

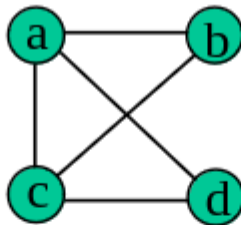
Grafi in memoria e Visite di grafi

Esistono diverse strutture che si possono usare per rappresentare i grafi in memoria. Principalmente si distinguono in due:

- **Matrici di Adiacenza**
- **Liste di Adiacenza**

Grafi non diretti

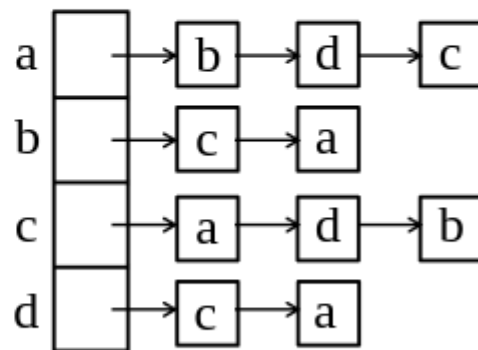
Quanto spazio?



	a	b	c	d
a	0	1	1	1
b	1	0	1	0
c	1	1	0	1
d	1	0	1	0

Matrice di adiacenza

$O(n^2)$



liste di adiacenza

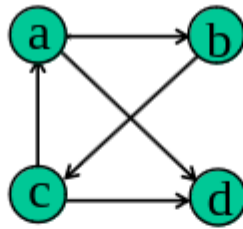
$O(m + n)$

Le operazioni che possiamo eseguire sono:

Operazione	Matrice di Adiacenza	Liste di Adiacenza
Elenco archi incidenti in v	$O(n)$	$O(\delta(v))$
Ricerca arco (u, v)	$O(1)$	$O(\min[\delta(u), \delta(v)])$

Grafi diretti

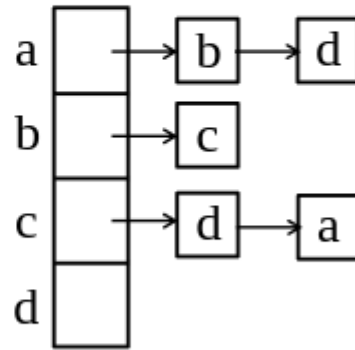
Quanto spazio?



	a	b	c	d
a	0	1	0	1
b	0	0	1	0
c	1	0	0	1
d	0	0	0	0

Matrice di adiacenza

$O(n^2)$



liste di adiacenza

$O(m + n)$

Le

operazioni che possiamo eseguire sono:

Operazione	Matrice di Adiacenza	Liste di Adiacenza
Elenco archi incidenti in v	$O(n)$	$O(\delta(v))$
Ricerca arco (u, v)	$O(1)$	$O(\delta(u))$

Scopo e tipi di visita

Una visita in un grafo ci permette di visitare i nodi in **modo sistematico**, ciò che si genera da questa visita è un **albero di visita**.

Esistono vari tipi di visite principalmente:

- **BFS** -> Visita in ampiezza
- **DFS** -> Visita in profondità

Visita in Ampiezza

Funzionamento

Dato un grafo G (non pesato) e un nodo s , trova tutte le distanze/cammini minimi da s verso ogni altro nodo v .

Applicazioni

- Web Crawling
- Social Networking
- Network Broadcast
- Garbage Collection
- Model Checking
- Risolvere puzzle

```
ALgoritmo visitaBFS(vertice s) -> albero
    Rendi tutti i vertici non marcati
    T <- Albero formato da un solo nodo s
    Coda F
    marca il vertice s; dist(s) <- 0
    F.enqueue(s)
    while (not F.isempty()) do
        u <- F.dequeue()
        for each (arco(u,v) in G) do
            if (v non è ancora marcato) then
                F.enqueue(v)
                marca il vertice v; dist(v) <- dist(u) + 1
                rendi u padre di v in T
    return T
```


Partiamo da un qualsiasi nodo, inseriamo nella coda tutti i nodi vicini a quello di partenza, una volta inseriti tutti, eliminiamo il nodo di partenza e facciamo la stessa cosa con il primo nodo disponibile nella coda.

Complessità:

Il tempo di esecuzione dipende dalla struttura che usiamo per rappresentare il grafo, infatti:

- Liste di Adiacenza -> $O(m + n)$
- Matrice di Adiacenza -> $O(n^2)$

Osservazioni

1. Se il grafo è connesso allora $m \geq n - 1$ e quindi $O(m + n) = O(m)$
2. Ricordiamo che $m \leq n(n - 1)$  si ha $O(m + n) = O(n^2)$

Per $m = o(n^2)$ la rappresentanza mediante liste di adiacenza è **temporalmente più efficiente**.

Teorema

Per ogni nodo v , il livello di v nell'albero BFS è pari alla distanza di v dalla sorgente s (Sia per i grafi orientati che non orientati).

Visita in Profondità

Quello che ci serve per creare un DFS è una pila ed una variabile booleana.

```
procedura visitaDFS Ricorsiva(vertice v, albero T):  
    marca e visita il vertice v  
    for each (arco(u,v)) do  
        if(w non è marcato) then  
            aggiungi l'arco (v,w) dell'albero T  
            visitaDFS Ricorsiva(w,T)  
  
algoritmo visitaDFS(vertice s) -> albero  
    T <- albero vuoto  
    visitaDFS Ricorsiva(s,T)  
    return T
```

Controlliamo ogni nodo possibile, finché non ci troviamo in un nodo che non ha più vicini, allora torniamo indietro al primo nodo che ha vicini ancora non esplorati.

Complessità: Il tempo di esecuzione dipende dalla struttura dati usata:

- Liste di Adiacenza -> $O(m + n)$
- Matrice di Adiacenza -> $O(n^2)$

Proprietà dell'albero DFS radicato in s:

1. Se il grafo non è orientato, per ogni arco (u, v) si ha che: (u, v) è un arco dell'albero DFS, oppure i nodi u e v sono l'uno **discentente/antenato** dell'altro.
2. Se il grafo è orientato, per ogni arco (u, v) si ha che: (u, v) è un arco dell'albero DFS, oppure i nodi u e v sono l'uno **discentente/antenato** dell'altro, oppure (u, v) è un arco

trasversalmente a sinistra, ovvero il vertice v è in un sottoalbero visitato precedentemente ad u .