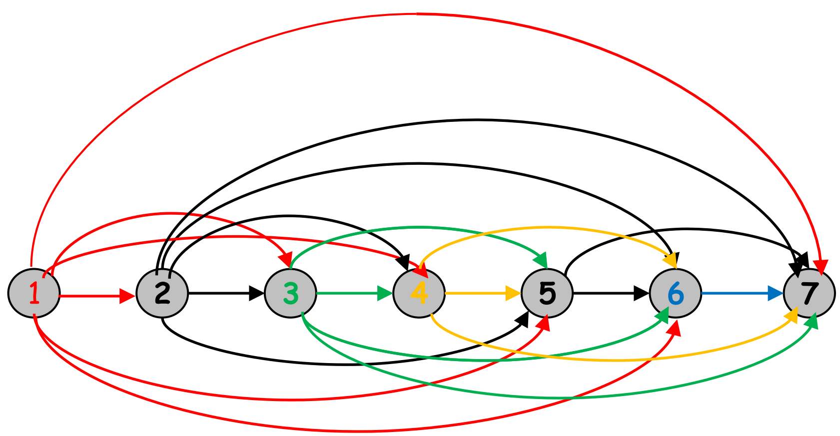
Grafi: Ordinamento topologico e DAG

Ritorno alla PD: primo approccio al problema dei cammini minimi

4 maggio 2023



Grafi

Il grafo è una delle strutture più espressive e fondamentali della matematica discreta.

Semplice modo di modellare relazioni a coppie in un insieme di oggetti.

Innumerevoli applicazioni.

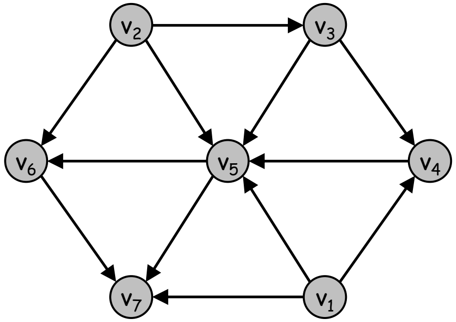
"Più si lavora coi grafi, più si tende a vederli ovunque".

3.6 DAGs and Topological Ordering

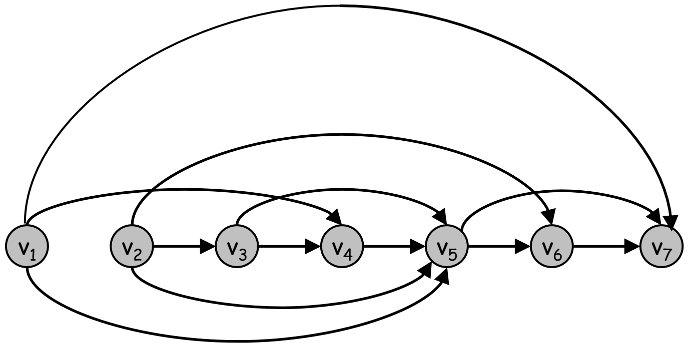
Def. An DAG is a directed graph that contains no directed cycles.

Ex. Precedence constraints: edge (v_i, v_j) means v_i must precede v_j.

Def. A topological order of a directed graph $G = (V, E)$ is an ordering of its nodes as $v_1, v_2, \ldots, v_n$ so that for every edge (v_i, v_j) we have $i < j$.



a DAG



a topological ordering

Precedence constraints. Edge (v_i, v_j) means task v_i must occur before v_j.

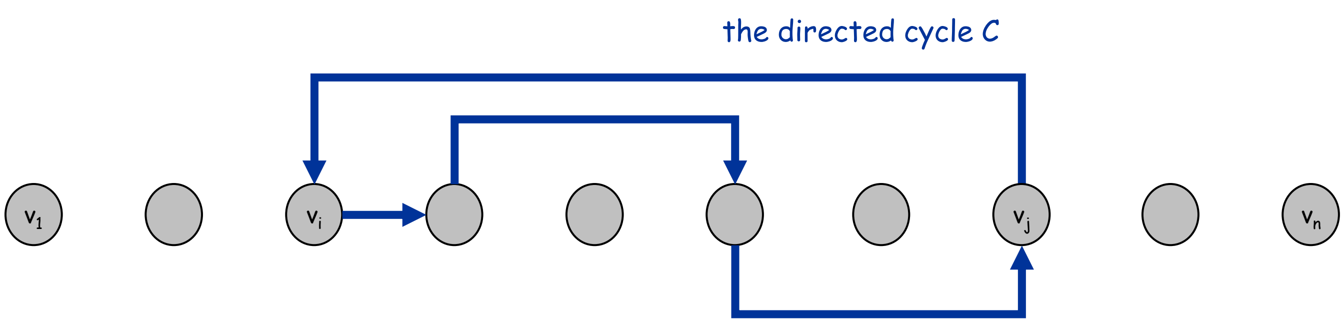
Applications.

- Course prerequisite graph: course v_i must be taken before v_j.
- Compilation: module v_i must be compiled before v_j. Pipeline of computing jobs: output of job v_i needed to determine input of job v_j.

Lemma. If G has a topological order, then G is a DAG.

Pf. (by contradiction)

- Suppose that G has a topological order $v_1, \ldots, v_n$ and that G also has a directed cycle C. Let's see what happens.
- Let v_i be the lowest-indexed node in C and let v_j be the node just before v_i; thus (v_j, v_i) is an edge.
- By our choice of i, we have $i < j$.
- On the other hand, since (v_j, v_i) is an edge and $v_1, \ldots, v_n$ is a topological order, we must have $j < i$, a contradiction. ▪



the supposed topological order: $v_1, \ldots, v_n$

Lemma. If G has a topological order, then G is a DAG.

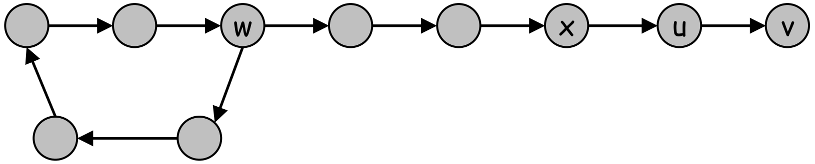
Q. Does every DAG have a topological ordering?

Q. If so, how do we compute one?

Lemma. If G is a DAG, then G has a node with no incoming edges.

Pf. (by contradiction)

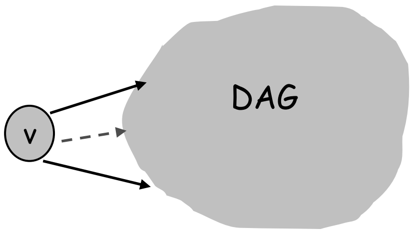
- Suppose that G is a DAG and every node has at least one incoming edge. Let's see what happens.
- Pick any node v, and begin following edges backward from v. Since v has at least one incoming edge (u, v) we can walk backward to u.
- Then, since u has at least one incoming edge (x, u), we can walk backward to x.
- Repeat until we visit a node, say w, twice.
- Let C denote the sequence of nodes encountered between successive visits to w. C is a cycle. ▪



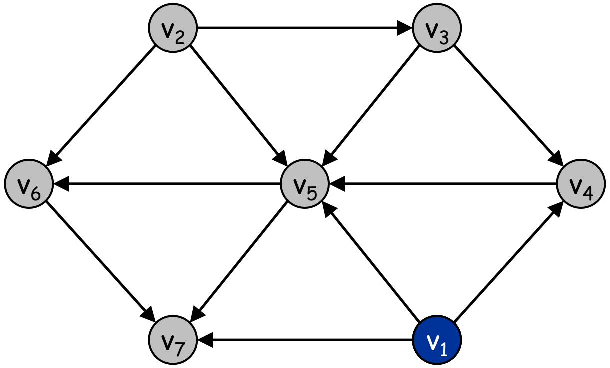
Lemma. If G is a DAG, then G has a topological ordering.

Pf. (by induction on n)
Base case: true if n = 1.
Given DAG on n > 1 nodes, find a node v with no incoming edges.
G - { v } is a DAG, since deleting v cannot create cycles.
By inductive hypothesis, G - { v } has a topological ordering.
Place v first in topological ordering; then append nodes of G - { v } in topological order. This is valid since v has no incoming edges. ▪

To compute a topological 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 v

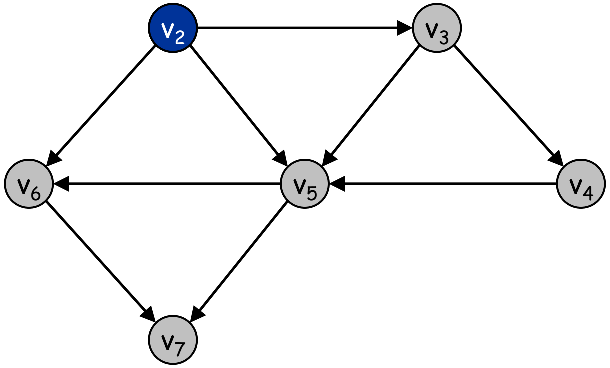


Topological Ordering Algorithm: Example



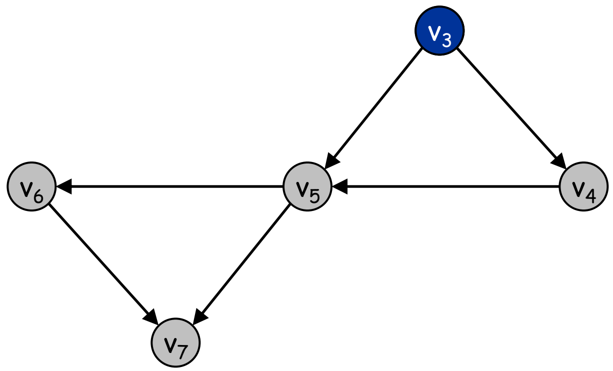
Topological order:

Topological Ordering Algorithm: Example



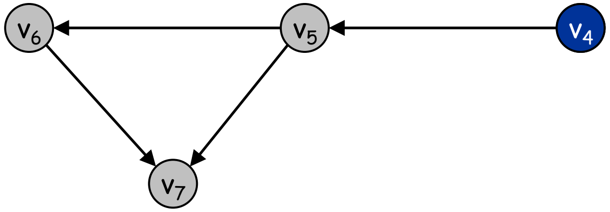
Topological order: v_1

Topological Ordering Algorithm: Example



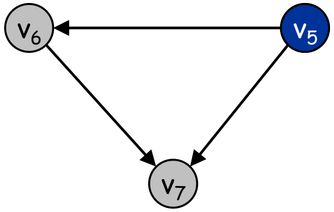
Topological order: v_1, v_2

Topological Ordering Algorithm: Example



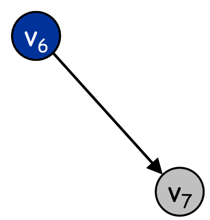
Topological order: v_1, v_2, v_3

Topological Ordering Algorithm: Example



Topological order: v_1, v_2, v_3, v_4

Topological Ordering Algorithm: Example

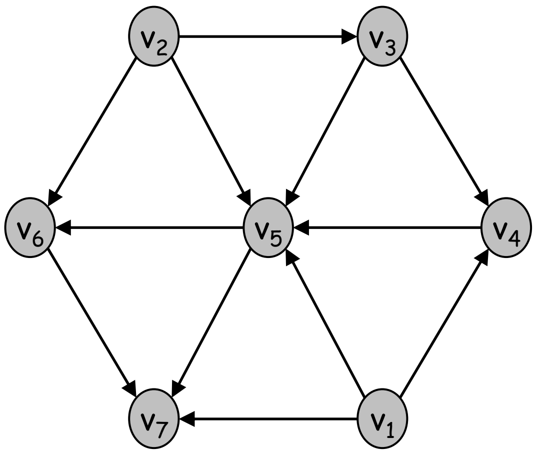


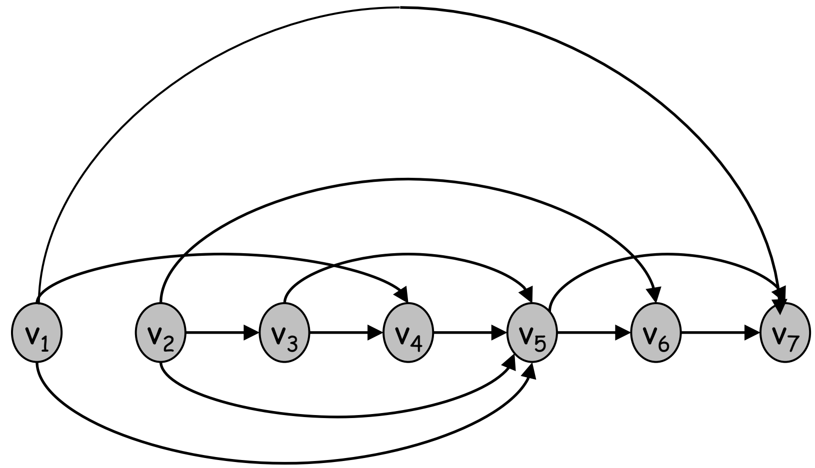
Topological order: v_1, v_2, v_3, v_4, v_5

Topological Ordering Algorithm: Example

Topological order: $v_1, v_2, v_3, v_4, v_5, v_6$

Topological Ordering Algorithm: Example



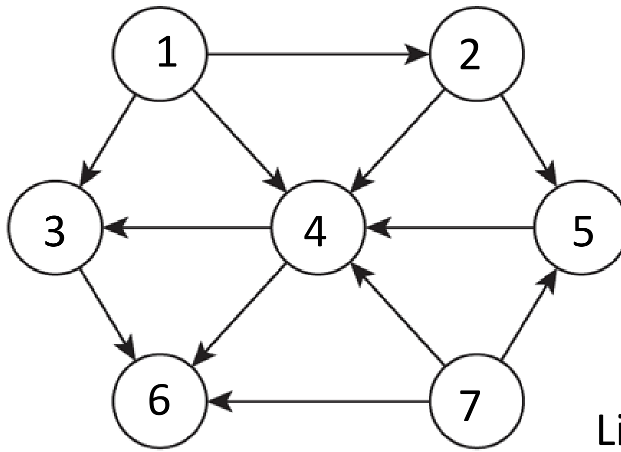


A topological order: $v_1, v_2, v_3, v_4, v_5, v_6, v_7$.

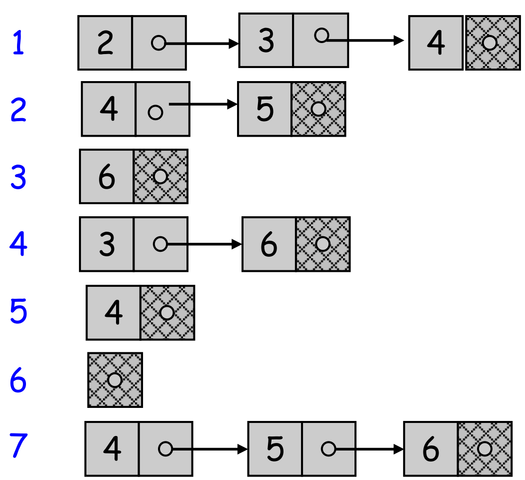
There are other topological orders, for example starting from: v_2.

Rappresentazione di un grafo diretto:

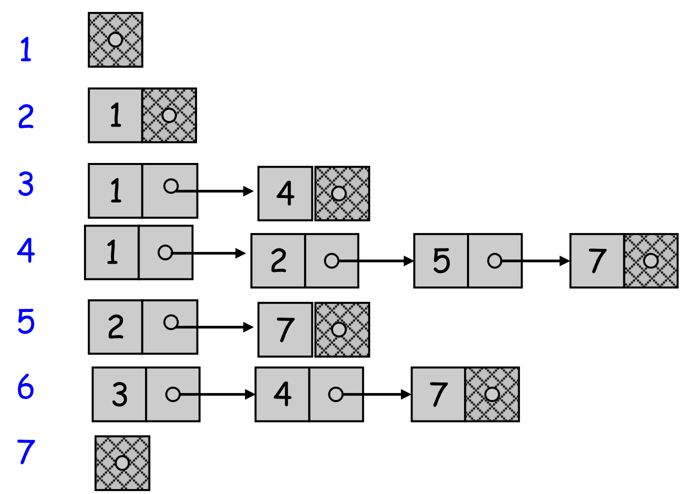
Lista di adiacenza: array di n liste indicizzate dai nodi.



Lista out



Lista in



Theorem. Algorithm finds a topological order in $O(m + n)$ time.

Pf.

- Maintain the following information:
  - `count[w]` = remaining number of incoming edges
  - S = set of remaining nodes with no incoming edges
- Initialization: $O(m + n)$ via single scan through graph.
- Update: to delete v
  - remove v from S
  - decrement `count[w]` for all edges from v to w, and add w to S if `count[w]` hits 0
  - this is $O(1)$ per edge ▪

Solved Exercise 1, page 104

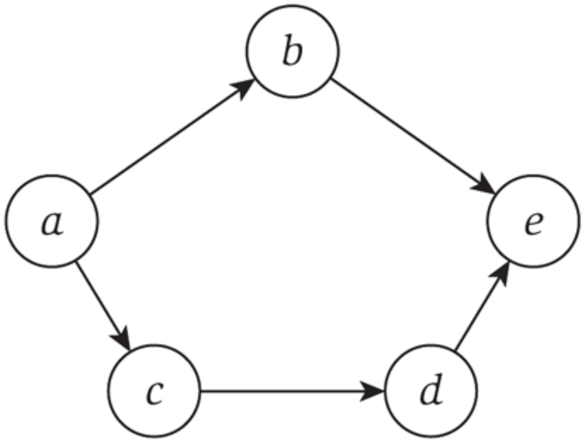


Figure 3.9 How many topological orderings does this graph have?

Exercise 1, page 107

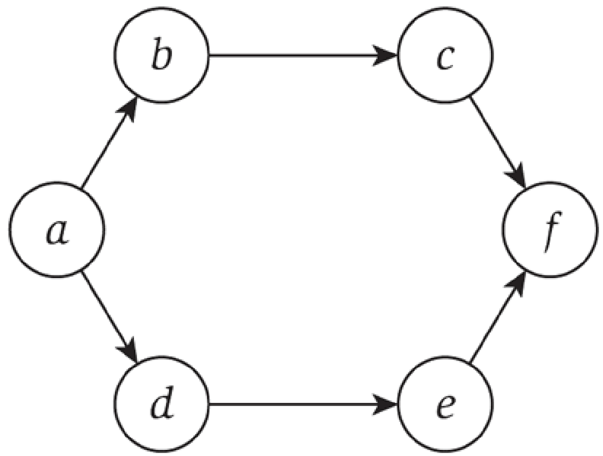


Figure 3.10 How many topological orderings does this graph have?

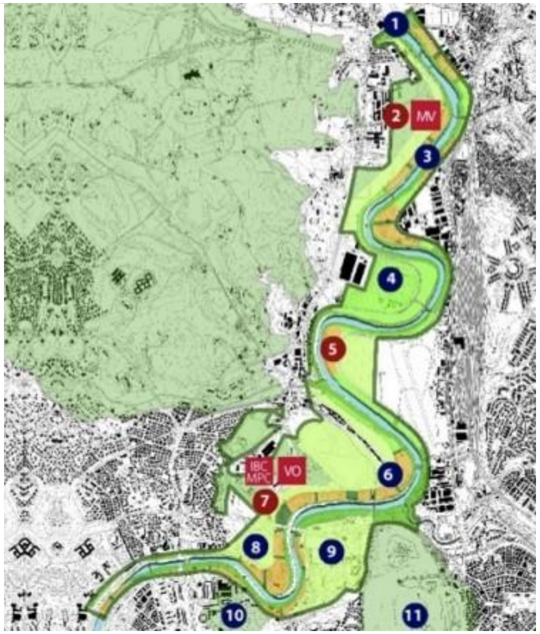
BACK to dynamic programming!

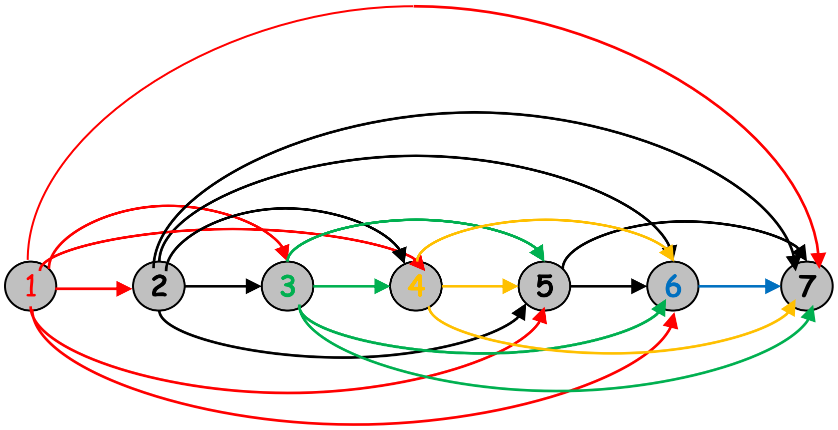
Problema della canoa

Esercizio 1 (detto della canoa)
Lungo un fiume ci sono n approdi. A ciascuno di questi approdi é possibile fittare una canoa che puó essere restituita ad un altro approdo. E' praticamente impossibile andare controcorrente. Il costo del fitto di una canoa da un punto di partenza i ad un punto di arrivo j, con $i < j$, é denotato con $C(i, j)$. E' possibile che per andare da i a j sia piú economico effettuare alcune soste e cambiare la canoa piuttosto che fittare una unica canoa. Se si fitta una nuova canoa in $k_1 < k_2 < ... < k_l$ allora il costo totale per il fitto é $C(i, k_1) + C(k_1, k_2) + ... + C(k_{l-1}, k_l) + C(k_l, j)$.
Descrivere un algoritmo che dato in input i costi $C(i, j)$, determini il costo minimo per recarsi da 1 ad n. Analizzare la complessitá dell'algoritmo proposto.

Esempio. Sia $n = 4$, e $C(1, 2) = 1$, $C(1, 3) = 2$, $C(1, 4) = 4$, $C(2, 3) = 1$, $C(2, 4) = 1$, $C(3, 4) = 1$. Allora i possibili modi per andare da 1 a 4 sono: $1 \to 4$, $1 \to 2 \to 4$, $1 \to 3 \to 4$, $1 \to 2 \to 3 \to 4$. I rispettivi costi sono: 4, 2, 3, 3. Il costo minimo é quindi 2.

PROBLEMA COMPUTAZIONALE





INPUT: un grafo diretto G=(V,E) con

V = {1,2,…,n}, E = {(i,j) t.c. i,j in V e i<j }

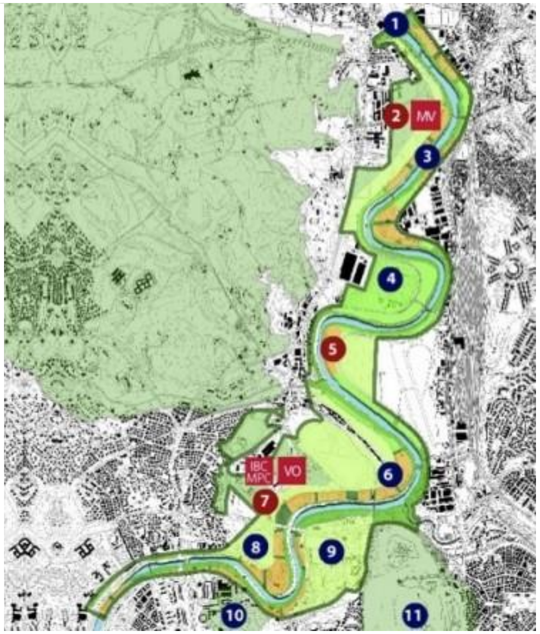
C(i,j) costo dell'arco (i,j) per ogni (i,j) in E

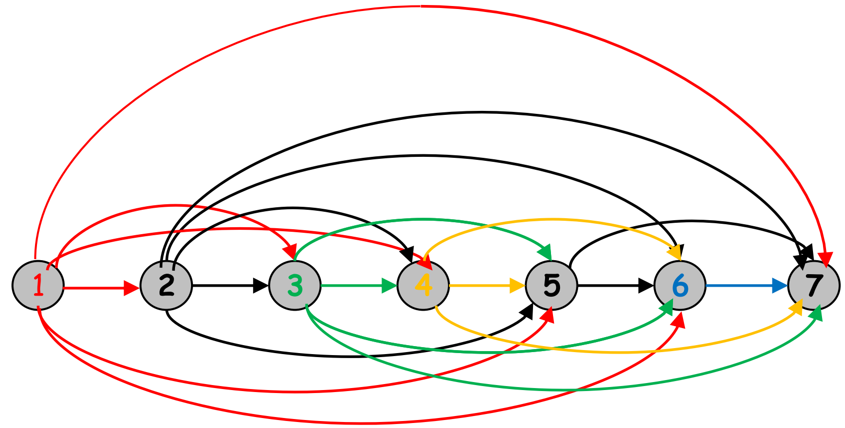
OUTPUT: un cammino da 1 a n di costo totale minimo

oppure

OUTPUT_2: costo minimo di un cammino da 1 a n

PROBLEMA COMPUTAZIONALE





INPUT: un grafo diretto aciclico G=(V,E) con

V = {1,2,…,n}, E = {(i,j) t.c. i,j in V e i<j }

C(i,j) costo dell'arco (i,j) per ogni (i,j) in E

OUTPUT: un cammino da 1 a n di costo totale minimo

oppure

OUTPUT_2: costo minimo di un cammino da 1 a n

Shortest Path Problem

Shortest path network.

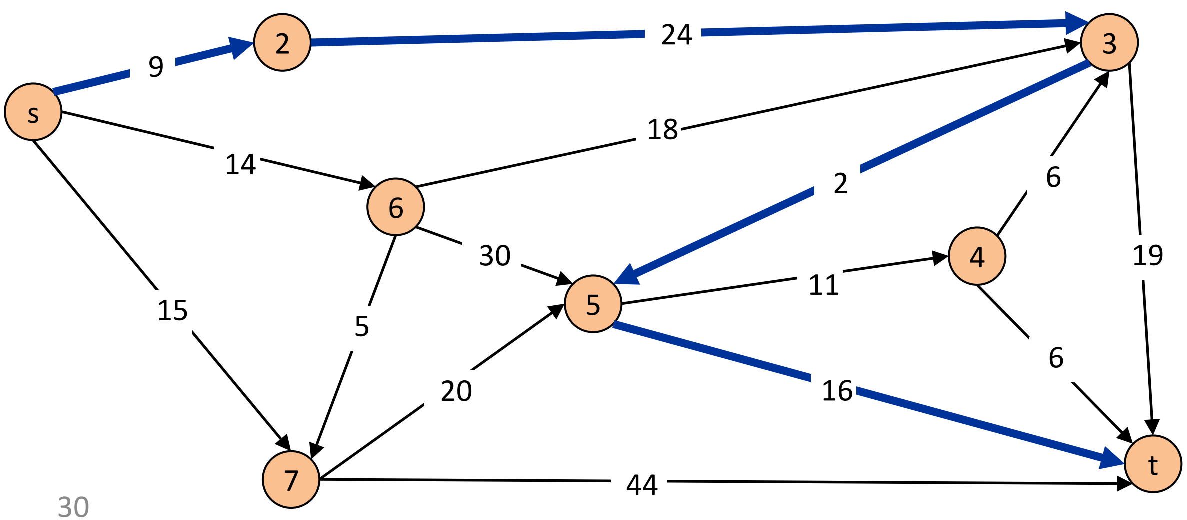
Directed graph $G = (V, E)$.

Source s, destination t.

Length λ_e = length/cost/weight of edge e.

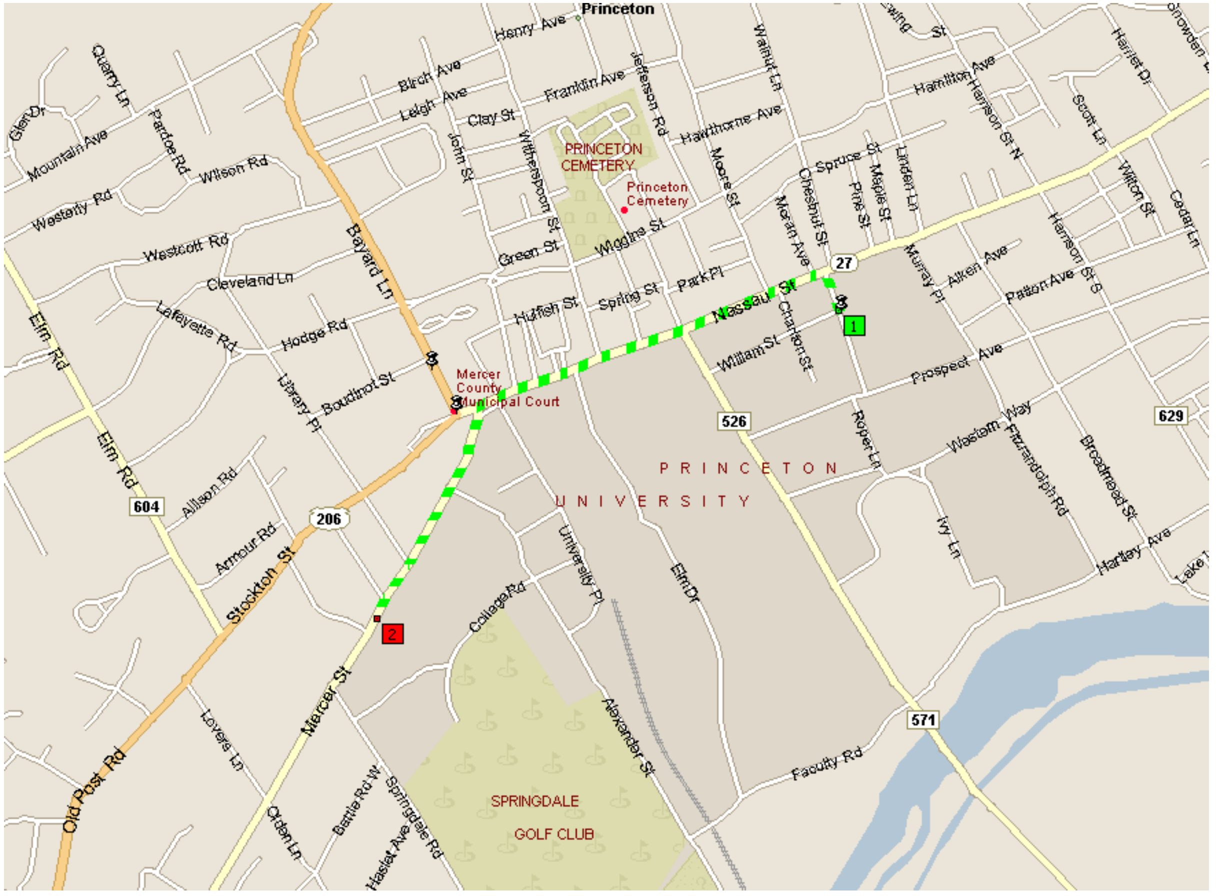
Shortest path problem: find shortest (min cost) directed path from s to t.

cost of path = sum of edge costs in path



Cost of path s-2-3-5-t
= 9 + 24 + 2 + 16
= 51.

Shortest Paths in a Graph



Shortest path from Princeton CS department to Einstein's house

Una applicazione

Problema dell'allineamento di sequenze

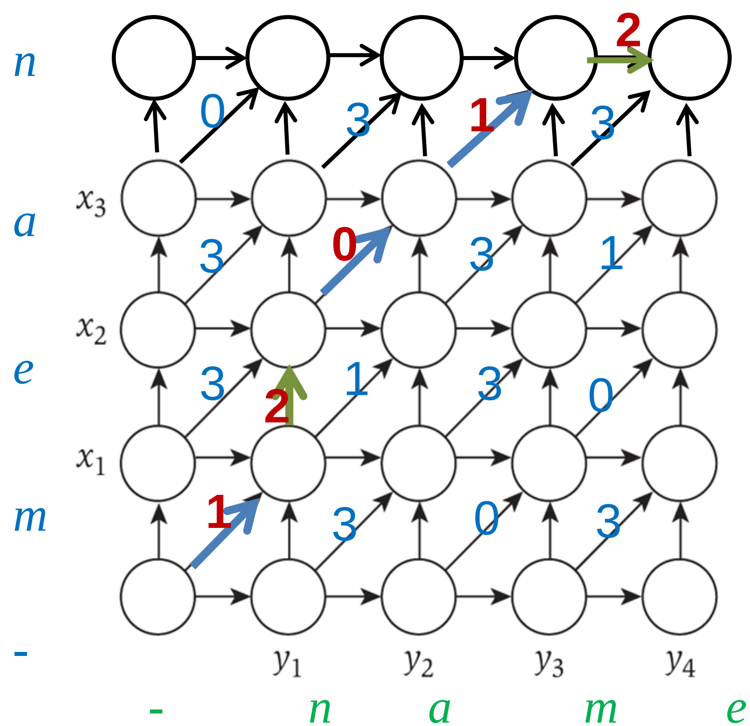


Figure 6.17 A graph-based picture of sequence alignment.

Ricerca di un cammino di costo minimo in un DAG con costi positivi……

Shortest paths

6.8 Shortest Paths in a Graph:

Bellman & Ford Algorithm: Dynamic Programming

4.4 Shortest Paths in a Graph (only with positive costs):

Dijkstra Algorithm: Greedy

Problema della canoa: come risolverlo?

Si può sempre provare con la **ricerca esaustiva** (brute force, naïf):

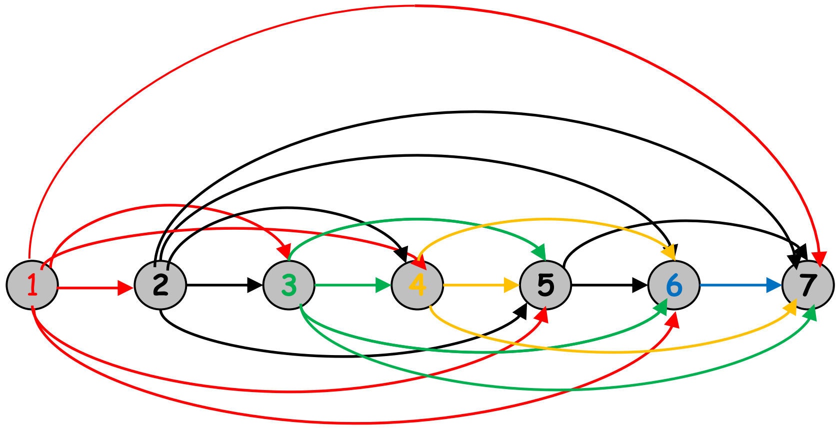
- Considero tutti i possibili cammini da 1 a n
- Per ognuno calcolo il peso
- Restituisco un cammino di peso minimo

Per piccoli input può andare, ma per input grandi?
Qual è la complessità del tempo di esecuzione al crescere della taglia dell'input?

Qual è il numero di tutti i cammini da 1 a n? 2^{n-2}

1,2,3,4,5 S={2,3,4}

(1,5)	(1,2,5)	(1,2,3,5)	(1,2,3,4,5)
	(1,3,5)	(1,2,4,5)	
	(1,4,5)	(1,3,4,5)	



C =

	1	2	3	4	5	6	7
1		1	2	4	1	2	9
2			1	1	3	2	4
3				1	1	5	2
4					2	1	2
5						3	4
6							5
7							

Problema con V =	Soluzione ottimale	Valore
{1}	()	0
{1,2}	(1,2)	1
{1,2,3}	(1,3) o (1,2,3)	2
{1,2,3,4}	(1,2,4)	2
{1,2,3,4,5}	(1,5)	1
{1,2,3,4,5,6}	(1,6)	2
{1,2,3,4,5,6,7}	(1,3,7) o (1,2,3,7) o (1,2,4,7)	4

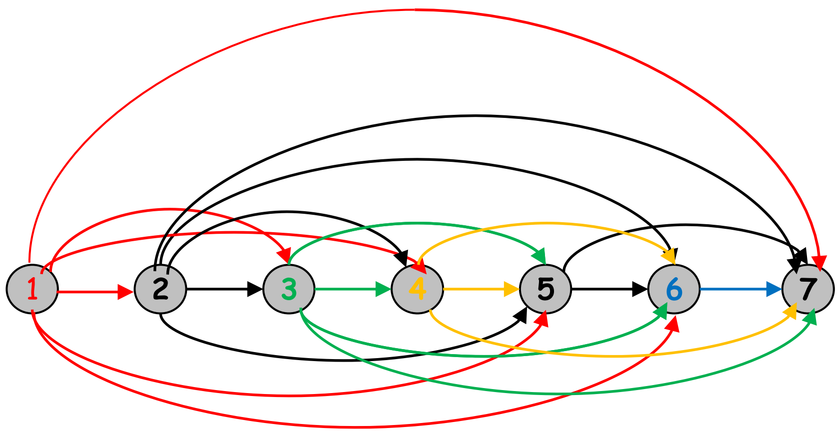
(1, 3) o (1,2,3)? min{2, 1+1} = 2

(1,4), (1,2,4), (1,3,4), (1,2,3,4)?
min{4, 1+1+1, 1+1, 2+1} =2

(1,5), (1,...,2,5), (1,..,3,5), (1, ..., 4, 5),
min{1, 1+3, 2+1, 2+2} = 1

(1,6), (1...2,6), (1...3,6), (1..4,6), (1...5,6)
min{ 2, 1+2, 2+5, 2+1, 1+3} =2

min{ 9, 1+4, 2+2, 2+2, 1+4, 2+5} = 4



C =

	1	2	3	4	5	6	7
1		1	2	4	1	2	9
2			1	1	3	2	4
3				1	1	5	2
4					2	1	2
5						3	4
6							5
7							

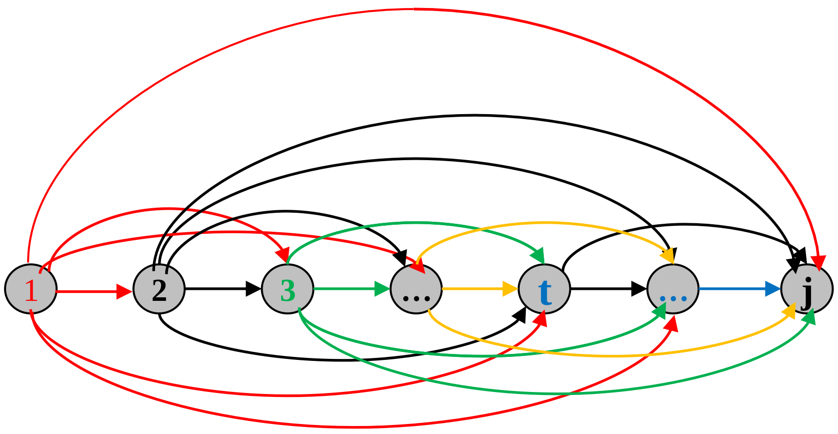
Problema con V =	Soluzione ottimale	Valore
{1}	()	0
{1,2}	(1,2)	1
{1,2,3}	(1,3) o (1,2,3)	2
{1,2,3,4}	(1,2,4)	2
{1,2,3,4,5}	(1,5)	1
{1,2,3,4,5,6}	(1,6)	2
{1,2,3,4,5,6,7}	(1,3,7) o (1,2,3,7) o (1,2,4,7)	4

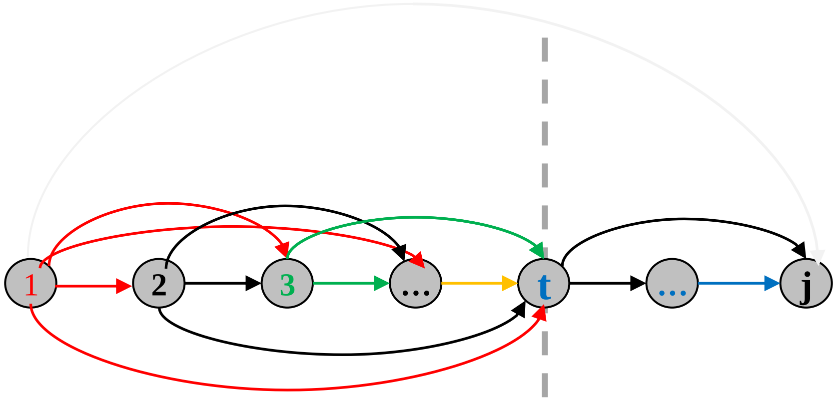
In generale:
come possiamo ottenere il valore ottimo per {1, 2, …, j-1, j },
supponendo di conoscere i valori ottimi per i problemi {1, …, i} più piccoli?
Considero j e vedo cosa conviene fra tute le possibilità per

- aggiungere l’arco (t,j) a una soluzione ottimale per {1, …,t} compatibile

Sottoproblemi

Cercare cammino di costo minimo da 1 a j





cammino di costo minimo da 1 a t e arco (t, j)

costo minimo di un cammino da 1 a t + C(t, j)

Dynamic Programming: Multiple Choice

Notation. **OPT(j)** = value of optimal solution to the problem consisting of vertices 1, 2, ..., j.

optimal substructure

Case t, for any t = 1, 2, ..., j-1:

an optimal path OPT selects edge (t, j) and add an optimal solution to problem consisting of vertices 1, 2, ..., t

$$OPT(j) = \begin{cases} 0 & \text{if } j = 1 \\ \min\limits_{t=1, \ldots, j-1} \{OPT(t) + C(t, j)\} & \text{otherwise} \end{cases}$$

Esercizio (Algoritmo Canoa 1)

- Scrivere lo pseudocodice di un algoritmo di programmazione dinamica che calcola OPT(n) con OPT(j) definito come nella slide precedente.
- Analizzarne la complessità di spazio e il tempo di esecuzione.

Dynamic Programming: symmetric view

Notation. **OPT(j)** = value of optimal solution to the problem consisting of vertices j, j+1, ..., n.

Case t, for any t=j+1, ..., n:

optimal substructure

an optimal path OPT selects edge (j, t) and add an optimal solution to problem consisting of vertices t, t+1, ..., n

$$OPT(j) = \begin{cases} 0 & \text{if } j = n \\ \min\limits_{t = j+1, \ldots, n} \{C(j,t) + OPT(t)\} & \text{otherwise} \end{cases}$$

Esercizio (Algoritmo Canoa 1-reverse)

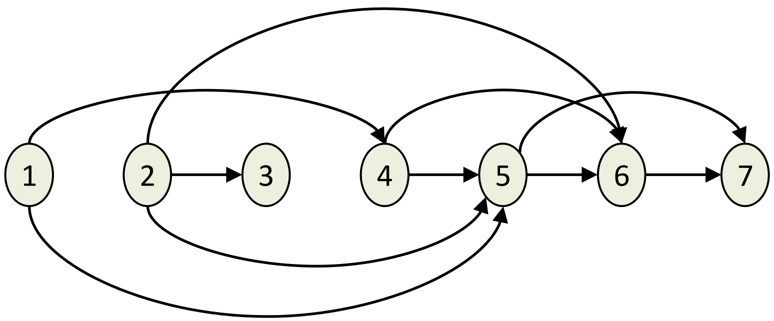
- Scrivere lo pseudocodice di un algoritmo di programmazione dinamica che calcola OPT(1) con OPT(j) definito come nella slide precedente.
- Analizzarne la complessità di spazio e il tempo di esecuzione.

DAG non ‘’completo’’

Se invece il DAG non fosse ‘’completo’’, cioè non fossero necessariamente presenti **tutti** gli archi (v,w) per ogni v, w in V.

Per qualche v potrebbe non esistere un cammino da v a t; in tal caso costo ∞.

Nell’esempio: non esistono archi uscenti da 3, e quindi non esistono cammini da 3 a 7. Esistono ugualmente cammini dagli altri vertici a 7.



OPT[v] = min costo di un cammino da v a n,
∞ se non esistono cammini da v ad n

$$OPT[n] = 0$$

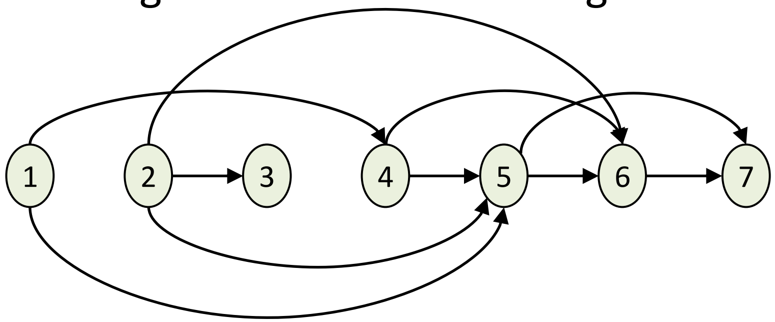
$$OPT[v] = \min \{ \infty, \min_{(v,w)\in E}\{ OPT[w] + C(v,w)\} \} \text{ se } v \neq n$$

Soluzione = OPT[1]

DAG non ''completo''

Se invece il DAG non fosse ''completo'', cioè non fossero necessariamente presenti **tutti** gli archi (v,w) per ogni v, w in V, per qualche v potrebbe non esistere un cammino da v a t.

Nell'esempio: non esistono archi uscenti da 3, e quindi non esistono cammini da 3 a 7. Esistono ugualmente cammini dagli altri vertici a 7.



OPT[v] = min costo di un cammino da v a n,
∞ se non esistono cammini da v ad n

OPT[n] = 0

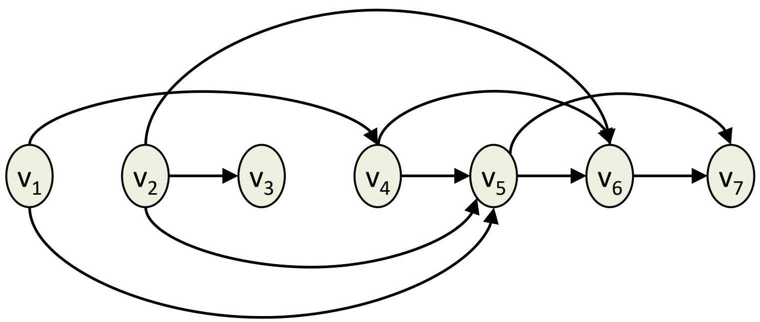
$$OPT[v] = \min \{ \infty, \min_{(v,w) \in E} \{ OPT[w] + C(v,w) \} \} \text{ se } v \neq n$$

E' un problema di cammini minimi, ma su DAG, ovvero su grafo diretto con un **ordinamento topologico** quindi, se effettuo i calcoli secondo l'ordinamento dei vertici dal più grande al più piccolo, quando calcolo OPT[v], OPT[w] è già stato calcolato

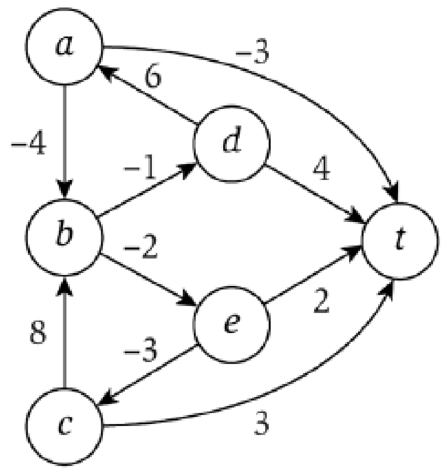
Soluzione = OPT[1]

```
Shortest-Path(G)
M[n]=0
for v=n-1 downto 1
        M[v]= ∞
        foreach (v,w)∈E
        if M[w]+C(v,w)< M[v]
            then M[v]= M[w]+C(v,w)
return M[1]
```

Grafo generico



E' un DAG, ovvero un grafo diretto con un **ordinamento topologico**.



Non è un DAG, ovvero **non** ha un **ordinamento topologico**.
Ho il ciclo c-b-e-c.

Se calcolassi

M[c] = min{ C(c,b)+M[b] , C(c,t)+M[t] }

in che ordine dovrei calcolare M[c], M[b], M[e] per essere sicuri di richiamare sempre valori già calcolati?

Bellman e Ford usano l'ordine di lunghezza dei cammini