punto posso dire che il grafo è bipartito se per ogni arco, esso ha un’estremità blu e una rossa; in altre parole, un arco non collega due nodi dello stesso colore. Esisterà almeno una colorazione rosso-blu in modo tale da avere ogni arco che collega solo nodi di colore differente.

Un problema interessante: Dato un grafo non direzionato, vogliamo scoprire se è bipartito. Perché ci è utile? Perché molti problemi sui grafi diventano [molto più facili]/[trattabili] se il grafo è bipartito.

Esempio di grafo bipartito: Agenzia matrimoniale. I collegamenti sono solo tra uomini e donne, non tra uomini e uomini oppure donne e donne. Ipotizziamo che questa agenzia guadagni ogni volta che si faccia un matrimonio. Di conseguenza si cerca di massimizzare il numero di matrimoni. Un matrimonio si può fare solo se c'è l'arco, e gli archi selezionabili sono particolari: di certo una donna non si può sposare contemporaneamente con più uomini! L'insieme di archi che non hanno in comune estremità si chiama matching. Un qualsiasi matching fornisce un insieme di possibili matrimoni, più grande è l'insieme più alto è il guadagno, per cui ho ragione di voler trovare il matching più grande possibile. Nella sua associazione più seria può essere visto come un insieme di job e processore: un job è connesso ad un processore se può essere eseguito da quel processore, assumendo che un processore possa eseguire un singolo job. Più grande è l'insieme di archi selezionato, più alto è il numero di job che eseguirò.

**Se un grafo G è bipartito, esso non può contenere un ciclo dispari (ossia un ciclo che contiene un numero dispari di archi). Se il grafo non è bipartito, quindi, siamo in grado di scoprire un ciclo dispari, che è a sua volta la prova che il grafo non sia bipartito.**

Facciamo una **dimostrazione** con gli esempi (pagina 11, slide 55).

**Dimostrazione per contraddizione**: Immaginiamo di avere un ciclo dispari fatto da un qualsiasi numero dispari di archi e di voler dimostrare che sia un grafo bipartito. Comincio a colorare i vertici. Scelgo arbitrariamente un nodo da cui partire ed inizio a colorare i nodi in colori alternati: rosso-blu-rosso-blu-... Arrivo ad un nodo che non so di quale colore colorare, perché il nodo precedente è di colore rosso, ad esempio, ma il nodo successivo è di colore blu. Per questo, non sono in grado di trovare una colorazione che eviti questo problema, se il grafo bipartito è un ciclo dispari.

Nota: Alcuni studenti dicono che se un grafo bipartito non contiene un ciclo dispari allora è automaticamente bipartito, ma questa è una cosa molto più difficile da dimostrare. Se vogliamo utilizzare l'assunzione dobbiamo anche dimostrarla. Altrimenti dobbiamo usare solo che se un grafo ha un ciclo dispari allora non è bipartito.

Un'altra proprietà che utilizzeremo per scoprire se un grafo è bipartito è:

**Dato un arco (x,y) in un grafo non orientato, si ha che x e y o si trovano nello stesso livello dell'albero BFS, o si trovano in livello [adiacenti]/[contigui] dell'albero BFS. In pratica se x sta al livello i, y si troverà o al livello i-1, o al livello i, o al livello i+1.**

**Dimostrazione:**

Indichiamo con l\_i il livello che contiene x, ed l\_j il livello che contiene y. Senza perdere di generalità dirò che x verrà scoperto prima di y, il che vuol dire che il l\_i precede il l\_j. Quindi i<=j. Consideriamo il momento in cui l'algoritmo esamina gli archi incidenti su x.

Si possono verificare due casi:

* **y è stato già scoperto:**

Quando io ho scoperto x, y non era stato scoperto, per ipotesi. Quel che accade è che quando vado a scandire la lista dei nodi adiacenti ad x, y è stato già scoperto, e vuol dire che y è stato già inserito in un livello. Il che vuol dire che, siccome y non era già stato scoperto in precedenza, non è nei livelli l\_0, l\_1, ..., l\_(i-1).

Ci sono quindi due possibilità:

* + **y si trova anche lui nel l\_i, se era adiacente a qualche altro nodo del l\_i**.
  + **y si trova nel l\_(i+1), perché è adiacente a qualche nodo che precede x all'interno del livello.**
* **y non è stato già scoperto:**

A quel punto, quando vado a scandire gli archi incidenti su x, troverò l'arco (x,y) il quale mi porta ad y che avrà ancora il discovered a false. Questo significa che y verrà messo nel livello successivo l\_(i+1).

**Quindi, dato un arco (x,y), o x e y stanno nello stesso livello, o in livelli vicini.**

Sia quindi (x,y) un arco dell'albero BFS creato, la distanza tra x ed y sarà 1 perché x ed y potranno essere al massimo in livelli consecutivi.

Dato un grafo G, immaginando di fare la visita BFS sul grafo, si possono verificare due cose:

**1)Nessun arco di G collega nodi sullo stesso livello**

**2)Oppure esiste un arco di G che collega nodi sullo stesso livello.**

Una delle due deve essere vera.

**Caso 1)**

Possiamo vedere che se vale la 1 allora il grafo è bipartito. Abbiamo visto che tutti gli archi collegano o nodi sullo stesso livello o nodi consecutivi. In questo caso non ci sono archi che collegano nodi sullo stesso livello dell'albero. Allora, per quel lemma, gli archi devono collegare soltanto nodi a distanza esattamente 1 nell'albero (consecutivi). Quindi coloriamo tutti i nodi dei livelli pari di blu e tutti i nodi dei livelli dispari di rosso. Avendo escluso la presenza di archi tra nodi dello stesso livello, gli archi congiungeranno solo nodi di livelli consecutivi. Ora siccome due livelli consecutivi hanno colori differenti, il grafo è bipartito.

**Caso 2)**

Esiste un arco tra nodi dello stesso livello. Chiamiamo (x,y) questo arco. Dimostriamo che esiste un ciclo dispari, formato dall'arco (x,y) sul livello l\_j e dai percorsi che vanno da x e y fino al loro antenato comune più vicino, ossia un nodo antenato sia di x che si y e si trova quanto più vicino possibile ad x ed y. Siccome x ed y sono sullo stesso livello è ovvio che la distanza tra x e questo antenato ed y e questo antenato sia la stessa. Ovviamente la radice è antenato comune di tutti i nodi, però sotto la radice ci potrebbe essere un nodo antenato sia di x che di y, ma che si trova più in basso. Noi andiamo a prendere quello che si trova più in basso in questo l\_i e lo chiameremo z. Al di sotto di z non c'è nessun nodo antenato sia di x che di y. (Pagina 13 slide 59). Esiste un percorso da x a z e da y a z. Prendo un percorso da x a z e un percorso da y a z, il percorso da x a z ha lunghezza j - i, così come il percorso da y a z. Quindi ho (j - i) + (j - i) = 2(j - i). Per chiudere il cerchio devo aggiungere l'arco che collega (x,y). In totale avrò che il percorso avrà lunghezza 2(j - i) + 1. Un numero pari più un numero dispari è un numero dispari, quindi questo è un ciclo dispari e automaticamente non è un grafo bipartito.

Quindi, se non esiste un arco tra nodi dello stesso livello, il grafo è bipartito.  
Quali sono le condizioni che testo?

**1) Esiste un arco tra nodi dello stesso livello**

**2) Non esiste un arco tra nodi dello stesso livello.**

Algoritmicamente, come possiamo fare per testare queste due condizioni? Faccio la visita BFS e coloro i nodi dei livelli pari di un colore e i nodi di livelli dispari di un altro colore. Fatto questo, scandiamo uno alla volta gli archi del grafo. Se troviamo un arco con l'estremità dello stesso colore, allora il grafo non è bipartito. Se invece nessuno degli archi congiunge due nodi dello stesso colore, il grafo è bipartito. Possiamo dire questo per quel lemma trattato: se esiste un arco con due nodi dello stesso colore, l'unica possibilità è che i due nodi sono sullo stesso livello, perché dato un arco, i due estremi o sono sullo stesso livello o su livello consecutivi, ma se sono dello stesso colore sono sullo stesso livello, perché se fossero stati su livelli consecutivi avrebbero avuto colori differenti. Se invece nessuno degli archi congiunge due nodi dello stesso colore vuol dire che ogni arco collega nodi di livelli consecutivi. Facendo questo test io scopro in quale di questi due casi mi trovo.

L'idea:

Facciamo una BFS, man mano che inseriamo i nodi nei livelli li coloriamo, i livelli pari di un certo colore e i livelli dispari di un certo altro colore. Alla fine, dopo aver eseguito la BFS modificata in questo modo andiamo a scandire tutti gli archi. Se ne troviamo uno con estremità uguali concludiamo che uno non è bipartito. Come facciamo a colorare i nodi? la cosa più semplice è usare un array A di colori che assocerà ad ogni cella un vertice, ed ogni volta che aggiungiamo il vertice u al livello i, se i è pari poniamo A[i]=rosso. se i è dispari poniamo A[i]=blu. La complessità resta la stessa.

BFS in cui coloro i nodi a seconda del livello in cui mi trovo

se trovo archi con gli estremi dello stesso colore

grafo non bipartito

altrimenti

grafo bipartito

Tempo O(n+m), ossia il tempo della BFS, perché fatta la BFS, la colorazione prende tempo O(n), la scansione degli archi O(m).

Nota: Posso modificare la BFS/DFS in modo che funzionino anche con grafi direzionati. Quando vado ad esplorare gli archi incidenti su un nodo vado ad esplorare solo gli uscenti da quel nodo. Ipotizziamo di avere due liste di adiacenza per ogni nodo. La lista dei nodi v per i quali esiste l'arco (u,v) e la lista di adiacenza dei nodi w per i quali esiste l'arco (w,u).

Il concetto di grafo connesso è collegato al concetto di grafo non direzionato perché in quest'ultimo, parlare di un percorso da u a v è lo stesso di parlare di un percorso da v a u. Per quanto riguarda i grafi direzionati invece questo non è vero. Ci potrebbe essere un percorso che va da u a v ma non un percorso che va da v a u.

**Connettività forte**

Nel caso di un grafo direzionato si parla di grafo fortemente connesso se per ogni coppia di nodi u,v del grafo, esiste sia il percorso che va da u a v che da v ad u. Quindi, se parlo di grafo connesso mi sto riferendo ad un grafo non direzionato, perché lì la direzione dei percorsi non c'è, mentre se parlo di grafi direzionati parlo di connessione forte, e vale se e solo se per ogni coppia di nodi esiste un percorso in entrambe le direzioni.

In un grafo G direzionato, G è fortemente connesso se e solo se, preso un qualsiasi nodo del grafo s, qualsiasi nodo u è raggiungibile da s e da quel nodo u è possibile raggiungere s.

Dimostrazione. Per vedere che questa cosa vale prendiamo due vertici qualsiasi u e v del grafo e devo verificare il lemma. Come? Devo supporre che l'asserto del lemma valga e dimostrare che dati due qualsiasi nodi u e v, posso andare da u a v, e da v ad u. L'asserto dice che preso s, posso andare da s ad u, da u ad s, da s ad v, da v ad s.

Siccome io posso andare da u ad s, e da s ad v, posso andare da u ad v.

Siccome io posso andare da v ad s, e da s ad u, posso andare da v ad u.

Quindi vale che presi due qualsiasi nodi u e v io possa andare da u a v e da v ad u.

**1) I nodi u e v sono mutualmente raggiungibili se c’è un percorso da u a v e anche un percorso da v a u.**

**2) Un grafo è fortemente connesso se ogni coppia di nodi è mutualmente raggiungibile**

**Dim 1)**

Se G è fortemente connesso allora posso andare da s ad u e da u ad s per ogni vertice u diverso da s. Il lemma afferma anche che vale l'implicazione contraria, cioè che io posso andare da qualsiasi altro vertice ad s.

**Dim 2)**

Per dimostrare la connettività forte mi basta prendere un vertice e provare che da questo vertice s io posso raggiungere qualsiasi altro vertice e viceversa. Ma per scoprire che da questo vertice s io posso raggiungere qualsiasi altro vertice, cosa posso fare? DFS/BFS. Alla fine se tutti i nodi avranno il discovered a true vorrà dire che li ho raggiunti tutti. Questo però non basta perché per provare la connettività forte devo dimostrare che da qualsiasi altro vertice io posso raggiungere s. Potrei invocare la visita BFS/DFS su ogni altro vertice, ma questo mi porterebbe via moltissimo tempo, allora decido di fare una cosa più furba: costruisco un grafo che ha gli stessi vertici del grafo g, però gli archi sono invertiti: se nel grafo originario G c'è l'arco (a,b), nel nuovo grafo G' ci sarà l'arco (b,a). A questo punto visito nuovamente con BFS/DFS avendo come sorgente s. Se raggiungo tutti i nodi vuol dire che da ciascun nodo c'è un percorso che porta ad s. Perché nell'albero BFS/DFS G' un percorso che va da s ad u corrisponde nel grafo G di partenza in un percorso che va da u ad s. Quindi se nel grafo G' esiste un percorso da s ad u vuol dire che esiste un percorso in direzione opposta nel grafo di partenza. Se i campi discovered saranno tutti uguali a true, il grafo è fortemente connesso.

**Grafi direzionati aciclici (DAG)**

Ora parliamo di grafi molto importanti per rappresentare relazioni di precedenza tra attività, per esempio propedeudicità tra gli esami, o qualsiasi altro insieme di attività che deve essere svolto in un certo ordine. Un DAG è un grafo direzionato aciclico.

Il fatto che sia aciclico significa che non ci sono cicli. Un ipotetico arco (a,b) fa pensare al fatto che a debba essere svolto prima di b.

Perché non dobbiamo avere i cicli? Se avessimo un ciclo che parte da A e torna ad A significherebbe che io devo svolgere A prima di qualsiasi altra attività, ma A si trova anche alla fine del ciclo, e questo non ha senso.

Un modo facile per vedere se un determinato grafo è un DAG è disporre il grafo stesso in un ordinamento particolare. Grazie a quello posso dedurre immediatamente che non ci sono cicli: tutti gli archi vanno in avanti, collegano un nodo della sequenza ad un nodo che è più avanti nella stessa sequenza. Questo tipo di ordinamento si chiama ordinamento topologico. I nodi sono ordinati in modo tale che se nel DAG esiste l'arco (a,b), a procede b nell'ordinamento topologico. In maniera più raffinata: un ordinamento topologico è un'etichettatura (ribattezzo i vertici) dei nodi del grafo fatta tramite interi successivi in modo tale che ogni arco (v\_i,v\_j) si ha che i < j.

**Ora dimostriamo che se un grafo direzionato G ha un ordinamento topologico, allora è un DAG.**

Io sto dicendo, quindi, che posso definire gli ordinamenti topologici solo per i DAG.

**Dimostrazione per assurdo:**

Supponiamo che G sia un grafo direzionato e che abbia un ordinamento topologico sui nodi v\_1, v\_2, ..., v\_n. Supponiamo quindi che G non sia un DAG.  
Allora, se G non è un DAG significa che c'è un ciclo C. Supponiamo che il nodo di indice più piccolo in questo ciclo sia v\_i. Ora, consideriamo il nodo che precede v\_i nel ciclo lo chiamiamo v\_j. Siccome esiste l'arco che va da v\_j a v\_i, allora v\_j deve precedere v\_i nell'ordinamento topologico, e quindi j < i. Siccome ho detto che v\_i è il nodo con indice più piccolo nell'ordinamento topologico nel ciclo C, deve essere i < j. Ciò è assurdo.

Ora ci chiediamo: "Abbiamo visto che se esiste l'ordinamento, allora il grafo è un DAG, ma è vero il viceversa?"

**Il nostro scopo finale sarà quello di trovare l'ordinamento topologico di un DAG, ma dobbiamo essere certi che esso esista.**

**Dimostriamolo:**

Per dimostrarlo, dimostriamo un risultato preliminare: Se G è un DAG, allora esiste un nodo senza archi entranti. Di conseguenza se ogni nodo di un grafo ha almeno un arco entrante, non si tratta di un DAG. Supponiamo per assurdo che ogni nodo abbia un arco entrante, allora mi metto in qualsiasi nodo u, che sicuramente avrà almeno un arco entrante. Decido di seguire uno degli archi (w,u) a ritroso, e mi ritroverò in w. Per ipotesi anche w avrà un arco entrante (z,w). Sempre per ipotesi lo avrà anche z, ma dopo n volte che ho fatto questa cosa io ho percorso n archi, attraversando n + 1 nodi. Ma i nodi di un grafo sono n e ciò vuol dire che avrò percorso un nodo almeno due volte, quindi c'è un ciclo.

Ora andiamo a **dimostrare** finalmente che **se G è un DAG, avrà un ordinamento topologico**. È una dimostrazione per induzione sul numero di nodi che compongono il grafo. Base dell'induzione: ho un solo nodo, n = 1. Se n = 1 l'ordinamento topologico è banalmente formato da quel nodo. Ipotesi induttiva: Supponiamo che l'asserto sia vero fino ad n + 1, cioè che ogni DAG composto da n + 1 nodi abbia un ordinamento topologico. Ho un grafo con n nodi. Tolgo un nodo v senza archi entranti (e ne abbiamo dimostrato l'esistenza) dal grafo e tutti gli archi uscenti che incidono su di esso, ritrovandomi con un grafo G-{v} di n nodi. L'ipotesi induttiva mi dice che esiste l'ordinamento topologico per n - 1 nodi. Prendo quindi i nodi ordinati per ordinamento topologico e piazzo davanti a tutti il nodo v che avevo tolto inizialmente. Ciò che ottengo è un ordinamento topologico: I nodi del grafo G-{v} sono ben ordinati tra di loro per ipotesi induttiva. Mettendo v davanti, prima di tutti gli altri nodi io non vado a violare l'ordinamento topologico, perché v non ha archi entranti e l'ordinamento topologico stabilisce che se ho l'arco (v,a), io visito v prima di a. Siccome io ho messo il mio nodo v prima di tutti quanti gli altri e tutti gli archi di v sono solo archi uscenti, v sta bene all'inizio dell'ordinamento topologico.

A partire da questa dimostrazione possiamo scrivere una procedura ricorsiva:

**To compute a topologial ordering of G:**

Find a node v with no incoming edges and order it first

Delete v from G

Recursively compute a topological ordering of G-{v}and append this order after

**Spiegazione**

**To compute a topologial ordering of G:**   
*trova uno dei vari ordinamenti topologici possibili di G.*

Find a node v with no incoming edges and order it first   
*Se il grafo ha un nodo v senza archi entranti*

Delete v from G  *Lo cancello dal grafo*

Recursively compute a topological ordering of G-{v}and append this order after *Invoco ricorsivamente la procedura su quel che resta in modo tale da ottenere un ordinamento topologico di quel grafo ridotto. Quando esco dalla chiamata ricorsiva io ho un ordinamento di tutti i nodi tranne quello v che ho tolto. Quel v che ho tolto lo metto all'inizio della lista.*

**Analisi tempo**

Trovare un nodo v senza archi entranti mi richiede tempo lineare nel numero di nodi (La prima volta n, poi n - 1, ... )

Cancellare v dal grafo insieme a tutti i suoi archi mi costa tempo lineare sul grado del nodo, quindi O(*deg*(v))

Il tempo della chiamata ricorsiva lo ignoriamo.

Per mettere v all'inizio dell'ordinamento topologico dopo la chiamata ricorsiva, (se abbiamo una lista ben implementata) O(1)

Se io considero tutte le chiamate ricorsive, trovare un nodo costa n la prima volta, n-1 la seconda, eccetera. Quindi mi costa n + (n-1) + ..., ossia una sommatoria che va da 2 ad n (2 perché quando mi rimane uno solo ci fermiamo), ed ottengo il solito quadrato, O(n^2).

Per capire quanto mi costa cancellare v dal grafo insieme a tutti i suoi archi sul totale di tutte le chiamate ricorsive io devo sommare tutti i gradi che sappiamo essere m (Il grafo è direzionato, quindi è solo m non 2m), quindi è O(m) in totale.

Ogni singolo append alla lista costa O(1), ripetuto n volte costa O(n).

Quindi O(n^2) + O(m) + O(n) è O(n^2 + m). n^2 è il termine dominante e quindi è O(n^2).

Vogliamo una procedura che sia più efficiente. Cosa richiede tanto tempo?

La cancellazione dell'elemento dipende dall'implementazione del grafo. Se è un grafo ben implementato richiede O(*deg*(v)) e non lo posso migliorare.

L'aggiunta alla lista non può richiedere meno di O(1). Quello su cui devo intervenire è la ricerca del nodo senza archi entranti. Prima la facevo in tempo lineare nel numero di nodi. Devo trovare un modo più efficiente per farlo. Ad ogni chiamata ricorsiva a me interessano solo i nodi che sono ancora nel grafo. Li chiamerò nodi attivi. Se riuscissi a mantenere un insieme S facile da gestire (come ad esempio una lista) di nodi attivi senza archi entranti, in tempo costante riuscirei a trovare un nodo senza archi entranti. Ho necessità di aggiornare la lista velocemente, perché ad ogni chiamata ricorsiva io potrei creare altri archi senza nodi entranti. Vediamo come farlo: per ogni nodo mantengo un contatore count (quindi creo un array), e ad ogni nodo v, count[v] conterrà il numero di archi entranti. Questo valore mi interessa solo per i nodi attivi. Non appena il count di nodo è 0 lo metto nell'insieme S dei nodi attivi senza archi entranti.

**Proviamo a costruire l'algoritmo.**

Inizializzazione dei campi count. Come posso farlo? Scandendo i nodi oppure scandendo gli archi.

**Scandendo gli archi**

Per ogni arco (u,v) che incontro incremento il count di v. Costa O(m)

Devo inizializzare S, a cosa? Per scoprire chi è S vado a vedere i campi count. Ogni volta che trovo un count[v]=0 inserisco v in S. O(n)

**Scandendo i nodi**

Per ogni vertice v, scandisco l'indeg(v) e lo inserisco in count. O(m)

Inizializzo S con tutti i vertici v tali che count[v]=0. O(n)

L'inizializzazione l'abbiamo fatta. Mi costa tempo O(n+m). Dove va fatta? fuori dalla procedura ricorsiva.

Devo trovare un nodo senza archi entranti.

Come faccio? vado in S e prendo il primo nodo. O(1)

Devo ora cancellare questo nodo dal grafo. Questa cancellazione comporta:

cancellarlo da S, tempo costante O(1)

cancellarlo da G, tempo O(deg(v))

aggiornare i campi count, tempo O(deg(v)).

In totale questa cancellazione costa O(deg(v)) + O(1).

Ignoriamo la chiamata ricorsiva.

L'append alla lista costa O(1).

**Algoritmo:**

S <-

Inizializza L ad una lista vuota

Foreach arco (u,v)

count[v] = count [v] + 1

Endforeach

Foreach u in G

If(count[v]=0)

S <- S {u}

Endif

Endforeach

OrdinamentoTopologico(S, L, G)

**OrdinamentoTopologico(S, L, G):**

if(S = )

return

Prende un elemento da S e lo mette in Z

Elimina Z da S

Foreach arco (z,w)

count[w] = count[w] – 1

if(count[w]=0)

Aggiungi w ad S

Endif

Endforeach

Cancella Z da G

OrdinamentoTopologico(S, L, G)

Aggiunge Z in testa ad L

Tutto questo, per tutte le chiamate ricorsive:

* O(m+n) per l'inizializzazione,
* O(n) per prendere il primo nodo da S in totale
* La cancellazione costa O(m + n)
* L'append costa O(n).

**In totale: O(m+n) + O(n) + O(m+n) + O(n), quindi in totale O(m+n)**