INDEX

Complessità Assintotica

Lezione2

- MaxConsecutivaSomma

Lezione3

- (n)
- Tabella complessit

Lezione4

- (n)
- (n)

Lezione5

- Teorema risoluzione relazione di equivalenza nella forma: T(n) = aT(n/c) + bn^k

Divide and Impera (Conquer)

Lezione6

- RicercaBinaria
- ContaPromossi (conteggio elementi >k)
- SlowPower
- FastPower
- CoppieIdentiche
- Radice

Lezione7

- MergeSort, Merge
- Ogni algoritmo basato su confronti richiede (n log n) (Dimostrazione)
- Mancante
- MaxProfitto

Lezione8

- ContaInversioni, ContaInversioniMerge
- Unimodale (indice h tale che a[0]>...>a [h]<...<a [n-1])
- ContaMin
- TrovaMaxRel
- Trova-i (identificare due valori consecutivi crescenti in un vettore

Lezione9

- QuickSelect, Distribuzione. (rango di un elemento x)

Lezione10

- Tecnica induttiva alla risoluzione di equazioni di ricorrenza
- CercaInMatrice
- CercaMaggioritario
- CercaPrimaOccorrenza. (RicercaBinaria a lezione6 gi lo fa ed pi semplice)

Programmazione Dinamica

Lezione11

- Fib, MemFib, IterFib
- Choose, MemChoose, IterChoose. (coefficiente binomiale)

Lezione12

- CalcolaModi (#modi di scrivere un intero X con n numeri da 1 a m)
- Zaino, MemZaino, IterZaino

Lezione13

- Rec_CambioMonete
- Esami (pi alto punteggio scegliendo a quali domande rispondere)

Lezione14

- Calcola_OPT, M_Calcola_OPT. (scheduling attivit)
- TrovaSoluzioni (trovate gli elementi appartenenti alla soluzione)
- Scelta monete Alice e Bob

Lezione15

- Mem_Rec (Longest Common Subsequence (LCS) (calcola la lunghezza della sequenza)
- LCS_Print (stampa la sequenza)
- Edit_distanza (trasformate sequenza s in sequenza t, contare il numero di operazioni necessarie)

Lezione16

- Non tutti i problemi si possono risolvere con Programmazione Dinamica
- Calcola_OPT (calcolare il massimo valore prendendo nodi in un albero che hanno un peso e due nodi u e v non appartengono a E, ossia non sono padre-figlio)
- Palindromo (calcolare numero minimo di operazioni per trasformare una sequenza in una sequenza palindroma)
- CalcolaP (dividere un insieme di valori in due sottoinsiemi disgiunti tale che la differenza del valore totale di ciascuno dei due sottoinsiemi sia piu piccola possibile)

Lezione17

- Calcola_Pes (Pi Corta Supersequenza)
- Calcola il numero di volte che una sequenza p compare in una sequenza s

Greedy

Lezione18

- Cambio_greedy (cambio monete, ma con greedy)
- Greedy_Activity_Selector (GAS, selezionare maggior numero di attivit compatibili che possono essere eseguite in un determinato tempo)

Lezione19

- Problema di minimizzare il tempo Idle dei processi su un unico processore
- Greedy-Zaino_Frazionario

Lezione20

- Codifica di Huffman

Lezione21

- Partizionamento attivit (aule corsi)
- MaxSottosequenzaCrescente
- Taglio tubo
- Rettangoli
- Temperature
- Brani musicali

Grafi parte 1

Lezione22

- Definizioni per i grafi
- BFS

Lezione23

- DFS
- Componente connessa
- Grafo diretto fortemente connesso

Lezione24

- Cicli in grafi
- DAG (Directed Acyclic Graph)
- Ordinamento Topologico
- AlgoritmoFC (Controllare se un grafo fortemente connesso)
- AlgoritmoCF (Calcolare componente fortemente connessa contenente un nodo s)
- AlgoritmoTutte_CF (Calcolare tutte le componenti fortemente connesse di un grafo)

Lezione25

- Heap
- Heap-Insert (inserimento elemento in uno Heap)
- Heap-Extract-Min (trovare e rimovere il minimo in uno Heap)
- Min-Heapify (ripristinare la propriet di base key(v) >= key(parent (v)))
- Minimum (trovare minimo in uno Heap)
- Aumenta-Chiave (aumentare il valore di una chiave in uno Heap)
- Diminuisci-Chiave (diminuire il valore di una chiave in uno Heap)
- HeapSort (ordinare on array usando uno Heap)
- MaxHeap si pu costruire, con opportune modifiche, seguendo il MinHeap

Grafi parte 2

Lezione26

- Dijkstra

Lezione27

- MST (Minimum Spanning Tree)
- Kruskal
- Prim

Lezione28

- MST-Kruskal
- Make-Set
- Find-Set
- Union

Lezione29

- Shortest-Path (cammino minimo con archi negativi)
- Ciclo di costo negativo

Lezione30

- Larghezza di un albero
- Distanza media in un grato
- Cicli in grafi
- Problema del matrimonio
- Calcolo dei vicini dei vicini di un nodo
- Orientazione aciclica di grafi completi
- Numero di cammini minimi tra due nodi

Algoritmi esaustivi

Lezione31

- Backtracking
- Sottoinsiemi (Generare tutti i sottoinsiemi)
- Permutazioni (Generare tutte le permutazioni)
- Sottoinsiemi-Vincolati (Generate tutti i sottoinsiemi con cardinalit c)
- Cicli-Hemiltoniani

Lezione32

- Problema delle 8 regine
- Sudoku
- Zaino-T (Trovare sottoinsiemi la cui somma T)

Lezione33

- Branch and Bound
- Assegnare n agenti ad n compiti con costo minimo
- Zaino
- Commesso viaggiatore

ESERCIZI 1

Nome e Cognome (maiuscolo) _____

Matricola: _____

Esercizio 1. Completa l'albero binario

Algoritmo 1(n):
if n > 0:
print(n)
Algoritmo 1(n-1)
Algoritmo 1(n-1)
else:
return

Si dica una versione di ricerca di Ternario e di Fibonacci di Algoritmo 1(n) e si scriva la relazione di ricorrenza.

Esercizio 2. Completa l'albero binario

Algoritmo 2(n):
if n > 0:
print(n)
Algoritmo 2(n/2)
Algoritmo 2(n/2)
else:
return

Si dica una versione di ricerca di Ternario e di Fibonacci di Algoritmo 2(n) e si scriva la relazione di ricorrenza.

Esercizio 3. Completa l'albero binario

Algoritmo 3(n):
if n > 0:
print(n)
Algoritmo 3(n/3)
Algoritmo 3(n/3)
Algoritmo 3(n/3)
else:
return

Si dica una versione di ricerca di Ternario e di Fibonacci di Algoritmo 3(n) e si scriva la relazione di ricorrenza.

Esercizio 4. Completa l'albero binario

Algoritmo 4(n):
if n > 0:
print(n)
Algoritmo 4(n/4)
Algoritmo 4(n/4)
Algoritmo 4(n/4)
Algoritmo 4(n/4)
else:
return

Si dica una versione di ricerca di Ternario e di Fibonacci di Algoritmo 4(n) e si scriva la relazione di ricorrenza.

Esercizio 5. Completa l'albero binario

Algoritmo 5(n):
if n > 0:
print(n)
Algoritmo 5(n/5)
Algoritmo 5(n/5)
Algoritmo 5(n/5)
Algoritmo 5(n/5)
Algoritmo 5(n/5)
else:
return

Si dica una versione di ricerca di Ternario e di Fibonacci di Algoritmo 5(n) e si scriva la relazione di ricorrenza.

1. *Esercizio.* **In ciascuno dei seguenti casi, indicare se vale che $f(n) = O(g(n))$, oppure se $f(n) = \Omega(g(n))$, oppure se $f(n) = \Theta(g(n))$ (in tal caso, si scriva inoltre che $f(n) = \Theta(g(n))$). Si giustifichi la risposta.**

(a) $f(n) = n - \sqrt{n}$, $g(n) = \frac{n}{2}$ $f(n) \in \Theta(g(n))$

(b) $f(n) = 3n\log n$, $g(n) = \frac{n(n+1)}{2}$ $f(n) \in O(g(n))$

(c) $f(n) = 3n + 5\sqrt{n}$, $g(n) = n - 3\sqrt{n}$ $f(n) \in \Theta(g(n))$

(d) $f(n) = n^3$, $g(n) = 3n\log n$ $f(n) \in \Omega(g(n))$

(e) $f(n) = 2\log n$, $g(n) = \log n^2$ $f(n) \in \Theta(g(n))$

(f) $f(n) = \sqrt{3n\log n}$, $g(n) = 3\log n$ $f(n) \in O(g(n))$

(g) $f(n) = 6n^2 + n\log n^2 + \sqrt{n}$, $g(n) = 6n^2 - 4n\log n^2 - 3\sqrt{n}$ $f(n) \in \Theta(g(n))$

(h) $f(n) = n^{0.01}$, $g(n) = n\log n^{0.01}$ $f(n) \in O(g(n))$

(i) $f(n) = 2^n$, $g(n) = 2^{n^2}$ $f(n) \in O(g(n))$

(j) $f(n) = 2^{2n}$, $g(n) = 2^{2n+1}$ $f(n) \in \Theta(g(n))$

(k) $f(n) = n^2$, $g(n) = n^{\frac{3}{2}}$ $f(n) \in \Omega(g(n))$

(l) $f(n) = n^{\frac{3}{2}}$, $g(n) = n^2$ $f(n) \in O(g(n))$

(m) $f(n) = n^2$, $g(n) = n^2$ $f(n) \in \Theta(g(n))$

(n) $f(n) = n^2 - n^{0.5}$, $g(n) = n^2 + n^{0.5}$ $f(n) \in \Theta(g(n))$

(o) $f(n) = 3n^{\frac{3}{2}}$, $g(n) = 3n^{\frac{3}{2}}$ $f(n) \in \Theta(g(n))$

2. *Esercizio.* **Provare le seguenti due affermazioni, **utilizzando** le equazioni centrali:**

(a) $\lim_{n\to\infty} \frac{n^k}{2^n} = 0$, $k > 0$

(b) $\lim_{n\to\infty} \frac{\log n}{n^k} = 0$, $k > 0$

Vale che $\lim_{n\to\infty} \frac{n^k}{2^n} = \Theta(\frac{\log n}{n^k})$. Giustificare la risposta.

3. *Esercizio.* **Provare le seguenti due affermazioni, **utilizzando** le equazioni centrali:**

(a) $\lim_{n\to\infty} \frac{\log n}{n^k} = 0$, $k > 0$

(b) $\lim_{n\to\infty} \frac{n^k}{2^n} = 0$, $k > 0$

Vale che $\lim_{n\to\infty} \frac{\log n}{n^k} = \Theta(\frac{n^k}{2^n})$. Giustificare la risposta.

4. *Esercizio.* **Per ciascuna delle seguenti coppie di funzioni f e g (scrive costanti c e N_0 e n_0 per cui $f(n) \leq c \cdot g(n)$ per ogni $n \geq n_0$.**

(a) $f(n) \in 3n^2 + 3n$ $g(n) = n^2$

(b) $f(n) \in 3n\log n + 4\sqrt{n}$ $g(n) = n\sqrt{n}$

(c) $f(n) = n^2 + n + 1$ $g(n) = 3n^2$

(d) $f(n) = 3n + 4\sqrt{n}$ $g(n) = n^2$

(e) $f(n) = 3n + 4\sqrt{n}$ $g(n) = n$

(f) $f(n) = n^2 + n + 1$ $g(n) = n^3$

(g) $f(n) = 3n\log n + 4\sqrt{n}$ $g(n) = 3n\log n$

(h) $f(n) = 3n^2 + 4\sqrt{n}$ $g(n) = n^2$

(i) $f(n) = n^2$ $g(n) = n^3$

(j) $f(n) = n^2 - n^{0.5}$ $g(n) = n^2 + n^{0.5}$

(k) $f(n) = 3n^{\frac{3}{2}}$ $g(n) = 3n^{\frac{3}{2}}$

Esercizio 5. Completa l'albero binario

Algoritmo 5(n):
if n > 0:
print(n)
Algoritmo 5(n/5)
Algoritmo 5(n/5)
Algoritmo 5(n/5)
Algoritmo 5(n/5)
Algoritmo 5(n/5)
else:
return

Si dica una versione di ricerca di Ternario e di Fibonacci di Algoritmo 5(n) e si scriva la relazione di ricorrenza.

Esercizio 4. Completa l'albero binario

Algoritmo 4(n):
if n > 0:
print(n)
Algoritmo 4(n/4)
Algoritmo 4(n/4)
Algoritmo 4(n/4)
Algoritmo 4(n/4)
else:
return

Si dica una versione di ricerca di Ternario e di Fibonacci di Algoritmo 4(n) e si scriva la relazione di ricorrenza.

Esercizio 3. Completa l'albero binario

Algoritmo 3(n):
if n > 0:
print(n)
Algoritmo 3(n/3)
Algoritmo 3(n/3)
Algoritmo 3(n/3)
else:
return

Si dica una versione di ricerca di Ternario e di Fibonacci di Algoritmo 3(n) e si scriva la relazione di ricorrenza.

Esercizio 2. Completa l'albero binario

Algoritmo 2(n):
if n > 0:
print(n)
Algoritmo 2(n/2)
Algoritmo 2(n/2)
else:
return

Si dica una versione di ricerca di Ternario e di Fibonacci di Algoritmo 2(n) e si scriva la relazione di ricorrenza.

Esercizio 1. Completa l'albero binario

Algoritmo 1(n):
if n > 0:
print(n)
Algoritmo 1(n-1)
Algoritmo 1(n-1)
else:
return

Si dica una versione di ricerca di Ternario e di Fibonacci di Algoritmo 1

ESERCIZI 2 P.1

- Si può fare in $n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots + 1 = 2n - \log n - 1$
- Ma è meno di $2n$ quindi, se $\frac{n}{2}$ è pari, $\frac{n}{2}$ è pari, $\frac{n}{4}$ è pari, $\frac{n}{8}$ è pari, $\frac{n}{16}$ è pari, $\frac{n}{32}$ è pari, $\frac{n}{64}$ è pari, $\frac{n}{128}$ è pari, $\frac{n}{256}$ è pari, $\frac{n}{512}$ è pari, $\frac{n}{1024}$ è pari, $\frac{n}{2048}$ è pari, $\frac{n}{4096}$ è pari, $\frac{n}{8192}$ è pari, $\frac{n}{16384}$ è pari, $\frac{n}{32768}$ è pari, $\frac{n}{65536}$ è pari, $\frac{n}{131072}$ è pari, $\frac{n}{262144}$ è pari, $\frac{n}{524288}$ è pari, $\frac{n}{1048576}$ è pari, $\frac{n}{2097152}$ è pari, $\frac{n}{4194304}$ è pari, $\frac{n}{8388608}$ è pari, $\frac{n}{16777216}$ è pari, $\frac{n}{33554432}$ è pari, $\frac{n}{67108864}$ è pari, $\frac{n}{134217728}$ è pari, $\frac{n}{268435456}$ è pari, $\frac{n}{536870912}$ è pari, $\frac{n}{1073741824}$ è pari, $\frac{n}{2147483648}$ è pari, $\frac{n}{4294967296}$ è pari, $\frac{n}{8589934592}$ è pari, $\frac{n}{17179869184}$ è pari, $\frac{n}{34359738368}$ è pari, $\frac{n}{68719476736}$ è pari, $\frac{n}{137438953472}$ è pari, $\frac{n}{274877906944}$ è pari, $\frac{n}{549755813888}$ è pari, $\frac{n}{1099511627776}$ è pari, $\frac{n}{2199023255552}$ è pari, $\frac{n}{4398046511104}$ è pari, $\frac{n}{8796093022208}$ è pari, $\frac{n}{17592186044416}$ è pari, $\frac{n}{35184372088832}$ è pari, $\frac{n}{70368744177664}$ è pari, $\frac{n}{140737488355328}$ è pari, $\frac{n}{281474976710656}$ è pari, $\frac{n}{562949953421312}$ è pari, $\frac{n}{1125899906842624}$ è pari, $\frac{n}{2251799813685248}$ è pari, $\frac{n}{4503599627370496}$ è pari, $\frac{n}{9007199254740992}$ è pari, $\frac{n}{18014398509481984}$ è pari, $\frac{n}{36028797018963968}$ è pari, $\frac{n}{72057594037927936}$ è pari, $\frac{n}{144115188075855872}$ è pari, $\frac{n}{288230376151711744}$ è pari, $\frac{n}{576460752303423488}$ è pari, $\frac{n}{1152921504606846976}$ è pari, $\frac{n}{2305843009213693952}$ è pari, $\frac{n}{4611686018427387904}$ è pari, $\frac{n}{9223372036854775808}$ è pari, $\frac{n}{18446744073709551616}$ è pari, $\frac{n}{36893488147419103232}$ è pari, $\frac{n}{73786976294838206464}$ è pari, $\frac{n}{147573952589676412928}$ è pari, $\frac{n}{295147905179352825856}$ è pari, $\frac{n}{590295810358705651712}$ è pari, $\frac{n}{1180591620717411303424}$ è pari, $\frac{n}{2361183241434822606848}$ è pari, $\frac{n}{4722366482869645213696}$ è pari, $\frac{n}{9444732965739290427392}$ è pari, $\frac{n}{18889465931478580854784}$ è pari, $\frac{n}{37778931862957161709568}$ è pari, $\frac{n}{75557863725914323419136}$ è pari, $\frac{n}{151115727451828646838272}$ è pari, $\frac{n}{302231454903657293676544}$ è pari, $\frac{n}{604462909807314587353088}$ è pari, $\frac{n}{1208925819614629174706176}$ è pari, $\frac{n}{2417851639229258349412352}$ è pari, $\frac{n}{4835703278458516698824704}$ è pari, $\frac{n}{9671406556917033397649408}$ è pari, $\frac{n}{19342813113834066795298816}$ è pari, $\frac{n}{38685626227668133590597632}$ è pari, $\frac{n}{77371252455336267181195264}$ è pari, $\frac{n}{154742504910672534362390528}$ è pari, $\frac{n}{309485009821345068724781056}$ è pari, $\frac{n}{618970019642690137449562112}$ è pari, $\frac{n}{1237940039285380274899124224}$ è pari, $\frac{n}{2475880078570760549798248448}$ è pari, $\frac{n}{4951760157141521099596496896}$ è pari, $\frac{n}{9903520314283042199192993792}$ è pari, $\frac{n}{19807040628566084398385987584}$ è pari, $\frac{n}{39614081257132168796771975168}$ è pari, $\frac{n}{79228162514264337593543950336}$ è pari, $\frac{n}{158456325028528675187087900672}$ è pari, $\frac{n}{316912650057057350374175801344}$ è pari, $\frac{n}{633825300114114700748351602688}$ è pari, $\frac{n}{1267650600228229401496703205376}$ è pari, $\frac{n}{2535301200456458802993406410752}$ è pari, $\frac{n}{5070602400912917605986812821504}$ è pari, $\frac{n}{10141204801825835211973625643008}$ è pari, $\frac{n}{20282409603651670423947251286016}$ è pari, $\frac{n}{40564819207303340847894502572032}$ è pari, $\frac{n}{81129638414606681695789005144064}$ è pari, $\frac{n}{162259276829213363391578010288128}$ è pari, $\frac{n}{324518553658426726783156020576256}$ è pari, $\frac{n}{649037107316853453566312041152512}$ è pari, $\frac{n}{1298074214633706907132624082305024}$ è pari, $\frac{n}{2596148429267413814265248164610048}$ è pari, $\frac{n}{5192296858534827628530496329220096}$ è pari, $\frac{n}{10384593717069655257060992658440192}$ è pari, $\frac{n}{20769187434139310514121985316880384}$ è pari, $\frac{n}{41538374868278621028243970633760768}$ è pari, $\frac{n}{83076749736557242056487941267521536}$ è pari, $\frac{n}{166153499473114484112975882535043072}$ è pari, $\frac{n}{332306998946228968225951765070086144}$ è pari, $\frac{n}{664613997892457936451903530140172288}$ è pari, $\frac{n}{1329227995784915872903807060280344576}$ è pari, $\frac{n}{2658455991569831745807614120560689152}$ è pari, $\frac{n}{5316911983139663491615228241121378304}$ è pari, $\frac{n}{10633823966279326983230456482242756608}$ è pari, $\frac{n}{21267647932558653966460912964485513216}$ è pari, $\frac{n}{42535295865117307932921825928971026432}$ è pari, $\frac{n}{85070591730234615865843651857942052864}$ è pari, $\frac{n}{170141183460469231731687303715884105728}$ è pari, $\frac{n}{340282366920938463463374607431768211456}$ è pari, $\frac{n}{680564733841876926926749214863536422912}$ è pari, $\frac{n}{1361129467683753853853498429727072845824}$ è pari, $\frac{n}{2722258935367507707706996859454145691648}$ è pari, $\frac{n}{5444517870735015415413993718908291383296}$ è pari, $\frac{n}{10889035741470030830827987437816582766592}$ è pari, $\frac{n}{21778071482940061661655974875633165533184}$ è pari, $\frac{n}{43556142965880123323311949751266331066368}$ è pari, $\frac{n}{87112285931760246646623899502532662132736}$ è pari, $\frac{n}{174224571863520493293247799005065324265472}$ è pari, $\frac{n}{348449143727040986586495598010130648530944}$ è pari, $\frac{n}{696898287454081973172991196020261297061888}$ è pari, $\frac{n}{1393796574908163946345982392040522594123776}$ è pari, $\frac{n}{2787593149816327892691964784081045188247552}$ è pari, $\frac{n}{5575186299632655785383929568162090376495104}$ è pari, $\frac{n}{11150372599265311570767859136324180752990208}$ è pari, $\frac{n}{22300745198530623141535718272648361505980416}$ è pari, $\frac{n}{44601490397061246283071436545296723011960832}$ è pari, $\frac{n}{89202980794122492566142873090593446023921664}$ è pari, $\frac{n}{178405961588244985132285746181186892047843328}$ è pari, $\frac{n}{356811923176489970264571492362373784095686656}$ è pari, $\frac{n}{713623846352979940529142984724747568191373312}$ è pari, $\frac{n}{1427247692705959881058285969449495136382746624}$ è pari, $\frac{n}{2854495385411919762116571938898990272765493248}$ è pari, $\frac{n}{5708990770823839524233143877797980545530986496}$ è pari, $\frac{n}{11417981541647679048466287755595961091061972992}$ è pari, $\frac{n}{22835963083295358096932575511191922182123945984}$ è pari, $\frac{n}{45671926166590716193865151022383844364247891968}$ è pari, $\frac{n}{91343852333181432387730302044767688728495783936}$ è pari, $\frac{n}{182687704666362864775460604089535377456991567872}$ è pari, $\frac{n}{365375409332725729550921208179070754913983135744}$ è pari, $\frac{n}{730750818665451459101842416358141509827966271488}$ è pari, $\frac{n}{1461501637330902918203684832716283019655932542976}$ è pari, $\frac{n}{2923003274661805836407369665432566039311865085952}$ è pari, $\frac{n}{5846006549323611672814739330865132078623730171904}$ è pari, $\frac{n}{11692013098647223345629478661730264157247460343808}$ è pari, $\frac{n}{23384026197294446691258957323460528314494920687616}$ è pari, $\frac{n}{46768052394588893382517914646921056628989841375232}$ è pari, $\frac{n}{93536104789177786765035829293842113257979682750464}$ è pari, $\frac{n}{187072209578355573530071658587684226515959365500928}$ è pari, $\frac{n}{374144419156711147060143317175368453031918731001856}$ è pari, $\frac{n}{748288838313422294120286634350736906063837462003712}$ è pari, $\frac{n}{1496577676626844588240573268701473812127674924007424}$ è pari, $\frac{n}{2993155353253689176481146537402947624255349848014848}$ è pari, $\frac{n}{5986310706507378352962293074805895248510699696029696}$ è pari, $\frac{n}{11972621413014756705924586149611790497021399392059392}$ è pari, $\frac{n}{23945242826029513411849172299223580994042798784118784}$ è pari, $\frac{n}{47890485652059026823698344598447161988085597568237568}$ è pari, $\frac{n}{95780971304118053647396689196894323976171195136475136}$ è pari, $\frac{n}{191561942608236107294793378393788647952342390272950272}$ è pari, $\frac{n}{383123885216472214589586756787577295904684780545900544}$ è pari, $\frac{n}{766247770432944429179173513575154591809369561091801088}$ è pari, $\frac{n}{1532495540865888858358347027150309183618739122183602176}$ è pari, $\frac{n}{3064991081731777716716694054300618367237478244367204352}$ è pari, $\frac{n}{6129982163463555433433388108601236734474956488734408704}$ è pari, $\frac{n}{12259964326927110866866776217202473468949912977468817408}$ è pari, $\frac{n}{24519928653854221733733552434404946937899825954937634816}$ è pari, $\frac{n}{49039857307708443467467104868809893875799651909875269632}$ è pari, $\frac{n}{98079714615416886934934209737619787751599303819750539264}$ è pari, $\frac{n}{196159429230833773869868419475239575503198607639501078528}$ è pari, $\frac{n}{392318858461667547739736838950479151006397215279002157056}$ è pari, $\frac{n}{784637716923335095479473677900958302012794430558004314112}$ è pari, $\frac{n}{1569275433846670190958947355801916604025588861116008628224}$ è pari, $\frac{n}{3138550867693340381917894711603833208051177722232017256448}$ è pari, $\frac{n}{6277101735386680763835789423207666416102355444464034512896}$ è pari, $\frac{n}{12554203470773361527671578846415332832204710888928069025792}$ è pari, $\frac{n}{25108406941546723055343157692830665664409421777856138051584}$ è pari, $\frac{n}{50216813883093446110686315385661331328818843555712276103168}$ è pari, $\frac{n}{100433627766186892221372630771322662657637687111424552206336}$ è pari, $\frac{n}{200867255532373784442745261542645325315275374222849104412672}$ è pari, $\frac{n}{401734511064747568885490523085290650630550748445698208825344}$ è pari, $\frac{n}{803469022129495137770981046170581301261101496891396417650688}$ è pari, $\frac{n}{1606938044258990275541962092341162602522202993782792835301376}$ è pari, $\frac{n}{3213876088517980551083924184682325205044405987565585670602752}$ è pari, $\frac{n}{6427752177035961102167848369364650410088811975131171341205504}$ è pari, $\frac{n}{12855504354071922204335696738729300820177623950262342682411008}$ è pari, $\frac{n}{25711008708143844408671393477458601640355247900524685364822016}$ è pari, $\frac{n}{51422017416287688817342786954917203280710495801049370729644032}$ è pari, $\frac{n}{102844034832575377634685573909834406561420991602098741459288064}$ è pari, $\frac{n}{205688069665150755269371147819668813122841983204197482918576128}$ è pari, $\frac{n}{411376139330301510538742295639337626245683966408394965837152256}$ è pari, $\frac{n}{822752278660603021077484591278675252491367932816789931674304512}$ è pari, $\frac{n}{1645504557321206042154969182557350504982735865633579863348609024}$ è pari, $\frac{n}{3291009114642412084309938365114701009965471731267159726697218048}$ è pari, $\frac{n}{6582018229284824168619876730229402019930943462534319453394436096}$ è pari, $\frac{n}{13164036458569648337239753460458804039861886925068638906788872192}$ è pari, $\frac{n}{26328072917139296674479506921917608079723773850137277813577744384}$ è pari, $\frac{n}{52656145834278593348959013843835216159447547700274555627155488768}$ è pari, $\frac{n}{105312291668557186697918027687670432318895095400549111254310977536}$ è pari, $\frac{n}{210624583337114373395836055375340864637790190801098222508621955072}$ è pari, $\frac{n}{421249166674228746791672110750681729275580381602196445017243910144}$ è pari, $\frac{n}{842498333348457493583344221501363458551160763204392890034487820288}$ è pari, $\frac{n}{1684996666696915987166688442938726917102321526408785780068975640576}$ è pari, $\frac{n}{3369993333393831974333376885877453834204643052817571560137951281152}$ è pari, $\frac{n}{6739986666787663948666753771754907668409286105635143120275902562304}$ è pari, $\frac{n}{13479973333575327897333507543509815336818572211270286240551805124608}$ è pari, $\frac{n}{26959946667150655794667015087019630673637144422540572481103610249216}$ è pari, $\frac{n}{53919893334301311589334030174039261347274288845081144962207220498432}$ è pari, $\frac{n}{107839786668602623178668060348078522694548577690162289924414440996864}$ è pari, $\frac{n}{215679573337205246357336120696157045389097155380324579848828881993728}$ è pari, $\frac{n}{431359146674410492714672241392314090778194310760649159697657763987456}$ è pari, $\frac{n}{862718293348820985429344482784628181556388621521298319395315527974912}$ è pari, $\frac{n}{1725436586697641970858688965569256363112777243042596638790631055949824}$ è pari, $\frac{n}{3450873173395283941717377931138512726225554485085193277581262111899648}$ è pari, $\frac{n}{6901746346790567883434755862277025452451108970170386555162524223799296}$ è pari, $\frac{n}{13803492693581135766869511724554050904902217940340773110325048447598592}$ è pari, $\frac{n}{27606985387162271533739023449108101809804435880681546220650096895197184}$ è pari, $\frac{n}{55213970774324543067478046898216203619608871761363092441300193790394368}$ è pari, $\frac{n}{110427941548649086134956093796432407239217743522726184882600387580788736}$ è pari, $\frac{n}{220855883097298172269912187592864814478435487045452369765200775161577472}$ è pari, $\frac{n}{441711766194596344539824375185729628956870974090904739530401550323154944}$ è pari, $\frac{n}{883423532389192689079648750371459257913741948181809479060803100646309888}$ è pari, $\frac{n}{1766847064778385378159297500742918515827483896363618958121606201292619776}$ è pari, $\frac{n}{3533694129556770756318595001485837031654967792727237916243212402585239552}$ è pari, $\frac{n}{7067388259113541512637190002971674063309935585454475832486424805170479104}$ è pari, $\frac{n}{14134776518227083025274380005943348126619871170908951664972849610340958208}$ è pari, $\frac{n}{28269553036454166050548760011886696253239742341817903329945699220681916416}$ è pari, $\frac{n}{56539106072908332101097520023773392506479484683635806659891398441363832832}$ è pari, $\frac{n}{113078212145816664202195040047546785012958969367271613319782796882727665664}$ è pari, $\frac{n}{226156424291633328404390080095093570025917938734543226639565593765455331328}$ è pari, $\frac{n}{452312848583266656808780160190187140051835877469086453279131187530910662656}$ è pari, $\frac{n}{904625697166533313617560320380374280103671754938172906558262375061821325312}$ è pari, $\frac{n}{1809251394333066627235120640760748560207343509876345813116524750123642650624}$ è pari, $\frac{n}{3618502788666131254470241281521497120414687019752691626233049500247285301248}$ è pari, $\frac{n}{7237005577332262508940482563042994240829374039505383252466099000494570602496}$ è pari, $\frac{n}{14474011154664525017880965126085988481658748079010766504932198000989141204992}$ è pari, $\frac{n}{28948022309329048035761930252171976963317496158021533009864396001978282409984}$ è pari, $\frac{n}{57896044618658097071523860504343953926634992316043066019728792003956564819968}$ è pari, $\frac{n}{115792089237316195423047721008687907853269984632086132039457584007913129639936}$ è pari, $\frac{n}{231584178474632390847095442017375815706539969264172264078915168015826259279872}$ è pari, $\frac{n}{463168356949264781694190884034751631413079938528344528157830336031652518559744}$ è pari, $\frac{n}{926336713898529563388381768069503262826159877056689056315660672063305037119488}$ è pari, $\frac{n}{1852673427797059126777763536138006525652319754113378112631321344126610074238976}$ è pari, $\frac{n}{3705346855594118253555527072276013051304639508226756225262642688253220148477952}$ è pari, $\frac{n}{7410693711188236507111054144552026102609279016453512450525285376506440296955904}$ è pari, $\frac{n}{14821387422376473014222108289104052205218558032907024901050570753012880593911808}$ è pari, $\frac{n}{29642774844752946028444216578208104410437116065814049802101141506025761187823616}$ è pari, $\frac{n}{59285549689505892056888433156416208820874232131628099604202283012051522375647232}$ è pari, $\frac{n}{118571099379011784113776866312832417641748464263256199208404566024103044751294464}$ è pari, $\frac{n}{237142198758023568227553732625664835283496928526512398416809132048206089502588928}$ è pari, $\frac{n}{474284397516047136455107465251329670566993857053024796833618264096412179005177856}$ è pari, $\frac{n}{948568795032094272910214930502659341133987714106049593667236528192824358010355712}$ è pari, $\frac{n}{1897137590064188545820429861005318682267975428212099187334473056385648716020711424}$ è pari, $\frac{n}{3794275180128377091640859722010637364535950856424198374668946112771297432041422848}$ è pari, $\frac{n}{7588550360256754183281719444021274729071701712848396749337892225542594864082845696}$ è pari, $\frac{n}{15177100720513508366563438888042549458143403425696793498675784451085189728165691392}$ è pari, $\frac{n}{30354201441027016733126877776085098916286806851393586997351568902170379456331382784}$ è pari, $\frac{n}{60708402882054033466253755552170197832573613702787173994703137804340758912662765568}$ è pari, $\frac{n}{121416805764108066932507511104340395665147227405574347989406275608681517825325531136}$ è pari, $\frac{n}{242833611528216133865015022208680791330294454811148695978812551217363035650651062272}$ è pari, $\frac{n}{485667223056432267730030044417361582660588909622297391957625102434726071301302124544}$ è pari, $\frac{n}{971334446112864535460060088834723165321177819244594783915250204869452142602604249088}$ è pari, $\frac{n}{1942668892225729070920120177669446330642355638489189567830500409738904285205208498176}$ è pari, $\frac{n}{3885337784451458141840240355338892661284711276978379135661000819477808570410416996352}$ è pari, $\frac{n}{7770675568902916283680480710677785322569422553956758271322001638955617140820833992704}$ è pari, $\frac{n}{15541351137805832567360961421355570645138845107913516542644003277911234281641667985408}$ è pari, $\frac{n}{31082702275611665134721922842711141290277690215827033085288006555822468563283335970816}$ è pari, $\frac{n}{62165404551223330269443845685422282580555380431654066170576013111644937126566671941632}$ è pari, $\frac{n}{124330809102446660538887691370844565161110760863308132341152026223289874253133343883264}$ è pari, $\frac{n}{248661618204893321077775382741689130322221521726616264682304052446579748506266687766528}$ è pari, $\frac{n}{497323236409786642155550765483378260644443043453232529364608104893159497012533375533056}$ è pari, $\frac{n}{994646472819573284311101530966756521288886086906465058729216209786318994025066751066112}$ è pari, $\frac{n}{1989292945639146568622203061933513042577772173812930117458432419572637988050133502132224}$ è pari, $\frac{n}{3978585891278293137244406123867026085155544347625860234916864839145275976100267004264448}$ è pari, $\frac{n}{7957171782556586274488812247734052170311088695251720469833729678290551952200534008528896}$ è pari, $\frac{n}{15914343565113172548977624495468104340622177390503440939667459356581103904401068017057792}$ è pari, $\frac{n}{31828687130226345097955248990936208681244354781006881879334918713162207808802136034115584}$ è pari, $\frac{n}{63657374260452690195910497981872417362488709562013763758669837426324415617604272068231168}$ è pari, $\frac{n}{127314748520905380391820995963744834724977419124027527517339674852648831235208544136462336}$ è pari, $\frac{n}{254629497041810760783641991927489669449954838248055055034679349705297662470417088272924672}$ è pari, $\frac{n}{509258994083621521567283983854979338899909676496110110069358699410595324940834176545849344}$ è pari, $\frac{n}{1018517988167243043134567967709958677799819352992220220138717398821190649881668353091698688}$ è pari, $\frac{n}{2037035976334486086269135935419916355599638705984440440277434797642381299763336706183397376}$ è pari, $\frac{n}{4074071952668972172538271870839832711199277411968880880554869595284762599526673412366794752}$ è pari, $\frac{n}{8148143905337944345076543741679665422398554823937761761109739190569525199053346824733589504}$ è pari, $\frac{n}{16296287810675888690153087483359330844797109647875523522219478381139050398106693649467179008}$ è pari, $\frac{n}{32592575621351777380306174966718661689594219295751047044438956762278100796213387298934358016}$ è pari, $\frac{n}{65185151242703554760612349933437323379188438591502094088877913524556201592426774597868716032}$ è pari, $\frac{n}{130370302485407109521224699866874646758376877183004188177755827049112403184853549195737432064}$ è pari, $\frac{n}{260740604970814219042449399733749293516753754366008376355511654098224806369707098391474864128}$ è pari, $\frac{n}{521481209941628438084898799467498587033507508732016752711023308196449612739414196782949728256}$ è pari, $\frac{n}{1042962419883256876169797598934997174067015017464033505422046616392899225478828393565899456512}$ è pari, $\frac{n}{2085924839766513752339595197869994348134030034928067010844093232785798450957656787131798913024}$ è pari, $\frac{n}{4171849679533027504679190395739988696268060069856134021688186465571596901915313574263597826048}$ è pari, $\frac{n}{8343699359066055009358380791479977392536120139712268043376372931143193803830627148527195652096}$ è pari, $\frac{n}{16687398718132110018716761582959954785072240279424536086752745862286387607661254297054391304192}$ è pari, $\frac{n}{33374797436264220037433523165919909570144480558849072173505491724572775215322508594108782608384}$ è pari, $\frac{n}{66749594872528440074867046331839819140288961117698144347010983449145550430645017188217565216768}$ è pari, $\frac{n}{133499189745056880149734092663679638280577922235396288694021966898291100861290034376435130433536}$ è pari, $\frac{n}{266998379490113760299468185327359276561155844470792577388043933796582201722580068752870260867072}$ è pari, $\frac{n}{533996758980227520598936370654718553122311688941585154776087867593164403445160137505740521734144}$ è pari, $\frac{n}{1067993517960455041197872741309437106244623377883170309552175735186328806890320275011481043468288}$ è pari, $\frac{n}{2135987035920910082395745482618874212489246755766340619104351470372657613780640550022962086936576}$ è pari, $\frac{n}{4271974071841820164791490965237748424978493511532681238208702940745315227561281100045924173873152}$ è pari, $\frac{n}{8543948143683640329582981930475496849956987023065362476417405881490630455122562200091848347746304}$ è pari, $\frac{n}{17087896287367280659165963860950993699913974046130724952834811762981260910245124400183696695492608}$ è pari, $\frac{n}{34175792574734561318331927721901987399827948092261449905669623525962521820490248800367393390985216}$ è pari, $\frac{n}{68351585149469122636663855443803974799655896184522899811339247051925043640980497600734786781970432}$ è pari, $\frac{n}{136703170298938245273327710887607949599311792369045799622678494103850087281960995201469573563940864}$ è pari, $\frac{n}{273406340597876490546655421775215899198623584738091599245356988207700174563921990402939147127881728}$ è pari, $\frac{n}{546812681195752981093310843550431798397247169476183198490713976415400349127843980805878294255763456}$ è pari, $\frac{n}{1093625362391505962186621687100863596794494338952366396981427952830800698255687961611756588511526912}$ è pari, $\frac{n}{2187250724783011924373243374201727193588988677904732793962855905661601396511375923223513177023053824}$ è pari, $\frac{n}{4374501449566023848746486748403454387177977355809465587925711811323202793022751846447026354046107648}$ è pari, $\frac{n}{8749002899132047697492973496806908774355954711618931175851423622646405586045503692894052708092215296}$ è pari, $\frac{n}{17498005798264095394985946993613817548711909423237862351702847245292811172091007385788105416184430592}$ è pari, $\frac{n}{34996011596528190789971893987227635097423818846475724703405694490585622344182014771576210832368861184}$ è pari, $\frac{n}{69992023193056381579943787974455270194847637692951449406811388981171244688364029543152421664737722368}$ è pari, $\frac{n}{139984046386112763159887575948910540389695275385902898813622777962342489376728059086304843329475444736}$ è pari, $\frac{n}{279968092772225526319775151897821080779390550771805797627245555924684978753456118172609686658950889472}$ è pari, $\frac{n}{559936185544451052639550303795642161558781101543611595254491111849369957506912236345219373317901778944}$ è pari, $\frac{n}{1119872371088902105279100607591284323117562203087223190508982223698739915013824472690438746635803557888}$ è pari, $\frac{n}{2239744742177804210558201215182568646235124406174446381017964447397479830027648945380877493271607115776}$ è pari, $\frac{n}{4479489484355608421116402430365137292470248812348892762035928894794959660055297890761754986543214231552}$ è pari, $\frac{n}{8958978968711216842232804860730274584940497624697785524071857789589919320110595781523509973086428463104}$ è pari, $\frac{n}{17917957937422433684465609721460549169880995249395570048143715579179838640221191563047019946172856926208}$ è pari, $\frac{n}{35835915874844867368931219442921098339761990498791140096287431158359677280442383126094039892345713852416}$ è pari, $\frac{n}{71671831749689734737862438885842196679523980997582280192574862316719354560884766252188079784691427704832}$ è pari, $\frac{n}{143343663499379469475724877771684393359047961995164560385149724633438709121769532504376159569382855409664}$ è pari, $\frac{n}{286687326998758938951449755543368786718095923990329120770299449266877418243539064008752319138765710819328}$ è pari, $\frac{n}{573374653997517877902899511086737573436191847980658241540598898533754836487078128017504638277531421638656}$ è pari, $\frac{n}{1146749307995035755805799022173475146872383695961316483081197797507509672974156256035009276555062843277312}$ è pari, $\frac{n}{2293498615990071511611598044346950293744767391922632966162395595015019345948312512070018553110125686554624}$ è pari, $\frac{n}{4586997231980143023223196088693900587489534783845265932324791190030038691896625024140037106220251373109248}$ è pari, $\frac{n}{9173994463960286046446392177387801174979069567690531864649582380060077383793250048280074212440502746218496}$ è pari, $\frac{n}{18347988927920572092892784354775602349958139135381063729299164760120154767586500096560148424881005492436992}$ è pari, $\frac{n}{36695977855841144185785568709551204699916278270762127458598329520240309535173000193120296849762010984873984}$ è pari, $\frac{n}{73391955711682288371571137419102409399832556541524254917196659040480619070346000386240593699524021969747968}$ è pari, $\frac{n}{146783911423364576743142274838204818799665113083048509834393318080961238140692000772481187399048043939495936}$ è pari, $\frac{n}{293567822846729153486284549676409637599330226166097019668786636161922476281384001544962374798096087878991872}$ è pari, $\frac{n}{587135645693458306972569099352819275198660452332194039337573272323844952562768003089924749596192175757983744}$ è pari, $\frac{n}{1174271291386916613945138198705638550397320904664388078675146544647689905125536006179849499192384351515967488}$ è pari, $\frac{n}{2348542582773833227890276397411277100794641809328776157350293089295379810251072012359698998384768703031934976}$ è pari, $\frac{n}{4697085165547666455780552794822554201589283618657552314700586178590759620502144024719397996769537406063869952}$ è pari, $\frac{n}{9394170331095332911561105589645108403178567237315104629401172357181519241004288049438795993539074812127739904}$ è pari, $\frac{n}{18788340662190665823122211179290216806357134474630209258802344714363038482008576098877591987078149624255479808}$ è pari, $\frac{n}{37576681324381331646244422358580433612714268949260418517604689428726076964017152197755183974156299248510959616}$ è pari, $\frac{n}{75153362648762663292488844717160867225428537898520837035209378857452153928034304395510367948312598497021919232}$ è pari, $\frac{n}{150306725297525326584977689434321734450857075797041674070418757714904307856068608791020735896625196994043838464}$ è pari, $\frac{n}{300613450595050653169955378868643468901714151594083348140837515429808615712137217582041471793250393988087676928}$ è pari, $\frac{n}{601226901190101306339910757737286937803428303188166696281675030859617231424274435164082943586500787976175353856}$ è pari, $\frac{n}{1202453802380202612679821515474573875606856606376333392563350061719234462848548870328165887173001575952350707712}$ è pari, $\frac{n}{2404907604760405225359643030949147751213713212752666785126700123438468925697097740656331774346003151904701415424}$ è pari, $\frac{n}{4809815209520810450719286061898295502427426425505333570253400246876937851394195481312663548692006303809402830848}$ è pari, $\frac{n}{9619630419041620901438572123796591004854852851010667140506800493753875702788390962625327097384012607618805661696}$ è pari, $\frac{n}{19239260838083241802877144247593182009709705702021334281013600987507751405576781925250654194768025215237611323392}$ è pari, $\frac{n}{38478521676166483605754288495186364019419411404042668562027201975015502811153563850501308389536050430475222646784}$ è pari, $\frac{n}{76957043352332967211508576990372728038838822808085337124054403950031005622307127701002616779072100860950445293568}$ è pari, $\frac{n}{153914086704665934423017153980745456077677645616170674248108807900062011244614255402005233558144201721900890587136}$ è pari, $\frac{n}{307828173409331868846034307961490912155355291232341348496217615800124022489228510804010467116288403443801781174272}$ è pari, $\frac{n}{615656346818663737692068615922981824310710582464682696992435231600248044978457021608020934232576806887603562348544}$ è pari, $\frac{n}{1231312693637327475384137231845963648621421164929365393984870463200496089956914043216041868465153613775207124697088}$ è pari, $\frac{n}{2462625387274654950768274463691927297242842329858730787969740926400992179913828086432083736930307227550414249394176}$ è pari, $\frac{n}{4925250774549309901536548927383854594485684659717461575939481852801984359827656172864167473860614455100828498788352}$ è pari, $\frac{n}{9850501549098619803073097854767709188971369319434923151878963705603968719655312345728334947721228910201656997576704}$ è pari, $\frac{n}{19701003098197239606146195709535418377942738638869846303757927411207937439310624691456669895442457820403313995153408}$ è pari, $\frac{n}{39402006196394479212292391419070836755885477277739692607515854822415874878621249382913339790884915640806627990306816}$ è pari, $\frac{n}{78804012392788958424584782838141673511770954555479385215031709644831749757242498765826679581769831281613255980613632}$ è pari, $\frac{n}{157608024785577916849169565676283347023541909110958770430063419289663499514484997531653359163539662563226511961227264}$ è pari, $\frac{n}{315216049571155833698339131352566694047083818221917540860126838579326998928969995063306718327079325126453023922454528}$ è pari, $\frac{n}{630432099142311667396678262705133388094167636443835081720253677158653997857939990126613436654158650252906047844909056}$ è pari, $\frac{n}{1260864198284623334793356525410266776188335272887670163440507354317307995715879980253226873308317300505812095689818112}$ è pari, $\frac{n}{2521728396569246669586713050820533552376670545775340326881014708634615991431759960506453746616634601011624191379636224}$ è pari, $\frac{n}{5043456793138493339173426101641067104753341091550680653762029417269231982863519921012907493233269202023248382759272448}$ è pari, $\frac{n}{10086913586276986678346852203282134209506682183101361307524058834538463965727039842025814986466538404046496765518544896}$ è pari, $\frac{n}{20173827172553973356693704406564268419013364366202722615048117669076927931454079684051629972933076808092993531037089792}$ è pari, $\frac{n}{40347654345107946713387408813128536838026728732405445230096235338153855862908159368103259945866153616185987062074179584}$ è pari, $\frac{n}{80695308690215893426774817626257073676053457464810890460192470676307711725816318736206519891732307232371974124148359168}$ è pari, $\frac{n}{161390617380431786853549635252514147352106914929621780920384941352615423451632637472413039783464614464743948248296718336}$ è pari, $\frac{n}{322781234760863573707099270505028294704213829859243561840769882705230846903265274944826079566929228929487896496593436672}$ è pari, $\frac{n}{645562469521727147414198541010056589408427659718487123681539765410461693806530549889652159133858457858975792993186873344}$ è pari, $\frac{n}{1291124939043454294828397082020113178816855319436974247363079530820923387613061099779304318267716915717951585986373746688}$ è pari, $\frac{n}{2582249878086908589656794164040226357633710638873948494726159061641846775226122199558608636535433831435903171972747493376}$ è pari, $\frac{n}{5164499756173817179313588328080452715267421277747896989452318123283693550452244399117217273070867662871806343945494986752}$ è pari, $\frac{n}{10328999512347634358627176656160905430534842555495793978904636246567387100904488798234434546141735325743612687890989973504}$ è pari, $\frac{n}{20657999024695268717254353312321810861069685110991587957809272493134774201808977596468869092283470651487225375781979947008}$ è pari, $\frac{n}{41315998049390537434508706624643621722139370221983175915618544986269548403617955192937738184566941302974450751563959894016}$ è pari, $\frac{n}{82631996098781074869017413249287243444278740443966351831237089972539096807235910385875476369133882605948901503127919788032}$ è pari, $\frac{n}{165263992197562149738034826498574486888557480887932703662474179945078193614471820771750952738267765211897803006255839576064}$ è pari, $\frac{n}{330527984395124299476069652997148973777114961775865407324948359890156387228943641543501905476535530423795606012511679152128}$ è pari, $\frac{n}{661055968790248598952139305994297947554229923551730814649896719780312774457887283086003810953071060847591212025023358304256}$ è pari, $\frac{n}{1322111937580497197904278611988595895108459847103461629299793439560625548915774566172007621906142121695182424050046716608512}$ è pari, $\frac{n}{2644223875160994395808557223977191790216919694206923258599586879121251097831549132344015243812284243390364848100093433217024}$ è pari, $\frac{n}{5288447750321988791617114447954383580433839388413846517199173758242502195663098264688030487624568486780729696200186866434048}$ è pari, $\frac{n}{10576895500643977583234228895908767160867678776827693034398347516485004391326196529376060975249136973561459392400373732868096}$ è pari, $\frac{n}{21153791001287955166468457791817534321735357553655386068796695032970008782652393058752121950498273947122918784800747465736192}$ è pari, $\frac{n}{42307582002575910332936915583635068643470715107310772137593390065940017565304786117504243900996547894245837569601494931472384}$ è pari, $\frac{n}{84615164005151820665873831167270137286941430214621544275186780131880035130609572235008487801993095788491675139202989862944768}$ è pari, $\frac{n}{169230328010303641331747662334540274573882860429243088550373560263760070261219144470016975603986191576983350278405979725889536}$ è pari, $\frac{n}{338460656020607282663495324669080549147765720858486177100747120527520140522438288940033951207972383153966700556811959451779072}$ è pari, $\frac{n}{676921312041214565326990649338161098295531441716972354201494241055040281044876577880067902415944766307933401113623918903558144}$ è pari, $\frac{n}{1353842624082429130653981298676322196591062883433944708402988482110080562089753155760135804831889532615866802227247837807116288}$ è pari, $\frac{n}{2707685248164858261307962597352644393182125766867889416805976964220161124179506311520271609663779065231733604454495675614232576}$ è pari, $\frac{n}{5415370496329716522615925194705288786364251533735778833611953928440322248359012623040543219327558130463467208908991351228465152}$ è pari, $\frac{n}{10830740992659433045231850389410577572728503067471557667223907856880644496718025246081086438655116260926934417817982702456930304}$ è pari, $\frac{n}{21661481985318866090463700778821155145456006134943115334447815713761288993436050492162172877310232521853868835635965404913860608}$ è pari, $\frac{n}{43322963970637732180927401557642310290912012269886230668895631427522577986872100984324345754620465043707737671271930809827721216}$ è pari, $\frac{n}{86645927941275464361854803115284620581824024539772461337791262855045155973744201968648691509240930087415475342543861619655442432}$ è pari, $\frac{n}{173291855882550928723709606230569241163648049079544922675582525710090311947488403937297383018481860174830950685087723239310884864}$ è pari, $\frac{n}{346583711765101857447419212461138482327296098159089845351165051420180623894976807874594766036963720349661901370175446478621769728}$ è pari, $\frac{n}{693167423530203714894838424922276964654592196318179690702330102840361247789953615749189532073927440699323802740350892957243539456}$ è pari, $\frac{n}{1386334847060407429789676849844553929309184392636359381404660205680722495579907231498379064147854881398647605

ESERCIZI 2 P.2

Si discuta, dopo aver correttamente impostato le relazioni ricorsive, quale tra i due tagli sia più efficiente, e perché.

$$T_{DI}(n) = \begin{cases} \Theta(1) & \text{se } n = 1 \\ \Theta(n) + T_{DI}(n-1) & \text{altrimenti} \end{cases}$$

$$T_{DI}(n) = \begin{cases} \Theta(1) & \text{se } n = 1 \\ \Theta(n) + T_{DI}(n/2) & \text{altrimenti} \end{cases}$$

$n = 2, 4, 8, \ldots$

$\Theta(n)$ è meno rilevante di $\Theta(\log n)$

13. *Esercizio:* Sia Sia $A = A[1] \ldots A[n]$ un vettore di n interi, tale che $A[i] \neq i$ per ogni i. L'indice $i \in \{1, \ldots, n\}$ è detto un indice fisso se $A[i] = i$. Si progetti un algoritmo che determina se A possiede un indice fisso. Si ha un solo A, e si richiede che l'algoritmo deterministico sia tecnica Divide et Impera almeno un valore di i per cui $A[i] = i$. Si analizzi la complessità di tempo dell'algoritmo proposto, giustificando le affermazioni fatte.

$A[p] \ldots A[q]$
$A[p] = p$
$A[p+1] = p+1$
$A[p+2] = p+2$
$A[p+3] = p+3$
$A[p+4] = p+4$
$A[p+5] = p+5$
$A[p+6] = p+6$
$A[p+7] = p+7$
$A[p+8] = p+8$
$A[p+9] = p+9$
$A[p+10] = p+10$
$A[p+11] = p+11$
$A[p+12] = p+12$
$A[p+13] = p+13$
$A[p+14] = p+14$
$A[p+15] = p+15$
$A[p+16] = p+16$
$A[p+17] = p+17$
$A[p+18] = p+18$
$A[p+19] = p+19$
$A[p+20] = p+20$
$A[p+21] = p+21$
$A[p+22] = p+22$
$A[p+23] = p+23$
$A[p+24] = p+24$
$A[p+25] = p+25$
$A[p+26] = p+26$
$A[p+27] = p+27$
$A[p+28] = p+28$
$A[p+29] = p+29$
$A[p+30] = p+30$
$A[p+31] = p+31$
$A[p+32] = p+32$
$A[p+33] = p+33$
$A[p+34] = p+34$
$A[p+35] = p+35$
$A[p+36] = p+36$
$A[p+37] = p+37$
$A[p+38] = p+38$
$A[p+39] = p+39$
$A[p+40] = p+40$
$A[p+41] = p+41$
$A[p+42] = p+42$
$A[p+43] = p+43$
$A[p+44] = p+44$
$A[p+45] = p+45$
$A[p+46] = p+46$
$A[p+47] = p+47$
$A[p+48] = p+48$
$A[p+49] = p+49$
$A[p+50] = p+50$
$A[p+51] = p+51$
$A[p+52] = p+52$
$A[p+53] = p+53$
$A[p+54] = p+54$
$A[p+55] = p+55$
$A[p+56] = p+56$
$A[p+57] = p+57$
$A[p+58] = p+58$
$A[p+59] = p+59$
$A[p+60] = p+60$
$A[p+61] = p+61$
$A[p+62] = p+62$
$A[p+63] = p+63$
$A[p+64] = p+64$
$A[p+65] = p+65$
$A[p+66] = p+66$
$A[p+67] = p+67$
$A[p+68] = p+68$
$A[p+69] = p+69$
$A[p+70] = p+70$
$A[p+71] = p+71$
$A[p+72] = p+72$
$A[p+73] = p+73$
$A[p+74] = p+74$
$A[p+75] = p+75$
$A[p+76] = p+76$
$A[p+77] = p+77$
$A[p+78] = p+78$
$A[p+79] = p+79$
$A[p+80] = p+80$
$A[p+81] = p+81$
$A[p+82] = p+82$
$A[p+83] = p+83$
$A[p+84] = p+84$
$A[p+85] = p+85$
$A[p+86] = p+86$
$A[p+87] = p+87$
$A[p+88] = p+88$
$A[p+89] = p+89$
$A[p+90] = p+90$
$A[p+91] = p+91$
$A[p+92] = p+92$
$A[p+93] = p+93$
$A[p+94] = p+94$
$A[p+95] = p+95$
$A[p+96] = p+96$
$A[p+97] = p+97$
$A[p+98] = p+98$
$A[p+99] = p+99$
$A[p+100] = p+100$
$A[p+101] = p+101$
$A[p+102] = p+102$
$A[p+103] = p+103$
$A[p+104] = p+104$
$A[p+105] = p+105$
$A[p+106] = p+106$
$A[p+107] = p+107$
$A[p+108] = p+108$
$A[p+109] = p+109$
$A[p+110] = p+110$
$A[p+111] = p+111$
$A[p+112] = p+112$
$A[p+113] = p+113$
$A[p+114] = p+114$
$A[p+115] = p+115$
$A[p+116] = p+116$
$A[p+117] = p+117$
$A[p+118] = p+118$
$A[p+119] = p+119$
$A[p+120] = p+120$
$A[p+121] = p+121$
$A[p+122] = p+122$
$A[p+123] = p+123$
$A[p+124] = p+124$
$A[p+125] = p+125$
$A[p+126] = p+126$
$A[p+127] = p+127$
$A[p+128] = p+128$
$A[p+129] = p+129$
$A[p+130] = p+130$
$A[p+131] = p+131$
$A[p+132] = p+132$
$A[p+133] = p+133$
$A[p+134] = p+134$
$A[p+135] = p+135$
$A[p+136] = p+136$
$A[p+137] = p+137$
$A[p+138] = p+138$
$A[p+139] = p+139$
$A[p+140] = p+140$
$A[p+141] = p+141$
$A[p+142] = p+142$
$A[p+143] = p+143$
$A[p+144] = p+144$
$A[p+145] = p+145$
$A[p+146] = p+146$
$A[p+147] = p+147$
$A[p+148] = p+148$
$A[p+149] = p+149$
$A[p+150] = p+150$
$A[p+151] = p+151$
$A[p+152] = p+152$
$A[p+153] = p+153$
$A[p+154] = p+154$
$A[p+155] = p+155$
$A[p+156] = p+156$
$A[p+157] = p+157$
$A[p+158] = p+158$
$A[p+159] = p+159$
$A[p+160] = p+160$
$A[p+161] = p+161$
$A[p+162] = p+162$
$A[p+163] = p+163$
$A[p+164] = p+164$
$A[p+165] = p+165$
$A[p+166] = p+166$
$A[p+167] = p+167$
$A[p+168] = p+168$
$A[p+169] = p+169$
$A[p+170] = p+170$
$A[p+171] = p+171$
$A[p+172] = p+172$
$A[p+173] = p+173$
$A[p+174] = p+174$
$A[p+175] = p+175$
$A[p+176] = p+176$
$A[p+177] = p+177$
$A[p+178] = p+178$
$A[p+179] = p+179$
$A[p+180] = p+180$
$A[p+181] = p+181$
$A[p+182] = p+182$
$A[p+183] = p+183$
$A[p+184] = p+184$
$A[p+185] = p+185$
$A[p+186] = p+186$
$A[p+187] = p+187$
$A[p+188] = p+188$
$A[p+189] = p+189$
$A[p+190] = p+190$
$A[p+191] = p+191$
$A[p+192] = p+192$
$A[p+193] = p+193$
$A[p+194] = p+194$
$A[p+195] = p+195$
$A[p+196] = p+196$
$A[p+197] = p+197$
$A[p+198] = p+198$
$A[p+199] = p+199$
$A[p+200] = p+200$
$A[p+201] = p+201$
$A[p+202] = p+202$
$A[p+203] = p+203$
$A[p+204] = p+204$
$A[p+205] = p+205$
$A[p+206] = p+206$
$A[p+207] = p+207$
$A[p+208] = p+208$
$A[p+209] = p+209$
$A[p+210] = p+210$
$A[p+211] = p+211$
$A[p+212] = p+212$
$A[p+213] = p+213$
$A[p+214] = p+214$
$A[p+215] = p+215$
$A[p+216] = p+216$
$A[p+217] = p+217$
$A[p+218] = p+218$
$A[p+219] = p+219$
$A[p+220] = p+220$
$A[p+221] = p+221$
$A[p+222] = p+222$
$A[p+223] = p+223$
$A[p+224] = p+224$
$A[p+225] = p+225$
$A[p+226] = p+226$
$A[p+227] = p+227$
$A[p+228] = p+228$
$A[p+229] = p+229$
$A[p+230] = p+230$
$A[p+231] = p+231$
$A[p+232] = p+232$
$A[p+233] = p+233$
$A[p+234] = p+234$
$A[p+235] = p+235$
$A[p+236] = p+236$
$A[p+237] = p+237$
$A[p+238] = p+238$
$A[p+239] = p+239$
$A[p+240] = p+240$
$A[p+241] = p+241$
$A[p+242] = p+242$
$A[p+243] = p+243$
$A[p+244] = p+244$
$A[p+245] = p+245$
$A[p+246] = p+246$
$A[p+247] = p+247$
$A[p+248] = p+248$
$A[p+249] = p+249$
$A[p+250] = p+250$
$A[p+251] = p+251$
$A[p+252] = p+252$
$A[p+253] = p+253$
$A[p+254] = p+254$
$A[p+255] = p+255$
$A[p+256] = p+256$
$A[p+257] = p+257$
$A[p+258] = p+258$
$A[p+259] = p+259$
$A[p+260] = p+260$
$A[p+261] = p+261$
$A[p+262] = p+262$
$A[p+263] = p+263$
$A[p+264] = p+264$
$A[p+265] = p+265$
$A[p+266] = p+266$
$A[p+267] = p+267$
$A[p+268] = p+268$
$A[p+269] = p+269$
$A[p+270] = p+270$
$A[p+271] = p+271$
$A[p+272] = p+272$
$A[p+273] = p+273$
$A[p+274] = p+274$
$A[p+275] = p+275$
$A[p+276] = p+276$
$A[p+277] = p+277$
$A[p+278] = p+278$
$A[p+279] = p+279$
$A[p+280] = p+280$
$A[p+281] = p+281$
$A[p+282] = p+282$
$A[p+283] = p+283$
$A[p+284] = p+284$
$A[p+285] = p+285$
$A[p+286] = p+286$
$A[p+287] = p+287$
$A[p+288] = p+288$
$A[p+289] = p+289$
$A[p+290] = p+290$
$A[p+291] = p+291$
$A[p+292] = p+292$
$A[p+293] = p+293$
$A[p+294] = p+294$
$A[p+295] = p+295$
$A[p+296] = p+296$
$A[p+297] = p+297$
$A[p+298] = p+298$
$A[p+299] = p+299$
$A[p+300] = p+300$
$A[p+301] = p+301$
$A[p+302] = p+302$
$A[p+303] = p+303$
$A[p+304] = p+304$
$A[p+305] = p+305$
$A[p+306] = p+306$
$A[p+307] = p+307$
$A[p+308] = p+308$
$A[p+309] = p+309$
$A[p+310] = p+310$
$A[p+311] = p+311$
$A[p+312] = p+312$
$A[p+313] = p+313$
$A[p+314] = p+314$
$A[p+315] = p+315$
$A[p+316] = p+316$
$A[p+317] = p+317$
$A[p+318] = p+318$
$A[p+319] = p+319$
$A[p+320] = p+320$
$A[p+321] = p+321$
$A[p+322] = p+322$
$A[p+323] = p+323$
$A[p+324] = p+324$
$A[p+325] = p+325$
$A[p+326] = p+326$
$A[p+327] = p+327$
$A[p+328] = p+328$
$A[p+329] = p+329$
$A[p+330] = p+330$
$A[p+331] = p+331$
$A[p+332] = p+332$
$A[p+333] = p+333$
$A[p+334] = p+334$
$A[p+335] = p+335$
$A[p+336] = p+336$
$A[p+337] = p+337$
$A[p+338] = p+338$
$A[p+339] = p+339$
$A[p+340] = p+340$
$A[p+341] = p+341$
$A[p+342] = p+342$
$A[p+343] = p+343$
$A[p+344] = p+344$
$A[p+345] = p+345$
$A[p+346] = p+346$
$A[p+347] = p+347$
$A[p+348] = p+348$
$A[p+349] = p+349$
$A[p+350] = p+350$
$A[p+351] = p+351$
$A[p+352] = p+352$
$A[p+353] = p+353$
$A[p+354] = p+354$
$A[p+355] = p+355$
$A[p+356] = p+356$
$A[p+357] = p+357$
$A[p+358] = p+358$
$A[p+359] = p+359$
$A[p+360] = p+360$
$A[p+361] = p+361$
$A[p+362] = p+362$
$A[p+363] = p+363$
$A[p+364] = p+364$
$A[p+365] = p+365$
$A[p+366] = p+366$
$A[p+367] = p+367$
$A[p+368] = p+368$
$A[p+369] = p+369$
$A[p+370] = p+370$
$A[p+371] = p+371$
$A[p+372] = p+372$
$A[p+373] = p+373$
$A[p+374] = p+374$
$A[p+375] = p+375$
$A[p+376] = p+376$
$A[p+377] = p+377$
$A[p+378] = p+378$
$A[p+379] = p+379$
$A[p+380] = p+380$
$A[p+381] = p+381$
$A[p+382] = p+382$
$A[p+383] = p+383$
$A[p+384] = p+384$
$A[p+385] = p+385$
$A[p+386] = p+386$
$A[p+387] = p+387$
$A[p+388] = p+388$
$A[p+389] = p+389$
$A[p+390] = p+390$
$A[p+391] = p+391$
$A[p+392] = p+392$
$A[p+393] = p+393$
$A[p+394] = p+394$
$A[p+395] = p+395$
$A[p+396] = p+396$
$A[p+397] = p+397$
$A[p+398] = p+398$
$A[p+399] = p+399$
$A[p+400] = p+400$
$A[p+401] = p+401$
$A[p+402] = p+402$
$A[p+403] = p+403$
$A[p+404] = p+404$
$A[p+405] = p+405$
$A[p+406] = p+406$
$A[p+407] = p+407$
$A[p+408] = p+408$
$A[p+409] = p+409$
$A[p+410] = p+410$
$A[p+411] = p+411$
$A[p+412] = p+412$
$A[p+413] = p+413$
$A[p+414] = p+414$
$A[p+415] = p+415$
$A[p+416] = p+416$
$A[p+417] = p+417$
$A[p+418] = p+418$
$A[p+419] = p+419$
$A[p+420] = p+420$
$A[p+421] = p+421$
$A[p+422] = p+422$
$A[p+423] = p+423$
$A[p+424] = p+424$
$A[p+425] = p+425$
$A[p+426] = p+426$
$A[p+427] = p+427$
$A[p+428] = p+428$
$A[p+429] = p+429$
$A[p+430] = p+430$
$A[p+431] = p+431$
$A[p+432] = p+432$
$A[p+433] = p+433$
$A[p+434] = p+434$
$A[p+435] = p+435$
$A[p+436] = p+436$
$A[p+437] = p+437$
$A[p+438] = p+438$
$A[p+439] = p+439$
$A[p+440] = p+440$
$A[p+441] = p+441$
$A[p+442] = p+442$
$A[p+443] = p+443$
$A[p+444] = p+444$
$A[p+445] = p+445$
$A[p+446] = p+446$
$A[p+447] = p+447$
$A[p+448] = p+448$
$A[p+449] = p+449$
$A[p+450] = p+450$
$A[p+451] = p+451$
$A[p+452] = p+452$
$A[p+453] = p+453$
$A[p+454] = p+454$
$A[p+455] = p+455$
$A[p+456] = p+456$
$A[p+457] = p+457$
$A[p+458] = p+458$
$A[p+459] = p+459$
$A[p+460] = p+460$
$A[p+461] = p+461$
$A[p+462] = p+462$
$A[p+463] = p+463$
$A[p+464] = p+464$
$A[p+465] = p+465$
$A[p+466] = p+466$
$A[p+467] = p+467$
$A[p+468] = p+468$
$A[p+469] = p+469$
$A[p+470] = p+470$
$A[p+471] = p+471$
$A[p+472] = p+472$
$A[p+473] = p+473$
$A[p+474] = p+474$
$A[p+475] = p+475$
$A[p+476] = p+476$
$A[p+477] = p+477$
$A[p+478] = p+478$
$A[p+479] = p+479$
$A[p+480] = p+480$
$A[p+481] = p+481$
$A[p+482] = p+482$
$A[p+483] = p+483$
$A[p+484] = p+484$
$A[p+485] = p+485$
$A[p+486] = p+486$
$A[p+487] = p+487$
$A[p+488] = p+488$
$A[p+489] = p+489$
$A[p+490] = p+490$
$A[p+491] = p+491$
$A[p+492] = p+492$
$A[p+493] = p+493$
$A[p+494] = p+494$
$A[p+495] = p+495$
$A[p+496] = p+496$
$A[p+497] = p+497$
$A[p+498] = p+498$
$A[p+499] = p+499$
$A[p+500] = p+500$
$A[p+501] = p+501$
$A[p+502] = p+502$
$A[p+503] = p+503$
$A[p+504] = p+504$
$A[p+505] = p+505$
$A[p+506] = p+506$
$A[p+507] = p+507$
$A[p+508] = p+508$
$A[p+509] = p+509$
$A[p+510] = p+510$
$A[p+511] = p+511$
$A[p+512] = p+512$
$A[p+513] = p+513$
$A[p+514] = p+514$
$A[p+515] = p+515$
$A[p+516] = p+516$
$A[p+517] = p+517$
$A[p+518] = p+518$
$A[p+519] = p+519$
$A[p+520] = p+520$
$A[p+521] = p+521$
$A[p+522] = p+522$
$A[p+523] = p+523$
$A[p+524] = p+524$
$A[p+525] = p+525$
$A[p+526] = p+526$
$A[p+527] = p+527$
$A[p+528] = p+528$
$A[p+529] = p+529$
$A[p+530] = p+530$
$A[p+531] = p+531$
$A[p+532] = p+532$
$A[p+533] = p+533$
$A[p+534] = p+534$
$A[p+535] = p+535$
$A[p+536] = p+536$
$A[p+537] = p+537$
$A[p+538] = p+538$
$A[p+539] = p+539$
$A[p+540] = p+540$
$A[p+541] = p+541$
$A[p+542] = p+542$
$A[p+543] = p+543$
$A[p+544] = p+544$
$A[p+545] = p+545$
$A[p+546] = p+546$
$A[p+547] = p+547$
$A[p+548] = p+548$
$A[p+549] = p+549$
$A[p+550] = p+550$
$A[p+551] = p+551$
$A[p+552] = p+552$
$A[p+553] = p+553$
$A[p+554] = p+554$
$A[p+555] = p+555$
$A[p+556] = p+556$
$A[p+557] = p+557$
$A[p+558] = p+558$
$A[p+559] = p+559$
$A[p+560] = p+560$
$A[p+561] = p+561$
$A[p+562] = p+562$
$A[p+563] = p+563$
$A[p+564] = p+564$
$A[p+565] = p+565$
$A[p+566] = p+566$
$A[p+567] = p+567$
$A[p+568] = p+568$
$A[p+569] = p+569$
$A[p+570] = p+570$
$A[p+571] = p+571$
$A[p+572] = p+572$
$A[p+573] = p+573$
$A[p+574] = p+574$
$A[p+575] = p+575$
$A[p+576] = p+576$
$A[p+577] = p+577$
$A[p+578] = p+578$
$A[p+579] = p+579$
$A[p+580] = p+580$
$A[p+581] = p+581$
$A[p+582] = p+582$
$A[p+583] = p+583$
$A[p+584] = p+584$
$A[p+585] = p+585$
$A[p+586] = p+586$
$A[p+587] = p+587$
$A[p+588] = p+588$
$A[p+589] = p+589$
$A[p+590] = p+590$
$A[p+591] = p+591$
$A[p+592] = p+592$
$A[p+593] = p+593$
$A[p+594] = p+594$
$A[p+595] = p+595$
$A[p+596] = p+596$
$A[p+597] = p+597$
$A[p+598] = p+598$
$A[p+599] = p+599$
$A[p+600] = p+600$
$A[p+601] = p+601$
$A[p+602] = p+602$
$A[p+603] = p+603$
$A[p+604] = p+604$
$A[p+605] = p+605$
$A[p+606] = p+606$
$A[p+607] = p+607$
$A[p+608] = p+608$
$A[p+609] = p+609$
$A[p+610] = p+610$
$A[p+611] = p+611$
$A[p+612] = p+612$
$A[p+613] = p+613$
$A[p+614] = p+614$
$A[p+615] = p+615$
$A[p+616] = p+616$
$A[p+617] = p+617$
$A[p+618] = p+618$
$A[p+619] = p+619$
$A[p+620] = p+620$
$A[p+621] = p+621$
$A[p+622] = p+622$
$A[p+623] = p+623$
$A[p+624] = p+624$
$A[p+625] = p+625$
$A[p+626] = p+626$
$A[p+627] = p+627$
$A[p+628] = p+628$
$A[p+629] = p+629$
$A[p+630] = p+630$
$A[p+631] = p+631$
$A[p+632] = p+632$
$A[p+633] = p+633$
$A[p+634] = p+634$
$A[p+635] = p+635$
$A[p+636] = p+636$
$A[p+637] = p+637$
$A[p+638] = p+638$
$A[p+639] = p+639$
$A[p+640] = p+640$
$A[p+641] = p+641$
$A[p+642] = p+642$
$A[p+643] = p+643$
$A[p+644] = p+644$
$A[p+645] = p+645$
$A[p+646] = p+646$
$A[p+647] = p+647$
$A[p+648] = p+648$
$A[p+649] = p+649$
$A[p+650] = p+650$
$A[p+651] = p+651$
$A[p+652] = p+652$
$A[p+653] = p+653$
$A[p+654] = p+654$
$A[p+655] = p+655$
$A[p+656] = p+656$
$A[p+657] = p+657$
$A[p+658] = p+658$
$A[p+659] = p+659$
$A[p+660] = p+660$
$A[p+661] = p+661$
$A[p+662] = p+662$
$A[p+663] = p+663$
$A[p+664] = p+664$
$A[p+665] = p+665$
$A[p+666] = p+666$
$A[p+667] = p+667$
$A[p+668] = p+668$
$A[p+669] = p+669$
$A[p+670] = p+670$
$A[p+671] = p+671$
$A[p+672] = p+672$
$A[p+673] = p+673$
$A[p+674] = p+674$
$A[p+675] = p+675$
$A[p+676] = p+676$
$A[p+677] = p+677$
$A[p+678] = p+678$
$A[p+679] = p+679$
$A[p+680] = p+680$
$A[p+681] = p+681$
$A[p+682] = p+682$
$A[p+683] = p+683$
$A[p+684] = p+684$
$A[p+685] = p+685$
$A[p+686] = p+686$
$A[p+687] = p+687$
$A[p+688] = p+688$
$A[p+689] = p+689$
$A[p+690] = p+690$
$A[p+691] = p+691$
$A[p+692] = p+692$
$A[p+693] = p+693$
$A[p+694] = p+694$
$A[p+695] = p+695$
$A[p+696] = p+696$
$A[p+697] = p+697$
$A[p+698] = p+698$
$A[p+699] = p+699$
$A[p+700] = p+700$
$A[p+701] = p+701$
$A[p+702] = p+702$
$A[p+703] = p+703$
$A[p+704] = p+704$
$A[p+705] = p+705$
$A[p+706] = p+706$
$A[p+707] = p+707$
$A[p+708] = p+708$
$A[p+709] = p+709$
$A[p+710] = p+710$
$A[p+711] = p+711$
$A[p+712] = p+712$
$A[p+713] = p+713$
$A[p+714] = p+714$
$A[p+715] = p+715$
$A[p+716] = p+716$
$A[p+717] = p+717$
$A[p+718] = p+718$
$A[p+719] = p+719$
$A[p+720] = p+720$
$A[p+721] = p+721$
$A[p+722] = p+722$
$A[p+723] = p+723$
$A[p+724] = p+724$
$A[p+725] = p+725$
$A[p+726] = p+726$
$A[p+727] = p+727$
$A[p+728] = p+728$
$A[p+729] = p+729$
$A[p+730] = p+730$
$A[p+731] = p+731$
$A[p+732] = p+732$
$A[p+733] = p+733$
$A[p+734] = p+734$
$A[p+735] = p+735$
$A[p+736] = p+736$
$A[p+737] = p+737$
$A[p+738] = p+738$
$A[p+739] = p+739$
$A[p+740] = p+740$
$A[p+741] = p+741$
$A[p+742] = p+742$
$A[p+743] = p+743$
$A[p+744] = p+744$
$A[p+745] = p+745$
$A[p+746] = p+746$
$A[p+747] = p+747$
$A[p+748] = p+748$
$A[p+749] = p+749$
$A[p+750] = p+750$
$A[p+751] = p+751$
$A[p+752] = p+752$
$A[p+753] = p+753$
$A[p+754] = p+754$
$A[p+755] = p+755$
$A[p+756] = p+756$
$A[p+757] = p+757$
$A[p+758] = p+758$
$A[p+759] = p+759$
$A[p+760] = p+760$
$A[p+761] = p+761$
$A[p+762] = p+762$
$A[p+763] = p+763$
$A[p+764] = p+764$
$A[p+765] = p+765$
$A[p+766] = p+766$
$A[p+767] = p+767$
$A[p+768] = p+768$
$A[p+769] = p+769$
$A[p+770] = p+770$
$A[p+771] = p+771$
$A[p+772] = p+772$
$A[p+773] = p+773$
$A[p+774] = p+774$
$A[p+775] = p+775$
$A[p+776] = p+776$
$A[p+777] = p+777$
$A[p+778] = p+778$
$A[p+779] = p+779$
$A[p+780] = p+780$
$A[p+781] = p+781$
$A[p+782] = p+782$
$A[p+783] = p+783$
$A[p+784] = p+784$
$A[p+785] = p+785$
$A[p+786] = p+786$
$A[p+787] = p+787$
$A[p+788] = p+788$
$A[p+789] = p+789$
$A[p+790] = p+790$
$A[p+791] = p+791$
$A[p+792] = p+792$
$A[p+793] = p+793$
$A[p+794] = p+794$
$A[p+795] = p+795$
$A[p+796] = p+796$
$A[p+797] = p+797$
$A[p+798] = p+798$
$A[p+799] = p+799$
$A[p+800] = p+800$
$A[p+801] = p+801$
$A[p+802] = p+802$
$A[p+803] = p+803$
$A[p+804] = p+804$
$A[p+805] = p+805$
$A[p+806] = p+806$
$A[p+807] = p+807$
$A[p+808] = p+808$
$A[p+809] = p+809$
$A[p+810] = p+810$
$A[p+811] = p+811$
$A[p+812] = p+812$
$A[p+813] = p+813$
$A[p+814] = p+814$
$A[p+815] = p+815$
$A[p+816] = p+816$
$A[p+817] = p+817$
$A[p+818] = p+818$
$A[p+819] = p+819$
$A[p+820] = p+820$
$A[p+821] = p+821$
$A[p+822] = p+822$
$A[p+823] = p+823$
$A[p+824] = p+824$
$A[p+825] = p+825$
$A[p+826] = p+826$
$A[p+827] = p+827$
$A[p+828] = p+828$
$A[p+829] = p+829$
$A[p+830] = p+830$
$A[p+831] = p+831$
$A[p+832] = p+832$
$A[p+833] = p+833$
$A[p+834] = p+834$
$A[p+835] = p+835$
$A[p+836] = p+836$
$A[p+837] = p+837$
$A[p+838] = p+838$
$A[p+839] = p+839$
$A[p+840] = p+840$
$A[p+841] = p+841$
$A[p+842] = p+842$
$A[p+843] = p+843$
$A[p+844] = p+844$
$A[p+845] = p+845$
$A[p+846] = p+846$
$A[p+847] = p+847$
$A[p+848] = p+848$
$A[p+849] = p+849$
$A[p+850] = p+850$
$A[p+851] = p+851$
$A[p+852] = p+852$
$A[p+853] = p+853$
$A[p+854] = p+854$
$A[p+855] = p+855$
$A[p+856] = p+856$
$A[p+857] = p+857$
$A[p+858] = p+858$
$A[p+859] = p+859$
$A[p+860] = p+860$
$A[p+861] = p+861$
$A[p+862] = p+862$
$A[p+863] = p+863$
$A[p+864] = p+864$
$A[p+865] = p+865$
$A[p+866] = p+866$
$A[p+867] = p+867$
$A[p+868] = p+868$
$A[p+869] = p+869$
$A[p+870] = p+870$
$A[p+871] = p+871$
$A[p+872] = p+872$
$A[p+873] = p+873$
$A[p+874] = p+874$
$A[p+875] = p+875$
$A[p+876] = p+876$
$A[p+877] = p+877$
$A[p+878] = p+878$
$A[p+879] = p+879$
$A[p+880] = p+880$
$A[p+881] = p+881$
$A[p+882] = p+882$
$A[p+883] = p+883$
$A[p+884] = p+884$
$A[p+885] = p+885$
$A[p+886] = p+886$
$A[p+887] = p+887$
$A[p+888] = p+888$
$A[p+889] = p+889$
$A[p+890] = p+890$
$A[p+891] = p+891$
$A[p+892] = p+892$
$A[p+893] = p+893$
$A[p+894] = p+894$
$A[p+895] = p+895$
$A[p+896] = p+896$
$A[p+897] = p+897$
$A[p+898] = p+898$
$A[p+899] = p+899$
$A[p+900] = p+900$
$A[p+901] = p+901$
$A[p+902] = p+902$
$A[p+903] = p+903$
$A[p+904] = p+904$
$A[p+905] = p+905$
$A[p+906] = p+906$
$A[p+907] = p+907$
$A[p+908] = p+908$
$A[p+909] = p+909$
$A[p+910] = p+910$
$A[p+911] = p+911$
$A[p+912] = p+912$
$A[p+913] = p+913$
$A[p+914] = p+914$
$A[p+915] = p+915$
$A[p+916] = p+916$
$A[p+917] = p+917$
$A[p+918] = p+918$
$A[p+919] = p+919$
$A[p+920] = p+920$
$A[p+921] = p+921$
$A[p+922] = p+922$
$A[p+923] = p+923$
$A[p+924] = p+924$
$A[p+925] = p+925$
$A[p+926] = p+926$
$A[p+927] = p+927$
$A[p+928] = p+928$
$A[p+929] = p+929$
$A[p+930] = p+930$
$A[p+931] = p+931$
$A[p+932] = p+932$
$A[p+933] = p+933$
$A[p+934] = p+934$
$A[p+935] = p+935$
$A[p+936] = p+936$
$A[p+937] = p+937$
$A[p+938] = p+938$
$A[p+939] = p+939$
$A[p+940] = p+940$
$A[p+941] = p+941$
$A[p+942] = p+942$
$A[p+943] = p+943$
$A[p+944] = p+944$
$A[p+945] = p+945$
$A[p+946] = p+946$
$A[p+947] = p+947$
$A[p+948] = p+948$
$A[p+949] = p+949$
$A[p+950] = p+950$
$A[p+951] = p+951$
$A[p+952] = p+952$
$A[p+953] = p+953$
$A[p+954] = p+954$
$A[p+955] = p+955$
$A[p+956] = p+956$
$A[p+957] = p+957$
$A[p+958] = p+958$
$A[p+959] = p+959$
$A[p+960] = p+960$
$A[p+961] = p+961$
$A[p+962] = p+962$
$A[p+963] = p+963$
$A[p+964] = p+964$
$A[p+965] = p+965$
$A[p+966] = p+966$
$A[p+967] = p+967$
$A[p+968] = p+968$
$A[p+969] = p+969$
$A[p+970] = p+970$
$A[p+971] = p+971$
$A[p+972] = p+972$
$A[p+973] = p+973$
$A[p+974] = p+974$
$A[p+975] = p+975$
$A[p+976] = p+976$
$A[p+977] = p+977$
$A[p+978] = p+978$
$A[p+979] = p+979$
$A[p+980] = p+980$
$A[p+981] = p+981$
$A[p+982] = p+982$
$A[p+983] = p+983$
$A[p+984] = p+984$
$A[p+985] = p+985$
$A[p+986] = p+986$
$A[p+987] = p+987$
$A[p+988] = p+988$
$A[p+989] = p+989$
$A[p+990] = p+990$
$A[p+991] = p+991$
$A[p+992] = p+992$
$A[p+993] = p+993$
$A[p+994] = p+994$
$A[p+995] = p+995$
$A[p+996] = p+996$
$A[p+997] = p+997$
$A[p+998] = p+998$
$A[p+999] = p+999$
$A[p+1000] = p+1000$

$A[p] \ldots A[q]$

$A[p] = p$

$A[p+1] = p+1$

$A[p+2] = p+2$

$A[p+3] = p+3$

$A[p+4] = p+4$

$A[p+5] = p+5$

$A[p+6] = p+6$

$A[p+7] = p+7$

$A[p+8] = p+8$

$A[p+9] = p+9$

$A[p+10] = p+10$

$A[p+11] = p+11$

$A[p+12] = p+12$

$A[p+13] = p+13$

$A[p+14] = p+14$

$A[p+15] = p+15$

$A[p+16] = p+16$

$A[p+17] = p+17$

$A[p+18] = p+18$

$A[p+19] = p+19$

$A[p+20] = p+20$

$A[p+21] = p+21$

$A[p+22] = p+22$

$A[p+23] = p+23$

$A[p+24] = p+24$

$A[p+25] = p+25$

$A[p+26] = p+26$

$A[p+27] = p+27$

$A[p+28] = p+28$

$A[p+29] = p+29$

$A[p+30] = p+30$

$A[p+31] = p+31$

$A[p+32] = p+32$

$A[p+33] = p+33$

$A[p+34] = p+34$

$A[p+35] = p+35$

$A[p+36] = p+36$

$A[p+37] = p+37$

$A[p+38] = p+38$

$A[p+39] = p+39$

$A[p+40] = p+40$

$A[p+41] = p+41$

$A[p+42] = p+42$

$A[p+43] = p+43$

$A[p+44] = p+44$

$A[p+45] = p+45$

$A[p+46] = p+46$

$A[p+47] = p+47$

$A[p+48] = p+48$

$A[p+49] = p+49$

$A[p+50] = p+50$

$A[p+51] = p+51$

$A[p+52] = p+52$

$A[p+53] = p+53$

$A[p+54] = p+54$

$A[p+55] = p+55$

$A[p+56] = p+56$

$A[p+57] = p+57$

$A[p+58] = p+58$

$A[p+59] = p+59$

$A[p+60] = p+60$

$A[p+61] = p+61$

$A[p+62] = p+62$

$A[p+63] = p+63$

$A[p+64] = p+64$

$A[p+65] = p+65$

$A[p+66] = p+66$

$A[p+67] = p+67$

$A[p+68] = p+68$

$A[p+69] = p+69$

$A[p+70] = p+70$

$A[p+71] = p+71$

$A[p+72] = p+72$

$A[p+73] = p+73$

$A[p+74] = p+74$

$A[p+75] = p+75$

$A[p+76] = p+76$

$A[p+77] = p+77$

$A[p+78] = p+78$

$A[p+79] = p+79$

$A[p+80] = p+80$

$A[p+81] = p+81$

$A[p+82] = p+82$

$A[p+83] = p+83$

$A[p+84] = p+84$

$A[p+85] = p+85$

$A[p+86] = p+86$

$A[p+87] = p+87$

$A[p+88] = p+88$

$A[p+89] = p+89$

$A[p+90] = p+90$

$A[p+91] = p+91$

$A[p+92] = p+92$

$A[p+93] = p+93$

$A[p+94] = p+94$

$A[p+95] = p+95$

$A[p+96] = p+96$

$A[p+97] = p+97$

$A[p+98] = p+98$

$A[p+99] = p+99$

$A[p+100] = p+100$

$A[p+101] = p+101$

$A[p+102] = p+102$

$A[p+103] = p+103$

$A[p+104] = p+104$

$A[p+105] = p+105$

$A[p+106] = p+106$

$A[p+107] = p+107$

$A[p+108] = p+108$

$A[p+109] = p+109$

$A[p+110] = p+110$

$A[p+111] = p+111$

$A[p+112] = p+112$

$A[p+113] = p+113$

$A[p+114] = p+114$

$A[p+115] = p+115$

$A[p+116] = p+116$

$A[p+117] = p+117$

$A[p+118] = p+118$

$A[p+119] = p+119$

$A[p+120] = p+120$

$A[p+121] = p+121$

$A[p+122] = p+122$

$A[p+123] = p+123$

$A[p+124] = p+124$

$A[p+125] = p+125$

$A[p+126] = p+126$

$A[p+127] = p+127$

$A[p+128] = p+128$

$A[p+129] = p+129$

$A[p+130] = p+130$

$A[p+131] = p+131$

$A[p+132] = p+132$

$A[p+133] = p+133$

$A[p+134] = p+134$

$A[p+135] = p+135$

$A[p+136] = p+136$

$A[p+137] = p+137$

$A[p+138] = p+138$

$A[p+139] = p+139$

$A[p+140] = p+140$

$A[p+141] = p+141$

$A[p+142] = p+142$

$A[p+143] = p+143$

$A[p+144] = p+144$

$A[p+145] = p+145$

$A[p+146] = p+146$

$A[p+147] = p+147$

$A[p+148] = p+148$

$A[p+149] = p+149$

$A[p+150] = p+150$

$A[p+151] = p+151$

$A[p+152] = p+152$

$A[p+153] = p+153$

$A[p+154] = p+154$

$A[p+155] = p+155$

$A[p+156] = p+156$

$A[p+157] = p+157$

$A[p+158] = p+158$

$A[p+159] = p+159$

$A[p+160] = p+160$

$A[p+161] = p+161$

$A[p+162] = p+162$

$A[p+163] = p+163$

$A[p+164] = p+164$

$A[p+165] = p+165$

$A[p+166] = p+166$

$A[p+167] = p+167$

$A[p+168] = p+168$

$A[p+169] = p+169$

$A[p+170] = p+170$

$A[p+171] = p+171$

$A[p+172] = p+172$

$A[p+173] = p+173$

$A[p+174] = p+174$

$A[p+175] = p+175$

$A[p+176]

ESERCIZI 3 P.1

- Per chiarezza si riporta di fianco la soluzione di ciascun esercizio, ma non è richiesto che le soluzioni siano scritte in modo così ordinato. Si può usare anche tutta la pagina, se si vuole.
- Si può usare la parte sotto la linea per appunti. Tenete presente che: (a) quello che è sotto la linea non sarà letto, (b) quanto di quanto è sopra la linea è ritenuto rilevante dipende dal vostro stile di scrittura, (c) quanto di quanto è sopra la linea è ritenuto rilevante dipende anche dalla lunghezza del testo, (d) quanto di quanto è sopra la linea è ritenuto rilevante dipende anche dalla chiarezza.
- Si può usare la matita e/o la gomma per cancellare. Tenete presente che: (a) quanto di quanto è sotto la linea è ritenuto rilevante dipende anche dalla pulizia, (b) quanto di quanto è sotto la linea è ritenuto rilevante dipende anche dalla chiarezza.

1. *Esercizio:* Dati interi $\boxed{3}$ e $\boxed{10}$ $\boxed{2}$ $\boxed{4}$ $\boxed{1}$ $\boxed{5}$ $\boxed{6}$ $\boxed{7}$ $\boxed{8}$ $\boxed{9}$ numeri di Stirling del secondo tipo nessi desumi con il metodo $\boxed{\ }$ dalla tabella seguente:

$$\left\{ {3 \atop 1} \right\} = \left\{ {3 \atop 2} \right\} = \left\{ {3 \atop 3} \right\} = \left\{ {10 \atop 1} \right\} = \left\{ {10 \atop 2} \right\} = \left\{ {10 \atop 3} \right\} = \left\{ {10 \atop 4} \right\} = \left\{ {10 \atop 5} \right\} = \left\{ {10 \atop 6} \right\} = \left\{ {10 \atop 7} \right\} = \left\{ {10 \atop 8} \right\} = \left\{ {10 \atop 9} \right\} = \left\{ {10 \atop 10} \right\} = 1,$$

e dalle condizioni iniziali

$$\left\{ {n \atop 1} \right\} = 1, \quad \left\{ {n \atop n-1} \right\} = \binom{n}{2}, \quad \text{per } n \geq 1, \quad \left\{ {n \atop 2} \right\} = 2^{n-1} - 1, \quad \forall n.$$

Si scriva in pseudocodice di Programmazione Dinamica che, partendo da input interi n, m, con $3 \leq n \leq 7$, m restituisca in output il corrispondente numero di Stirling $\boxed{\ }$. Si analizzi la complessità dell'algoritmo.

```
f(n,m):
  if n=m or m=1:
    return 1
  else if n=1 or m=0:
    return 0
  else if n=2 and m=2:
    return 1
  else if n=3 and m=2:
    return 3
  else if n=3 and m=3:
    return 1
  else if n=10 and m=1:
    return 1
  else if n=10 and m=2:
    return 511
  else if n=10 and m=3:
    return 9330
  else if n=10 and m=4:
    return 42525
  else if n=10 and m=5:
    return 55980
  else if n=10 and m=6:
    return 22827
  else if n=10 and m=7:
    return 3025
  else if n=10 and m=8:
    return 145
  else if n=10 and m=9:
    return 1
  else return 0
```

La tabella T[i,j] per n = 0 vale: 0 0 0 0 0 0 0 0

Si supponga di non conoscere la tabella qualche riga e qualche colonna sopra: ad esempio si ha $\left\{ {4 \atop 2} \right\} = 7$.

```
f(n,m):
  if n=m or m=1:
    return 1
  else if n=1 or m=0:
    return 0
  else if n=2 and m=2:
    return 1
  else if n=3 and m=2:
    return 3
  else if n=3 and m=3:
    return 1
  else if n=10 and m=1:
    return 1
  else if n=10 and m=2:
    return 511
  else if n=10 and m=3:
    return 9330
  else if n=10 and m=4:
    return 42525
  else if n=10 and m=5:
    return 55980
  else if n=10 and m=6:
    return 22827
  else if n=10 and m=7:
    return 3025
  else if n=10 and m=8:
    return 145
  else if n=10 and m=9:
    return 1
  else return 0
```

La tabella T[i,j] per n = 1 vale: 0 1 1 1 1 1 1 1 1 1

Si supponga di non conoscere la tabella qualche riga e qualche colonna sopra: ad esempio si ha $\left\{ {5 \atop 2} \right\} = 15$.

```
f(n,m):
  if n=m or m=1:
    return 1
  else if n=1 or m=0:
    return 0
  else if n=2 and m=2:
    return 1
  else if n=3 and m=2:
    return 3
  else if n=3 and m=3:
    return 1
  else if n=10 and m=1:
    return 1
  else if n=10 and m=2:
    return 511
  else if n=10 and m=3:
    return 9330
  else if n=10 and m=4:
    return 42525
  else if n=10 and m=5:
    return 55980
  else if n=10 and m=6:
    return 22827
  else if n=10 and m=7:
    return 3025
  else if n=10 and m=8:
    return 145
  else if n=10 and m=9:
    return 1
  else return 0
```

La tabella T[i,j] per n = 2 vale: 0 1 3 7 15 31 63 127 255 511

Si supponga di non conoscere la tabella qualche riga e qualche colonna sopra: ad esempio si ha $\left\{ {6 \atop 2} \right\} = 31$.

```
f(n,m):
  if n=m or m=1:
    return 1
  else if n=1 or m=0:
    return 0
  else if n=2 and m=2:
    return 1
  else if n=3 and m=2:
    return 3
  else if n=3 and m=3:
    return 1
  else if n=10 and m=1:
    return 1
  else if n=10 and m=2:
    return 511
  else if n=10 and m=3:
    return 9330
  else if n=10 and m=4:
    return 42525
  else if n=10 and m=5:
    return 55980
  else if n=10 and m=6:
    return 22827
  else if n=10 and m=7:
    return 3025
  else if n=10 and m=8:
    return 145
  else if n=10 and m=9:
    return 1
  else return 0
```

La tabella T[i,j] per n = 3 vale: 0 1 4 13 40 121 364 1093 3280 9841

Si supponga di non conoscere la tabella qualche riga e qualche colonna sopra: ad esempio si ha $\left\{ {7 \atop 2} \right\} = 63$.

```
f(n,m):
  if n=m or m=1:
    return 1
  else if n=1 or m=0:
    return 0
  else if n=2 and m=2:
    return 1
  else if n=3 and m=2:
    return 3
  else if n=3 and m=3:
    return 1
  else if n=10 and m=1:
    return 1
  else if n=10 and m=2:
    return 511
  else if n=10 and m=3:
    return 9330
  else if n=10 and m=4:
    return 42525
  else if n=10 and m=5:
    return 55980
  else if n=10 and m=6:
    return 22827
  else if n=10 and m=7:
    return 3025
  else if n=10 and m=8:
    return 145
  else if n=10 and m=9:
    return 1
  else return 0
```

La tabella T[i,j] per n = 4 vale: 0 1 5 16 51 171 624 2380 9495 40�آ

Si supponga di non conoscere la tabella qualche riga e qualche colonna sopra: ad esempio si ha $\left\{ {8 \atop 2} \right\} = 127$.

```
f(n,m):
  if n=m or m=1:
    return 1
  else if n=1 or m=0:
    return 0
  else if n=2 and m=2:
    return 1
  else if n=3 and m=2:
    return 3
  else if n=3 and m=3:
    return 1
  else if n=10 and m=1:
    return 1
  else if n=10 and m=2:
    return 511
  else if n=10 and m=3:
    return 9330
  else if n=10 and m=4:
    return 42525
  else if n=10 and m=5:
    return 55980
  else if n=10 and m=6:
    return 22827
  else if n=10 and m=7:
    return 3025
  else if n=10 and m=8:
    return 145
  else if n=10 and m=9:
    return 1
  else return 0
```

La tabella T[i,j] per n = 5 vale: 0 1 6 22 85 341 1555 7770 42525 246�آ

Si supponga di non conoscere la tabella qualche riga e qualche colonna sopra: ad esempio si ha $\left\{ {9 \atop 2} \right\} = 255$.

```
f(n,m):
  if n=m or m=1:
    return 1
  else if n=1 or m=0:
    return 0
  else if n=2 and m=2:
    return 1
  else if n=3 and m=2:
    return 3
  else if n=3 and m=3:
    return 1
  else if n=10 and m=1:
    return 1
  else if n=10 and m=2:
    return 511
  else if n=10 and m=3:
    return 9330
  else if n=10 and m=4:
    return 42525
  else if n=10 and m=5:
    return 55980
  else if n=10 and m=6:
    return 22827
  else if n=10 and m=7:
    return 3025
  else if n=10 and m=8:
    return 145
  else if n=10 and m=9:
    return 1
  else return 0
```

La tabella T[i,j] per n = 6 vale: 0 1 7 29 141 771 4�آ 28�آ 179�آ 1197

Si supponga di non conoscere la tabella qualche riga e qualche colonna sopra: ad esempio si ha $\left\{ {10 \atop 2} \right\} = 511$.

```
f(n,m):
  if n=m or m=1:
    return 1
  else if n=1 or m=0:
    return 0
  else if n=2 and m=2:
    return 1
  else if n=3 and m=2:
    return 3
  else if n=3 and m=3:
    return 1
  else if n=10 and m=1:
    return 1
  else if n=10 and m=2:
    return 511
  else if n=10 and m=3:
    return 9330
  else if n=10 and m=4:
    return 42525
  else if n=10 and m=5:
    return 55980
  else if n=10 and m=6:
    return 22827
  else if n=10 and m=7:
    return 3025
  else if n=10 and m=8:
    return 145
  else if n=10 and m=9:
    return 1
  else return 0
```

La tabella T[i,j] per n = 7 vale: 0 1 8 36 204 1341 10آ 86آ 7آ 6آ

2. *Esercizio:* Si supponga di dover detenere un caso di sole lunga metà n. Esso è poi composto da una lunga metà $n/2$ e da una lunga metà $n/2$. Si supponga che la lunga metà $n/2$ sia pari. Per questo a sua volta [illegible] Programmazione Dinamica che, partendo da input interi n [illegible] n, restituisca in output il corrispondente numero di Catalano C_n. Si analizzi la complessità dell'algoritmo.

```
f(n):
  if n=0 or n=1:
    return 1
  else:
    s=0
    for i=0 to n-1 do:
      s=s+f(i)*f(n-1-i)
    return s
```

Posto, a scopo di esempio, che n = 4, si ha:

f(0)=1 f(1)=1 f(2)=2 f(3)=5

Si ha: f(4)=14

Si ha: f(5)=42

Si ha: f(6)=132

Si ha: f(7)=429

Si ha: f(8)=1430

Si ha: f(9)=4آ آ

Si ha: f(10)=16آ آ

Si ha: f(11)=58آ آ

Si ha: f(12)=208آ آ

Si ha: f(13)=75آ آ

Si ha: f(14)=27آ آ

Si ha: f(15)=98آ آ

Si ha: f(16)=35آ آ

Si ha: f(17)=12آ آ

Si ha: f(18)=46آ آ

Si ha: f(19)=16آ آ

Si ha: f(20)=6آ آ

Si ha: f(21)=2آ آ

Si ha: f(22)=8آ آ

Si ha: f(23)=3آ آ

Si ha: f(24)=1آ آ

Si ha: f(25)=4آ آ

Si ha: f(26)=1آ آ

Si ha: f(27)=6آ آ

Si ha: f(28)=2آ آ

Si ha: f(29)=8آ آ

Si ha: f(30)=3آ آ

Si ha: f(31)=1آ آ

Si ha: f(32)=4آ آ

Si ha: f(33)=1آ آ

Si ha: f(34)=6آ آ

Si ha: f(35)=2آ آ

Si ha: f(36)=8آ آ

Si ha: f(37)=3آ آ

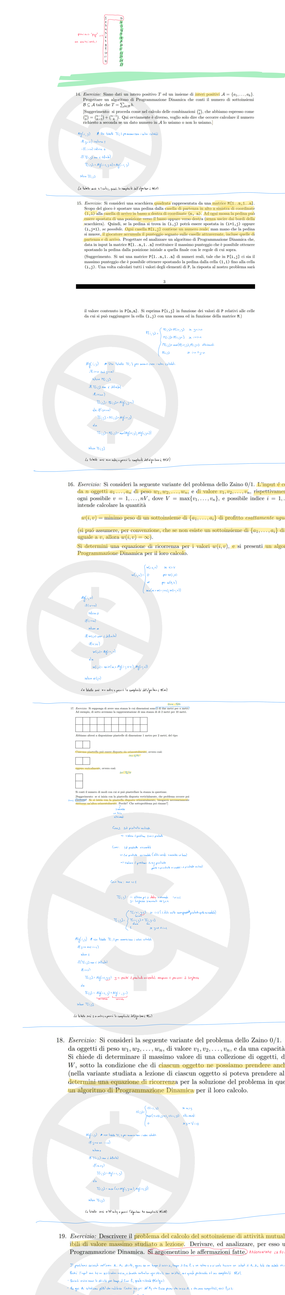
Si ha: f(38)=1آ آ

Si ha: f(39)=4آ آ

Si ha: f(40)=1آ آ

Si ha:

ESERCIZI 3 P.2



14. *Esercizio:* Siano dati un intero positivo T ed un insieme di interi positivi $A = \{a_1, \ldots, a_n\}$. Progettare un algoritmo di Programmazione Dinamica che conti il numero di sottoinsiemi di A la cui somma è T.

(Suggerimento: si procede come nel calcolo delle combinazioni $\binom{n}{k}$; che abbiamo espresso come $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$. Qui ovviamente è diverso, voglio solo dire che occorre calcolare il numero (ricorsivo) a seconda se un dato numero a_i fa untarto o non fa untarto).

$$\begin{array}{ll} \mathrm{Pg}[i,j] & \text{Il fatto che } \{i,\ldots,n\} \text{ permetta } \text{untarto } j \\ \mathrm{Pg}[i,j] = \mathrm{False} & \\ \text{if } i > n \text{ and } j > 0 & \\ \quad \mathrm{Pg}[i,j] = \mathrm{False} & \\ \text{if } i > n \text{ and } j = 0 & \\ \quad \mathrm{Pg}[i,j] = \mathrm{True} & \\ \text{else: } \mathrm{Pg}[i,j] & \end{array}$$

(A fatto per $\{i,\ldots,n\}$, qual è il valore di j?)

15. *Esercizio:* Si consideri una scacchiera quadrata rappresentata da una matrice $M[1..n,1..n]$. Scopo del gioco è spostare una pedina dalla cella di partenza in alto a sinistra di coordinate $(1,1)$ alla cella di arrivo in basso a destra di coordinate (n,n). Ad ogni mossa la pedina può spostarsi di una cella verso destra o verso il basso, ma non può entrare in celle contenenti ostacoli. Quindi, se la pedina si trova in (i,j) può spostarsi verso $(i,j+1)$ o verso $(i+1,j)$, se possibile. Ogni cella $M[i,j]$ contiene un numero naturale: una mossa che la pedina si muove, il giocatore accumula il punteggio segnato sulla cella attraversata (anche quella di partenza e di arrivo). Progettare ed analizzare un algoritmo di Programmazione Dinamica che, data la matrice M, calcoli il massimo punteggio che il giocatore può accumulare spostando la pedina dalla posizione iniziale a quella finale con le regole di cui sopra.

(Suggerimento. Si sa una matrice $P[1..n,1..n]$ di numeri reali, tale che $\forall P[i,j]$ (o che il massimo punteggio che è possibile ottenere spostando la pedina dalla cella $(1,1)$ fino alla cella (i,j). Una volta calcolati tutti i valori degli elementi di P, la risposta al nostro problema sarà

$$P[n,n]$$

il valore contenuto in $P[n,n]$. Si supposa $P[1,1]$ la funzione dei valori di P relativi alle celle da cui si può raggiungere la cella $(1,1)$ (con una mossa ed la funzione della matrice M.)

$$\begin{array}{ll} P[1,1] & \begin{cases} M[1,1] & \text{se } i = j = 1 \\ +\infty & \text{se } i = j = 1 \\ +\infty & \text{se } M[i,j] \text{ è ostacolo} \\ M[i,j] & \text{se } i = 1 \text{ o } j = 1 \end{cases} \end{array}$$

$$\begin{array}{ll} \mathrm{Pg}[i,j] & \text{Il fatto che } \{i,\ldots,n\} \text{ permetta } \text{untarto } j \\ \mathrm{Pg}[i,j] = \mathrm{False} & \\ \text{if } i > n \text{ and } j > 0 & \\ \quad \mathrm{Pg}[i,j] = \mathrm{False} & \\ \text{if } i > n \text{ and } j = 0 & \\ \quad \mathrm{Pg}[i,j] = \mathrm{True} & \\ \text{if } M[i,j] = +\infty & \\ \quad \mathrm{Pg}[i,j] = \mathrm{False} & \\ \text{if } i = 1 & \\ \quad \mathrm{Pg}[i,j] = \mathrm{Pg}[i,j-1] & \\ \text{if } j = 1 & \\ \quad \mathrm{Pg}[i,j] = \mathrm{Pg}[i-1,j] & \\ \text{else: } \mathrm{Pg}[i,j] = \mathrm{Pg}[i-1,j] \vee \mathrm{Pg}[i,j-1] & \end{array}$$

(A fatto per $\{i,\ldots,n\}$, qual è il valore di j?)

16. *Esercizio:* Si consideri la seguente variante del problema dello Zaino 0/1. L'input è costituito da n oggetti $o_1, \ldots, o_n$ di pesi $w_1, w_2, \ldots, w_n$ e di valore $v_1, v_2, \ldots, v_n$, rispettivamente. Per ogni possibile $v = 1, \ldots, nV$, dove $V = \max\{v_1, \ldots, v_n\}$, e possibile indice $i = 1, \ldots, n$, si intende calcolare la quantità

$$w[i, v] = \text{minimo peso di un sottoinsieme di } \{o_1, \ldots, o_i\} \text{ di profitto esattamente uguale a } v$$

(si può assumere, per convenzione, che se non esiste un sottoinsieme di $\{o_1, \ldots, o_i\}$ di profitto uguale a v, allora $w[i, v] = \infty$).

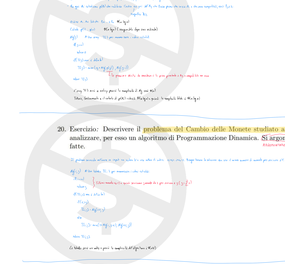
Si determini una equazione di ricorrenza per i valori $w[i, v]$, e si presenti un algoritmo di Programmazione Dinamica per il loro calcolo.

$$\begin{array}{ll} w[i,v] & \begin{cases} \infty & \text{se } v > n \\ w[i-1,v] & \text{se } v \leq n \text{ e } v < v_i \\ w & \text{se } v \leq n \text{ e } v \geq v_i \\ \min\{w, w[i-1, v-v_i]\} & \end{cases} \end{array}$$

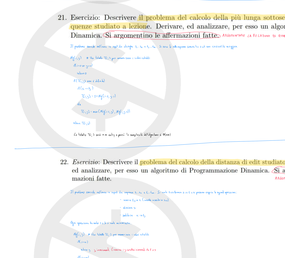
$$\begin{array}{ll} \mathrm{Pg}[i,v] & \\ \mathrm{Pg}[i,v] = \mathrm{False} & \\ \text{if } v = 0 & \\ \quad \mathrm{Pg}[i,v] = \mathrm{True} & \\ \text{if } i = 1 & \\ \quad \mathrm{Pg}[i,v] = \mathrm{Pg}[1,v] \vee \mathrm{Pg}[1,v-v_1] & \\ \text{if } v \geq v_1 & \\ \quad \mathrm{Pg}[i,v] = \mathrm{Pg}[i-1,v] \vee \mathrm{Pg}[i-1,v-v_i] & \\ \text{else: } \mathrm{Pg}[i,v] = \mathrm{Pg}[i-1,v] & \end{array}$$

(A fatto per $\{i,\ldots,n\}$, qual è il valore di v?)

17. *Esercizio:* Si suppongo di avere una stanza e un elencazione non di 0 di 1 ed 1 ADULTO di stanza, di cui ciascuna è rappresentata da una stanza di 0 e 1 ADULTO di stanza.

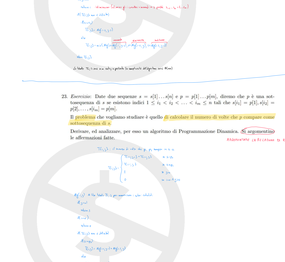


Stanza vuota = Stanza piena di 0 elementi (una per l'unità di tempo)



Stanza piena = Stanza dopo la ricorsione

$$\begin{array}{l} \text{Stanza} = \text{Stanza vuota, stanza piena} \\ \text{Stanza piena} = \text{stanza, stanza} \end{array}$$



Stanza = 1 elemento di stanza vuota o di una piena(ma la stanza in questione.

Domanda: se la stanza non ha spazio disponibile, significa che, prendere nessun pezzo [illegible] la ricorsione non ha spazio disponibile? Significa? [illegible]

Domanda: quale è la condizione di arresto? Che interpretazione per stanza?

$$\begin{array}{ll} \mathrm{Pg}[i,j] & \\ \mathrm{Pg}[i,j] = \mathrm{False} & \\ \text{if } i > n \text{ and } j > 0 & \\ \quad \mathrm{Pg}[i,j] = \mathrm{False} & \\ \text{if } i > n \text{ and } j = 0 & \\ \quad \mathrm{Pg}[i,j] = \mathrm{True} & \\ \text{if } M[i,j] = +\infty & \\ \quad \mathrm{Pg}[i,j] = \mathrm{False} & \\ \text{if } i = 1 & \\ \quad \mathrm{Pg}[i,j] = \mathrm{Pg}[1,j] \vee \mathrm{Pg}[1,j-v_1] & \\ \text{if } j \geq v_1 & \\ \quad \mathrm{Pg}[i,j] = \mathrm{Pg}[i-1,j] \vee \mathrm{Pg}[i-1,j-v_i] & \\ \text{else: } \mathrm{Pg}[i,j] = \mathrm{Pg}[i-1,j] & \end{array}$$

(A fatto per $\{i,\ldots,n\}$, qual è il valore di j?)

18. *Esercizio:* Si consideri la seguente variante del problema dello Zaino 0/1. L'input è costituito da oggetti di peso $w_1, w_2, \ldots, w_n$, di valore $v_1, v_2, \ldots, v_n$, e da una capacità di carico totale W. Si chiede di determinare il massimo valore di una collezione di oggetti, di peso totale al più W, sotto la condizione che di ciascun oggetto ne possiamo prendere anche più di una copia (nella variante studiata a lezione di ciascun oggetto si poteva prendere al più una copia). Si determini una equazione di ricorrenza per la soluzione del problema in questione, e si presenti un algoritmo di Programmazione Dinamica per il loro calcolo.

$$\begin{array}{ll} v[i,w] & \begin{cases} 0 & \text{se } i = n \\ \max\{v_i + v[i, w - w_i], v[i+1, w]\} & \text{se } w \geq w_i \\ v & \text{se } w < w_i \end{cases} \end{array}$$

$$\begin{array}{ll} \mathrm{Pg}[i,j] & \text{Il fatto che } \{i,\ldots,n\} \text{ permetta } \text{zaino } j \text{ ed } \\ \mathrm{Pg}[i,j] = \mathrm{False} & \\ \text{if } i > n & \\ \quad \mathrm{Pg}[i,j] = \mathrm{False} & \\ \text{if } M[i,j] = +\infty & \\ \quad \mathrm{Pg}[i,j] = \mathrm{False} & \\ \text{if } i = 1 & \\ \quad \mathrm{Pg}[i,j] = \mathrm{Pg}[1,j] \vee \mathrm{Pg}[1,j-v_1] & \\ \text{if } j \geq v_1 & \\ \quad \mathrm{Pg}[i,j] = \mathrm{Pg}[i-1,j] \vee \mathrm{Pg}[i-1,j-v_i] & \\ \text{else: } \mathrm{Pg}[i,j] = \mathrm{Pg}[i-1,j] & \end{array}$$

(A fatto per $\{i,\ldots,n\}$, qual'è il valore di j ed W?)

19. *Esercizio:* Descrivere il problema del calcolo del sottoinsieme di attività mutualmente compatibili di valore massimo studiato a lezione. Derivare, ed analizzare, per esso un algoritmo di Programmazione Dinamica. Si argomentino le affermazioni fatte. Algoritmo di Pirlisino lo Sveglio

Il problema quindi [illegible] di n [illegible] di n [illegible] [illegible] $[s_i, f_i]$ [illegible] $[s_i, f_i]$ [illegible] $[s_i, f_i]$

ESERCIZI 4

Esercizi (Algoritmi #17)

1. *Esercizio*: Si applichi l'algoritmo **Greedy Activity Selector** sul seguente insieme di attività. $A = \{A_1, A_2, \ldots, A_{12}\}$, con $A_1 \equiv [0, 12]$, $A_2 \equiv [3, 14]$, $A_3 \equiv [9, 10]$, $A_4 \equiv [9, 13]$, $A_5 \equiv [13, 16]$, $A_6 \equiv [7, 11]$, $A_7 \equiv [2, 5]$, $A_8 \equiv [8, 9]$, $A_9 \equiv [4, 6]$, $A_{10} \equiv [1, 7]$, illustrandone il comportamento ad ogni passo. $A_{11} \equiv$ [illegible], $A_{12} \equiv$ [illegible]

2. Dubito le abbia, ma bisogna il fatto che siano ordinate mi rende difficile. [illegible]
3. Si sceglie $S = \{A_1, A_3, A_5, A_7, A_8, A_9, A_{10}, A_{12}\}$.
4. Dimostrazione: Si l'unione ottenuta è la maggior di essere.
5. $S = \{A_1\}$. Inizio: $A_1 \equiv 1$ è quindi il più piccolo finito.
6. Successivo: Si sceglie la prima attività che è compatibile con A_1 ossia l'attività A_3 (la A_2 finisce dopo).
7. Scelta: A_3. Inizio: $f_3 = 5$ perché è compatibile con A_1. $S = \{A_1, A_3\}$
8. Ripeto: successivo è A_5 perché finisce la prima che è compatibile con A_3.
9. Scelta: A_5, A_7. Successivo è A_7 perché è compatibile con A_5. $S = \{A_1, A_3, A_7\}$
10. Ripeto: è rimasto A_8 perché finisce la prima che è compatibile con A_7.
11. Scelta: A_8, A_9. Successivo è A_9 perché è compatibile con A_8. $S = \{A_1, A_3, A_7, A_8\}$
12. Ripeto: è rimasto A_{10} perché è compatibile con A_9. $S = \{A_1, A_3, A_7, A_8, A_9\}$
13. Scelta: A_{10}. Successivo è A_{12}.
14. Ripeto: $S = \{A_1, A_3, A_7, A_8, A_9, A_{10}, A_{12}\}$

3. *Esercizio*: Si supponga di dover effettuare un viaggio dalla località A alla località B con un'auto che ha un'autonomia di k chilometri. Lungo il percorso, a partire da A sono presenti n distributori di benzina, ciascuno distante dal precedente meno di k chilometri e l'ultimo dista meno di k chilometri da B. Sia d_i la distanza che separa il distributore i dal distributore $i+1$, per $i = 1, 2, \ldots, n-1$, sia d_n la distanza tra A ed il primo distributore e sia d_n la distanza da B dall'ultimo distributore. Descrivere un algoritmo greedy che selezioni un numero minimo di distributori in cui far tappa durante il viaggio. Giustificare le affermazioni fatte.

```
Attuale = 0
Fermate = {}
Ciclo for (i = 1 ... n)
    if (distanza(Attuale, i) > k)
        Fermate = Fermate + {i}
        Attuale = i
    else
        continua(distanza(Attuale, i))
return Fermate
```

← indichiamo il numero di passi di partenza e di arrivo

← il mio numero di regolo (distanza) è quello di Attuale gioco

C'è qualche errore di indice, e poi manca la fermata di destinazione.
Se a la regola è il distributore i_k ne sarà un passo prima di i_k (distanza d_k). Complessità $\Theta(n)$

4. *Esercizio*: Sia dato un intero N, ed un vettore di interi $a = a[1], \ldots, a[n]$. Il problema è di calcolare il massimo valore di k per cui possiamo trovare indici $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ per cui

$$a[i_1] + a[i_2] + \ldots + a[i_k] \leq N$$

Dare un algoritmo greedy che risolve il problema, provandone la correttezza. Come al solito, il primo passo consisterà nell'ordinare il vettore a in modo opportuno.

Ordinare a in un certo modo. Ciclo = n.
i1
sum(i1)
Ciclo(n)
  N.N.a
  i++
  k++
  sum
  sum

return i-1

A gran grosso l'algoritmo individua a_i a $i = 1, 2, \ldots, n$ in n, ordina in N valore.
N' prima scrivere bisogna valore N gradiamo per N_i con $N - a_i$.

Ci vuole ordinare N + chiamata.
N valore sia ottenuto con la chiamata gran. Il valore ordinato. $\sum_{i=1}^{n} k$ N
Allora Θ (chiamata) aggiunto sono che gran N, a_i, sono ordinati.
non in quale non l'algoritmo sarebbe complessità $\Theta(N)$

5. *Esercizio*: Si descriva in dettaglio l'algoritmo greedy per il problema della Selezione di Attività, e si provi che esso produce una soluzione ottima.

Input: Attività $A_1, A_2, \ldots, A_n$ tali che A_i ha inizio s_i e fine f_i, con $s_i < f_i$.
Output: Un sottoinsieme $S \subseteq A$ contenente il massimo numero di due attività mutuamente non compatibili, cioè $(s_i, f_i) \cap (s_j, f_j) = \emptyset$.

Ordina A secondo i valori di f_i (f[i]).
$S = \{A_1\}$
$j = 1$
Ciclo(i = 2 ... n)
  if $(s_i \geq f_j)$
    $S = S \cup \{A_i\}$
    $j = i$
  fine
fine

Ci compatibile $\Theta(n)$ l'algoritmo è parziale differenziando per ciascun $\Theta(\log n)$
Il Θ ciascun $\Theta(n)$ compatibile $\Theta(n)$ l'algoritmo $\Theta(n \log n)$

Prova che l'algoritmo produce la soluzione ottima

Assumiamo per contraddizione, quindi supponiamo che esiste un algoritmo OPT che produce una soluzione di cardinalità maggiore.
$A = \{A_1, \ldots, A_k\}$ soluzione di GREDY.
$E = \{E_1, \ldots, E_m\}$ soluzione di OPT.

Sia r il primo indice tale che $A_r \neq E_r$. Sia A_r l'attività che GREDY sceglie per la r-esima volta.
Ma OPT deve scegliere un'attività E_r, quindi OPT sceglie E_r prima.
Ma se A_r finisce prima di E_r, allora A_r è compatibile con $A_1, \ldots, A_{r-1}$ e quindi GREDY sceglierà A_r invece di E_r, contraddizione.
Se A_r finisce dopo E_r, allora A_r è in conflitto con E_r e OPT non può scegliere A_r dopo E_r.
Quindi OPT non può scegliere A_r e quindi A_r è in A_{r+1}.
Quindi sempre che GREDY sceglie un'attività in più di OPT, stiamo ancora lasciando più spazio per scegliere attività successive.
Se ne deduce che la soluzione di OPT deve essere uguale alla soluzione di GREDY.

6. *Esercizio*: Si descriva in dettaglio l'algoritmo greedy per il problema della Selezione di Attività, e si provi che esso produce una soluzione ottima.

Input: Attività $A_1, A_2, \ldots, A_n$ tali che A_i ha inizio s_i e fine f_i, con $s_i < f_i$.
Output: Un sottoinsieme $S \subseteq A$ contenente il massimo numero di due attività mutuamente non compatibili, cioè $(s_i, f_i) \cap (s_j, f_j) = \emptyset$.

Ordina A secondo i valori di f_i (f[i]).
$S = \{A_1\}$
$j = 1$
Ciclo(i = 2 ... n)
  if $(s_i \geq f_j)$
    $S = S \cup \{A_i\}$
    $j = i$
  fine
fine

Ci compatibile $\Theta(n)$ l'algoritmo è parziale differenziando per ciascun $\Theta(\log n)$
Il Θ ciascun $\Theta(n)$ compatibile $\Theta(n)$ l'algoritmo $\Theta(n \log n)$

Prova che l'algoritmo produce la soluzione ottima

Assumiamo per contraddizione, quindi supponiamo che esiste un algoritmo OPT che produce una soluzione di cardinalità maggiore.
$A = \{A_1, \ldots, A_k\}$ soluzione di GREDY.
$E = \{E_1, \ldots, E_m\}$ soluzione di OPT.

Sia r il primo indice tale che $A_r \neq E_r$. Sia A_r l'attività che GREDY sceglie per la r-esima volta.
Ma OPT deve scegliere un'attività E_r, quindi OPT sceglie E_r prima.
Ma se A_r finisce prima di E_r, allora A_r è compatibile con $A_1, \ldots, A_{r-1}$ e quindi GREDY sceglierà A_r invece di E_r, contraddizione.
Se A_r finisce dopo E_r, allora A_r è in conflitto con E_r e OPT non può scegliere A_r dopo E_r.
Quindi sempre che GREDY sceglie un'attività in più di OPT, stiamo ancora lasciando più spazio per scegliere attività successive.
Se ne deduce che la soluzione di OPT deve essere uguale alla soluzione di GREDY.

8. *Esercizio*: Si costruisca un codifica di Huffman per il seguente insieme di caratteri e relative frequenze
a:7 b:7 c:14 d:41 e:51 f:65 1:68 1:98, r:16
Usando la codifica costruita, si codifichi il testo **bccfrabla**
Sappiamo già grazie ai e studia, di avere quattro 1 e 4 caratteri che a 1 hanno, ma non ci sono regali

9. *Esercizio*: Si codifichi il testo **abbracaddabbrra!!** usando una codifica di Huffman, da calcolarsi in via preliminare sulle relative frequenze dei caratteri così come essi appaiono nel testo.

10. *Esercizio*: Si costruisca un codifica di Huffman per il seguente insieme di caratteri e relative frequenze
a:4 b:5 c:7 d:8 e:10 i:12 r:20.
Usando la codifica costruita, si codifichi il testo **abbecedari**

11. *Esercizio*: Dato un alfabeto di caratteri $C = \{c_1, c_2, \ldots, c_n\}$, con relative frequenze $f(c_1), f(c_2), \ldots, f(c_n)$, si consideri il problema di determinare un albero di codifica ottimo per C. Si dimostri, con precisione, che esiste sicuramente un albero ottimo T in cui i due caratteri $x, y \in C$ di frequenze minime appaiono nell'albero T alla profondità massima, e sono fratelli.

Partendo dunque che, sulle ali, l'Albero Ottimo è un tre e il livello più basso del albero.
Supponiamo di un'albero T, con oppure di livello più basso, ma è vicino il livello più basso.
Inoltre, nell'albero T l'Ottimo, e è basso primo sorriso.
Sia $h(T)$ l'angolo dell'Ottimo che appare. $\sum_{c \in C} f(c) \cdot h(c)$
Sia T' l'albero.

Ci sono $h(T) - h(T') = \sum_{c \in C} f(c) \cdot h(c) - \sum_{c \in C} f(c) \cdot h'(c)$
$= f(x) \cdot h(x) + f(y) \cdot h(y) - f(x) \cdot h'(x) - f(y) \cdot h'(y)$ Sia $h(x) = h(y) = h$ e $h'(x) = h'(y) = h - 1$
$= f(x) \cdot h + f(y) \cdot h - f(x) \cdot (h-1) - f(y) \cdot (h-1)$
$= f(x) \cdot h + f(y) \cdot h - f(x) \cdot h + f(x) - f(y) \cdot h + f(y)$
$= f(x) + f(y)$ Ma $h(T) = h(T') + f(x) + f(y)$ e $h(T) \geq h(T')$
Ci sono per $h(T) = h(T')$ e $h(T) \geq h(T')$ Dunque T' è ottimo dato e sempre il livello più basso.

Se T' è ottimo, allora $h(T) = h(T')$. Dunque T' è ottimo dato e sempre il livello più basso.

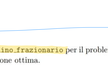
12. *Esercizio*: Si descriva in dettaglio l'algoritmo greedy per il Partizionamento delle Attività, e si provi che esso produce una soluzione ottima.

(Gantt Activities #17)

2. *Esercizio*: Si intende riparare un tubo di gomma che ha n fori su di esso. Si rappresenti l'i-esimo ciascun foro con il numero x_i, dove x_i è la distanza, in centimetri, del foro i-esimo dalla estremità sinistra del tubo. La riparazione avviene attaccando pezzi di nastro adesivo al tubo, dove ciascun pezzo di nastro adesivo è lungo k centimetri, k parametro dato. Il problema è quello di riparare il tubo (ovvero coprire tutti i buchi) utilizzando il *minor numero* di pezzi di nastro adesivo.

Si consideri il seguente algoritmo greedy per il problema appena enunciato: si ordinino i numeri x_i in modo tale che $x_1 \leq x_2 \leq \ldots \leq x_n$, indi si attacchi un pezzo di nastro adesivo al tubo, partendo dal buco x_1 fino a $x_1 + k$. Si fieri sui buchi rimasti scoperti, fin a quando tutti i buchi sono stati coperti.

Si provi che tale algoritmo copre tutti i buchi utilizzando minor numero di pezzi di nastro adesivo.



Sia quindi j l'attacco è la ultima fibra j pezzo di nastro x_i k.
Ci k x_j ultima fibra j pezzo x_i x_j $x_i + k$ Ma sopra di x_i x_j $x_i + k$ Ci sono
x_i x_j $x_i + k$
x_i x_j $x_i + k$
Si x_i $x_i + k$ x_i $x_i + k$ x_i $x_i + k$ x_i $x_i + k$

7. *Esercizio*: Si descriva in dettaglio l'algoritmo greedy per trovare l'ordinamento di lavori con tempo medio di attesa minimo e si provi che esso produce una soluzione ottima.

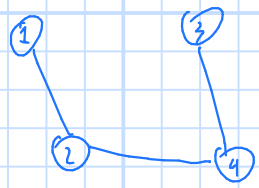
Input: un insieme di n lavori j dato a tempo p_j ciascuno lavoro.
Output: un ordinamento σ tale che $\sum_{j=1}^{n} C_j$ è minimo, dove C_j è il tempo di completamento del lavoro j.

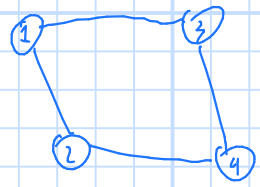
Ordinare il problema secondo $\frac{p_j}{w_j}$, in modo tale che $\frac{p_1}{w_1} \leq \ldots \leq \frac{p_n}{w_n}$.
Il tempo di completamento del lavoro j è $C_j = \sum_{i=1}^{j} p_i$ e quindi $\sum_{j=1}^{n} w_j C_j = \sum_{j=1}^{n} w_j \sum_{i=1}^{j} p_i$.
Questo è una somma.
  Ci compilare Θ (ordinamento) $\Theta(n \log n)$
  $\Theta(n)$

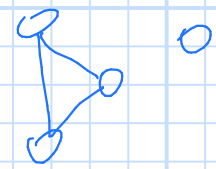
Ci dimostriamo Θ (che è un ordine ottimale).
  Ci vuole ordinare Θ Θ (quindi minimo) Θ Θ Θ.
  Ci vuole un ordine ottimo. Ma non sono sicuro che sia dato Θ vero Θ.
  Contraddizione: Θ Θ Θ Θ Θ Θ Θ Θ.
  Ci Θ Θ Θ Θ Θ Θ Θ

ESERCIZI 5

1. *Esercizio*: Progettare ed analizzare un algoritmo che, prendendo in input un grafo non orientato $G = (V, E)$, determina se G contiene o meno cicli. La complessità dell'algoritmo deve essere $O(|V|)$, *indipendentemente* da $|E|$.







Ciclo

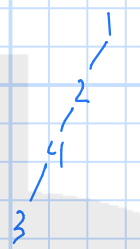
1 : 2
2 : 4, 1
3 : 4
4 : 2, 3

No cicli

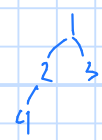
1 : 2, 3
2 : 4, 1
3 : 4, 1
4 : 2, 3

Ciclo

DFS



BFS




```
Alg(G)
∀u∈V, Scoperto[u]= False
Aggiungiamo un nodo s∈V alla coda Q
Scoperto[s]= True
parent[s]=∅      ← per ogni nodo, ci salviamo il suo parent
While Q non è vuoto
    Estrai u da Q
    Per ogni nodo x adiacente a u
        if Scoperto[x] == False
            Scoperto[x] = True
            parent[x]=u
            Inserisci x in Q
        else
            if parent[u] ≠ x
                return True
return False
```

L'algoritmo effettua una BFS, durante la quale viene salvato il parent di ogni nodo. Se un nodo x viene esplorato da un suo adiacente u, il parent di x sarà u. Se in una successiva esplorazione due nodi già esplorati i e j sono adiacenti e non è valido parent[i] == j. Allora c'è un ciclo.

La complessità è $O(|V|)$ perché l'algoritmo si ferma non appena trova un ciclo e se $|E| \geq |V|$ ci sarà sicuramente un ciclo che verrà rilevato dopo aver analizzato al massimo un insieme di A edges, dove $A \subset E$, $|A| \leq |V|$.

G deve essere rappresentato tramite liste di adiacenza, altrimenti con una matrice di adiacenze la complessità è $\Theta(|V|^2)$ nel peggiore dei casi

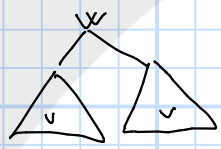
2. *Esercizio*: Dato un grafo non diretto $G = (V, E)$, e tre vertici $u, v, w \in V$, descrivere ed analizzare un algoritmo che ritorna `True` se esiste un cammino in G che parte da u ed arriva a v, passando per w, ritorna `False` altrimenti. Il cammino può anche contenere vertici ripetuti.

```
Alg(G, u, w, v)
  Q ← w
  Scoperto[w] = True
  while Q ≠ ∅
    estrai il nodo x da Q
    per ogni nodo y adiacente a x
      if Scoperto[y] == False
        Scoperto[y] = True
        Q ← y

  if Scoperto[u] == True and Scoperto[v] == True
    return True

  return False
```

- BFS esplora G in ampiezza a partire da il nodo w e scopre tutti i cammini verso gli altri nodi a partire da w. Essendo G non diretto, se esiste un cammino da w a x, allora lo si può percorrere a contrario per andare da x a w.
- BFS costruisce un albero T con radice w in cui a ogni profondità ci sono i nodi raggiungibili da w e la profondità rappresenta la distanza da w al nodo i nella profondità p_i.
- Se u e v $\in V'$, dove $T(V', E') = BFS(G, w)$ allora c'è sicuramente un cammino da u a v che passa per w.

  Si possono avere due casi:

  1:



  In questo caso sappiamo che esiste un cammino da w a u e viceversa (poiché grafo non diretto) e da w a v e viceversa. E quindi esiste il cammino da u a w a v e quindi il cammino da u a v passando da w.

  2:

  Anche in questo caso ci sono i cammini da u a w e viceversa e da v a w e viceversa.

  Per andare da u a v passando da w basta partire da u "salire" a w e "riscendere" a v.

  In questo caso sicuramente si attraverseranno dei nodi più volte, ma ciò è permesso.

- BFS ha complessità $\Theta(|V|+|E|)$.

  La ricerca di due elementi in un insieme richiede $O(|V|)$.

  La complessità totale è dominata da BFS e quindi l'algoritmo ha complessità $\Theta(|V|+|E|)$.

3. *Esercizio*: Dato un grafo diretto $G = (V, E)$, e due vertici $u, v \in V$, descrivere ed analizzare un algoritmo che ritorna `True` se esiste un cammino in G che parte da u ed arriva a v, e contemporaneamente esiste un cammino in G che parte da v ed arriva a u, ritorna `False` altrimenti.

```
Alg( G, u, v)
    T(V, E) = BFS(G, u)
    T'(V', E') = BFS(G, v)
    if( v∈V and u∈V')
        return True
    else
        return False
```

andrebbe bene anche DFS, in quanto non viene chiesto il cammino minimo

Si potrebbe fare anche $BFS(G^R, v)$ in quanto calcola tutti i nodi che hanno un cammino verso u

- $BFS(G, s)$ esplora in ampiezza tutti i nodi raggiungibili a partire da s, ne calcola le distanze e costruisce un albero $T(V, E)$ nel quale ci saranno tutti i nodi raggiungibili a partire da s e ogni profondità rappresenta la distanza da s agli altri nodi.

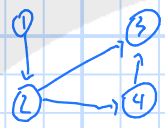
- Nell'algoritmo viene effettuata $BFS(G, u)$ per calcolare i nodi raggiungibili a partire da u e salvato l'albero generato in $T(V, E)$. Dopodiché viene effettuata $BFS(G, v)$ per calcolare i nodi raggiungibili a partire da v e salvato l'albero generato in $T'(V', E')$.

- Se $v \in V$ allora esiste un cammino da u a v e se $u \in V'$ allora esiste un cammino da v a u. Se sono vere entrambe, allora esiste un cammino da u a v e da v a u.

- La complessità della BFS è $\Theta(|V| + |E|)$. La ricerca di un elemento in un insieme è $O(|V|)$. La complessità totale dell'algoritmo è $\Theta(|V| + |E|)$.

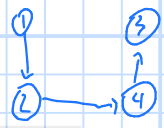
N.B: invece della $BFS(G, v)$ si poteva fare $BFS(G^R, u)$ in quanto essa calcola tutti i nodi che hanno un cammino verso u.

4. *Esercizio*: Sia $G = (V, E)$ un grafo diretto, $V = \{1, 2, \ldots, n\}$, rappresentato mediante la sua matrice delle adiacenze $A = [a_{ij}]$, in cui $a_{ij} = 1$ se e solo se esiste un arco *diretto* dal vertice i al vertice j. Diremo che un vertice $u \in V$ è un *pozzo* se u non ha archi uscenti da esso ed ha $n - 1$ archi entranti. Determinare in tempo $O(n)$ se G ha un pozzo o meno.



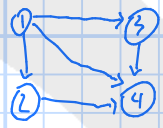
i \ j	1	2	3	4
1	0	1	0	0
2	0	0	1	1
3	0	0	0	0
4	0	0	1	0

no pozzo poichè:
- $2 \leq n-1 = 3$ archi entranti



i \ j	1	2	3	4
1	0	1	0	0
2	0	0	0	1
3	0	0	0	0
4	0	0	1	0

no pozzo perchè:
- 3 non ha n-1 archi entranti



i \ j	1	2	3	4
1	0	1	1	1
2	0	0	0	1
3	0	0	0	1
4	0	0	0	0

4 pozzo perchè
- non ha archi uscenti
- ha n-1 archi entranti

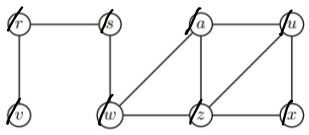
SI CERCA UNA ROW i TUTTI 0 E CHE ABBIA $A_{ji} = 1$, $j \neq i$ (n-1 nodi entranti)

```
Alg (A, n)

  i = 0
  j = 0

  while (i < n and j < n)

    if (A_ij == 1)

      i++        ← scarto il nodo i poichè ha un arco uscente

    else

      j++

  if (i > n)     ← significa che ogni nodo ha un arco uscente
    return False

  for (j=0; j<n; j++)
    if (A_ij == 1)   ← significa che il nodo i ha un arco uscente
      return False
    if (A_ji == 0 and j ≠ i)   ← controlliamo che il nodo i abbia n-1 archi entranti
      return False

  return True
```

- L'algoritmo all'inizio analizza la matrice di adiacenza e scarta (i+1) i nodi che hanno un arco uscente.
  Se alla fine dell'analisi sono stati scartati tutti i nodi ($i > n$), allora non esiste un pozzo.
  Altrimenti ho trovato un nodo che potrebbe essere un pozzo.
  Controllo che esso non abbia nodi uscenti e n-1 nodi entranti. Se una delle condizioni è falsa, riporto False, True altrimenti.

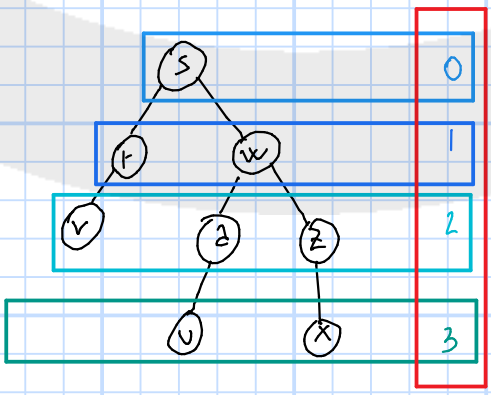
- I due cicli non sono innestati e richiedono rispettivamente $\Theta(n)$ e $\Theta(n)$, il resto è costante.
  Perciò la complessità totale dell'algoritmo è $\Theta(n)$.

5. *Esercizio*: Dato il grafo G rappresentato in figura



eseguire su di esso l'algoritmo BFS(G, s), indicando ad ogni passo dell'algoritmo sia l'arco considerato dall'algoritmo che la composizione della coda Q. Si calcoli inoltre sia l'albero BFS risultante che i valori $d[c]$ per ogni nodo c del grafo.

passo #	coda Q	arco considerato (i, c)	distanza di c da s d[c]=d[i]+1
1	[ ]		
2	[s]		d[s]=0
3	[r]	(s,r)	1
4	[r,w]	(s,w)	1
5	[w]	(r,s)	c già esplorato
6	[w,v]	(r,v)	2
7	[v]	(w,s)	c già esplorato
8	[v,a]	(w,a)	2
9	[v,a,z]	(w,z)	2
10	[a,z]	(v,r)	c già esplorato
11	[z]	(a,z)	c già esplorato
12	[z]	(a,w)	c già esplorato
13	[z,u]	(a,u)	3
14	[u]	(z,w)	c già esplorato
15	[u]	(z,a)	c già esplorato
16	[u]	(z,u)	c già esplorato
17	[u,x]	(z,x)	3
18	[x]	(u,a)	c già esplorato
19	[x]	(u,z)	c già esplorato
20	[x]	(u,x)	c già esplorato
21	[ ]	(x,u)	c già esplorato
22	[ ]	(x,z)	c già esplorato

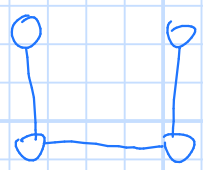


profondità di ogni nodo = distanza di ogni nodo dalla radice s.

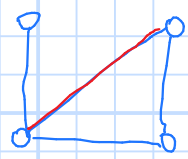
6. *Esercizio*: Progettare ed analizzare un algoritmo per il seguente problema:

Input: Un grafo non diretto $G = (V, E)$.

Output: "Sì" se esiste un arco $e \in E$ che si può rimuovere da G, in modo tale che $G' = (V, E - \{e\})$ sia connesso, "No", altrimenti.



No



Sì

Si può fare solamente se c'è un ciclo in G.

Basta risolvere il problema di controllare se G ha un ciclo, se ha allora esiste tale arco che si può rimuovere e G rimane connesso

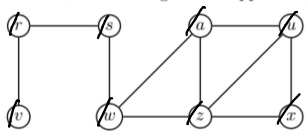
Algoritmo simile a quello dell'esercizio 1

```
Alg( G)
Q ← s     # s ∈ V, qualsiasi
Scoperto[s] = True
parent[s] = Q
while Q non è vuota
    estrai nodo u da Q
    Per ogni nodo x adiacente a u
        if Scoperto[x] == False
            Scoperto[x] = True
            Q ← x
            parent[x] = u
        else
            if parent[u] ≠ x
                return "Sì"
return "No"
```

- L'algoritmo in pratica esegue una BFS salvandosi il "parent" di ogni nodo non scoperto.
  - Se vi è un ciclo lo si scopre analizzando i nodi e ciò può avvenire prima di aver analizzato tutti i nodi oppure all'iterazione $|V|+1$.
  - Se vi è un ciclo (u,x), questo è l'arco che si può rimuovere senza perdere la proprietà che G è connesso.

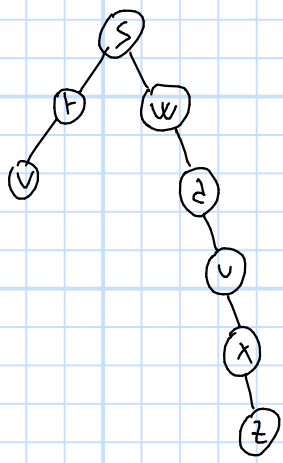
- La complessità dell'algoritmo è $\Theta(|V|)$ nel caso peggiore, perché come detto in precedenza, se c'è un ciclo, questo viene scoperto massimo all'iterazione $|V|+1$ e poi l'algoritmo termina. Se non c'è un ciclo, essendo il grafo non diretto, vi sono al massimo $|V|-1$ archi e quindi la complessità rimane lineare in $|V|$.

7. *Esercizio*: Dato il grafo G rappresentato in figura



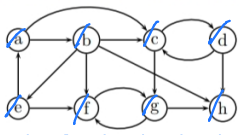
eseguire su di esso l'algoritmo DFS(G, s), indicando ad ogni passo dell'algoritmo sia l'arco considerato dall'algoritmo che la composizione dello stack Q. Si calcoli inoltre sia l'albero DFS risultante.

passo #	stack Q	arco considerato (i, c)	
1	[ ]		
2	[s]		
3	[r]	(s, r)	
4	[v]	(r, v)	
5	[ ]	(v, r)	già explorato
6	[ ]	(r, s)	già explorato
7	[w]	(s, w)	
8	[z]	(w, z)	
9	[ ]	(z, w)	già explorato
10	[u]	(z, u)	
11	[ ]	(u, z)	già explorato
12	[x]	(u, x)	
13	[ ]	(x, u)	già explorato
14	[z]	(x, z)	
15	[ ]	(z, x)	già explorato
16	[ ]	(z, u)	già explorato
17	[ ]	(z, z)	già explorato
18	[ ]	(u, z)	già explorato
19	[ ]	(z, z)	già explorato
20	[ ]	(w, s)	già explorato
21	[ ]	(w, z)	già explorato
22	[ ]	(w, z)	già explorato



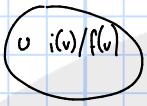
8. *Esercizio*:

Dato il grafo rappresentato in figura, eseguire su di esso l'algoritmo per il calcolo delle componenti fortemente connesse, spiegando passo passo le computazioni fatte.

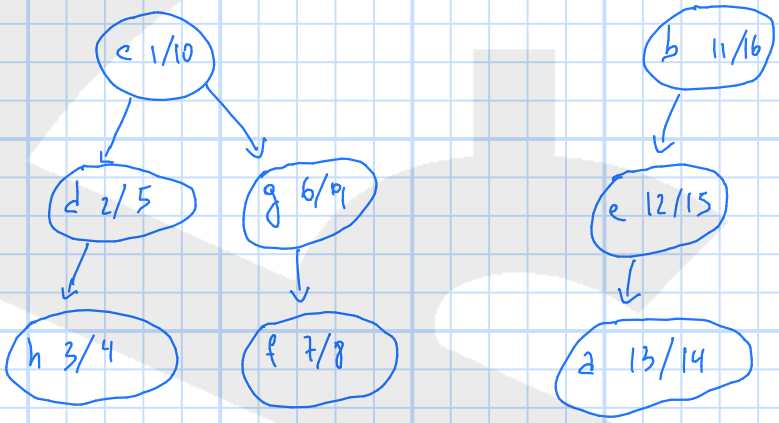


1. Eseguiamo DFS su G su ogni nodo $v \in V$ non già esplorato.

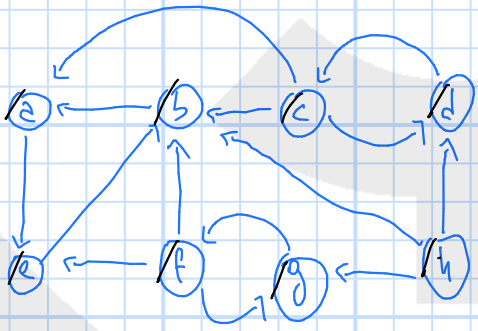
Mentre lo facciamo, ci salviamo $i(v)$ che indica quando abbiamo scoperto il nodo v e $f(v)$ che indica quando è finita la ricorsione sul nodo v.



← nodo v
$i(v)$ = quando viene scoperto
$f(v)$ = quando termina la ricorsione su di esso

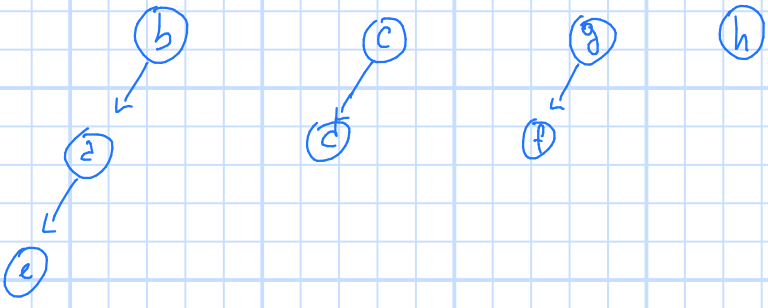


2. Calcoliamo G^R



$\not{v}$ = esplorato per il punto successivo

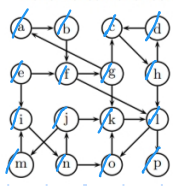
3. Calcoliamo DFS di G^R per ogni nodo $v \in V$ se esso non è stato già esplorato, ma invece di prenderne uno qualsiasi, prendiamo quello con $f(v)$ più grande.



Queste sono le componenti fortemente connesse di G

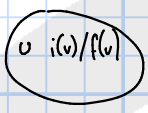
9. *Esercizio*:

Dato il grafo rappresentato in figura, eseguire su di esso l'algoritmo per il calcolo delle componenti fortemente connesse, spiegando passo passo le computazioni fatte.

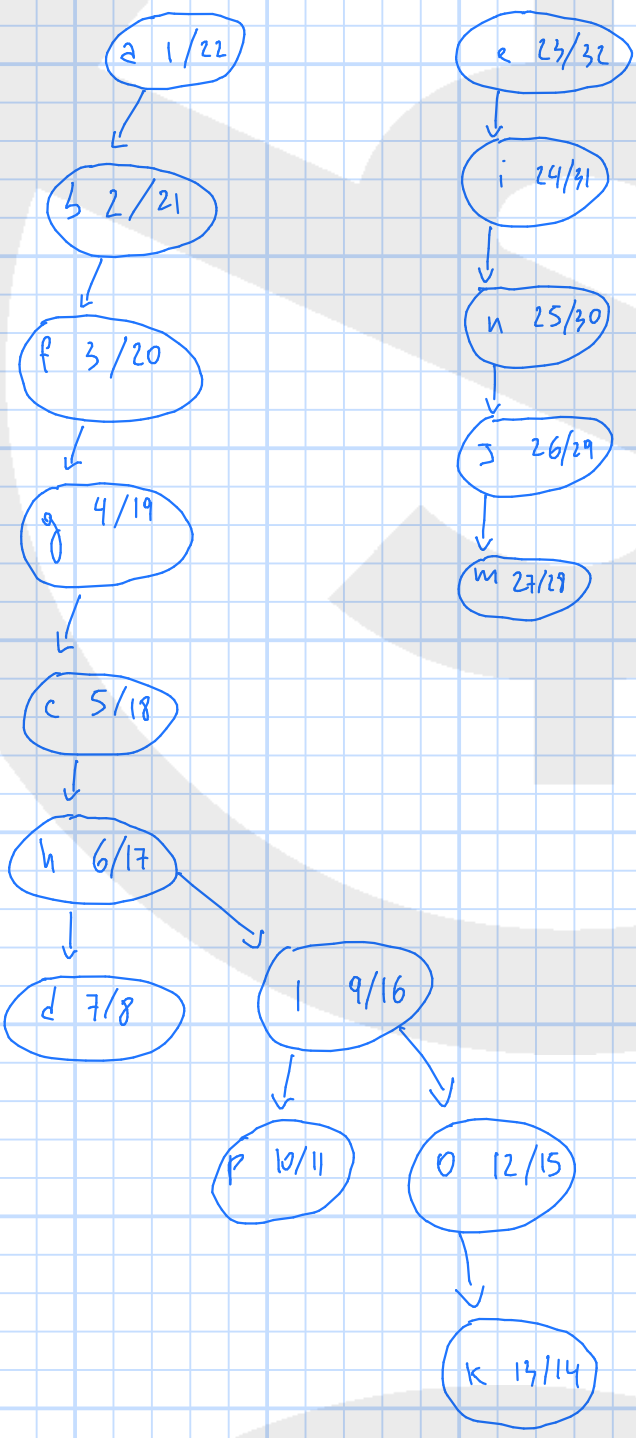


1. Eseguiamo DFS su G su ogni nodo $v \in V$ non già esplorato.

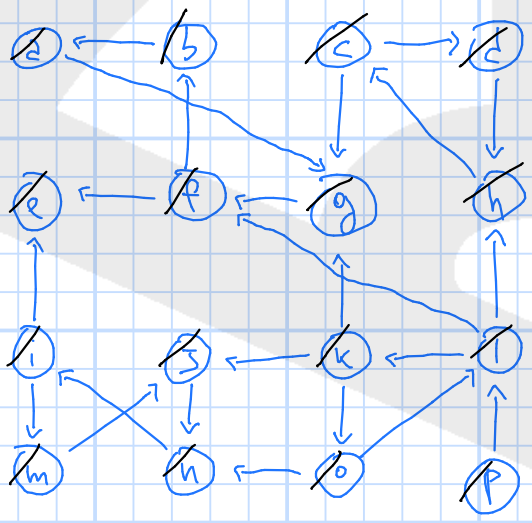
Mentre lo facciamo, ci salviamo $i(v)$ che indica quando abbiamo scoperto il nodo v e $f(v)$ che indica quando è finita la ricorsione sul nodo v.



← nodo v
$i(v)$ = quando viene scoperto
$f(v)$ = quando termina la ricorsione su di esso

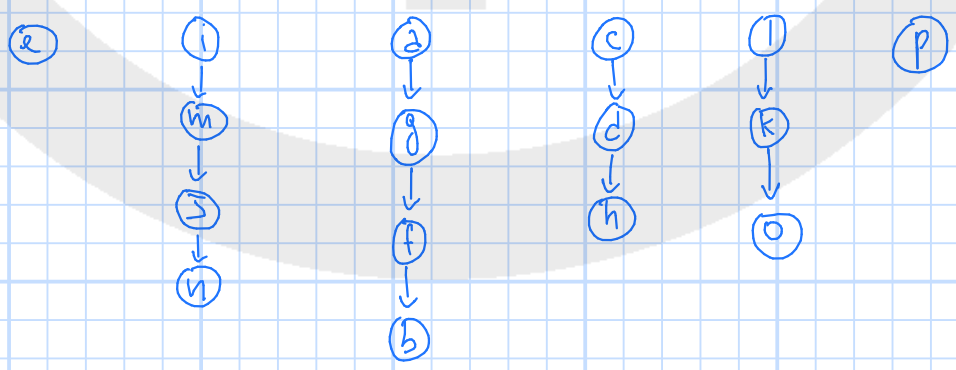


2. Calcolo di G^R



v = nodo scoperto per il passo successivo

3. Eseguiamo DFS su G^R su $v \in V$ se v non è stato già scoperto, ma non partiamo da un v qualsiasi, ma quello con $f(v)$ più grande.



Queste sono le componenti fortemente connesse di G

10. *Esercizio*: Dato un grafo $G = (V, E)$ non diretto, il suo diametro $D(G)$ è definito come la distanza tra i due vertici di G massimalmente distanti, se G è connesso, è definito come ∞ altrimenti. Formalmente:

$$D(G) = \begin{cases} \infty & \text{se } G \text{ non è connesso} \\ \max_{u,v \in V} d(u,v) & \text{altrimenti} \end{cases}$$

dove $d(u, v)$ è la distanza tra i due vertici u e v. Analogamente, l'eccentricità di un vertice u è definita come

$$e(u) = \begin{cases} \infty & \text{se } G \text{ non è connesso} \\ \max_{v \in V} d(u,v) & \text{altrimenti} \end{cases}$$

mentre il raggio $R(G)$ di G è definito come $R(G) = \min_{u \in V} e(u)$.

Si dia un algoritmo che, dato in input un grafo $G = (V, E)$ non diretto, calcoli $D(G)$ e $R(G)$.

N.B.: i cammini sono quelli più corti calcolati da BFS

$e(u)$ = massima distanza a cui si può arrivare dal nodo u

$D(G)$ = i due nodi u e v in G che distano più lontani

$= \max\{e(u) \mid u \in V\}$

$R(G) = \min\{e(u) \mid u \in V\}$

non si sta cercando la minima distanza tra due nodi, il problema sarebbe banale poiché la soluzione è sempre 1.

Si sta cercando tra i valori $e(u)$, $u \in V$, quello minimo.

https://youtu.be/YbCn8d4Enos

spiegazione di eccentricità, diametro e raggio

```
Alg(G(V,E))
  ∀u ∈ V
    Q = [ ]
    Q ← u
    d_u[u] = 0
    e(u) = 0
    Scoperto_u[u] = True
    T_u = (V_u = {u}, E_u = ∅)
    while Q non è vuota
      Estrai y da Q
      Per ogni nodo z adiacente a y
        if(Scoperto_u[z] == False)
          Scoperto[z] = True
          d_u[z] = d_u[y] + 1
          Q ← z
          V_u ← z
          E_u ← (y, z)
          if(e(u) < d_u[z])
            e(u) = d_u[z]
  if (V ≠ V_u)        ← controllo se è connesso, va bene qualsiasi u∈V
    return (∞, ∞)
  diameter = max{e(u) : u ∈ V}
  radius = min{e(u) : u ∈ V}
  return (diameter, radius)
```

- L'algoritmo esegue una BFS su ogni nodo $u \in V$ e si salva per ogni nodo la sua eccentricità $e(u)$.
  - Viene controllato se il grafo è connesso e nel caso non lo fosse, ritorna (∞, ∞).
  - Infine calcola il valore max tra i $e(u)$ e il valore min, questi rispettivamente saranno diametro e raggio.

- BFS ha complessità $\Theta(|V|+|E|)$
  - Vengono eseguite $|V|$ BFS quindi il ciclo più esterno ha complessità $\Theta(|V|(|V|+|E|)) = \Theta(|V|^2 + |V||E|)$
  - Il calcolo del max e min di $e(u)$ si può fare in una sola iterazione e quindi la complessità è $\Theta(|V|)$
  - La complessità totale dell'algoritmo è $\Theta(|V|^2 + |V||E|)$.

11. *Esercizio*: Data la rappresentazione mediante liste di adiacenza di un grafo *diretto* $G = (V, E)$, scrivere un algoritmo che determini il numero di archi uscenti da ogni vertice ed un separato algoritmo per determinare il numero di archi entranti in ogni vertice.

```
AlgUscenti(L1, L2, ..., L|V|)
    x = [0, 0 ... 0], |x| = |V|
    for(i=1; i ≤ |V|; i++)
    • x[i] = Li.size   (di solito le liste hanno un parametro size)
        Se non c'è size allora:
    • for ogni nodo nella lista Li
            x[i] = x[i] + 1
    return x
```

- L'algoritmo accede a ogni Lista di adiacenza, solamente questo ha complessità $\Theta(|V|)$.

Poi vi sono due casi a seconda che il parametro "size" esiste o meno:

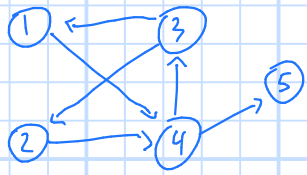
1. Se esiste, si accede al parametro in $\Theta(1)$ e quindi la complessità dell'algoritmo è $\Theta(|V|)$
2. Non esiste, allora bisogna scorrere ogni lista di adiacenza. Nel caso peggiore in cui ogni nodo ha un arco uscente a tutti gli altri, ossia $E = V \times V = \{(u,v) \mid u, v \in V\}$, allora per ogni lista bisogna scorrere $|V|$ elementi e quindi la complessità dell'algoritmo è $\Theta(|V|^2)$, <u>tuttavia si scrive $\Theta(|V| + |E|)$</u>

```
AlgEntranti(L1, L2, ..., L|V|)
    x = [0, 0, ..., 0], |x| = |V|
    for(i=1; i ≤ |V|; i++)
        for ogni nodo v nella lista Li
            x[v] = x[v] + 1
    return x
```

- L'algoritmo scorre le liste di adiacenza e per ogni lista scorre i nodi in essa presenti.
  - Ogni volta che un nodo v viene incontrato in una lista di adiacenza L_i, significa che $(i, v) \in E$, ossia c'è un arco uscente da i in v e quindi v ha l'arco entrante che parte da i e quindi si aumenta il contatore di archi entranti in v.
- Nel caso peggiore ogni nodo ha un arco uscente verso gli altri nodi e quindi $E = V \times V = \{(u,v) \mid u, v \in V\}$.

Quindi per ogni L_i $1 \le i \le |V|$ si scorriranno tutti i nodi e quindi la complessità dell'algoritmo è $\Theta(|V|^2)$

12. *Esercizio*: Il quadrato di un grafo orientato $G = (V, E)$ è il grafo $G^2 = (V, E^2)$ tale che $(u, w) \in E^2$ se e solo se $\exists v : (u, v) \in E \wedge (v, w) \in E$. In altre parole, se esiste un percorso di due archi fra i nodi u e w. Scrivere un algoritmo che, dato un grafo G rappresentato con matrice di adiacenza, restituisce il grafo G^2.



	1	2	3	4	5
1	0	0	0	1	0
2	0	0	0	1	0
3	1	1	0	0	0
4	0	0	1	0	1
5	0	0	0	0	0

```
Alg( a[|V|][|V|])
    a²[|V|][|V|] = [0 ... 0; ⋮ ⋱ ⋮; 0 ... 0]
    for (i=1; i≤|V|; i++)                    Θ(|V|³)
        for (j=1; j≤|V|; j++)                Θ(|V|²)
            if(a[i][j]==1)
                for (k=1; k≤|V|; k++)        Θ(|V|)
                    if(a[j][k]==1)
                        a²[i][k]=1

    return a²
```

- Per ogni arco $(u,v) \in E$ si aggiunge a G^2 gli archi $(u,x) : (v,x) \in E$.
  - L'algoritmo controlla che vi siano questi archi, il problema sta nel fatto che esso per ogni nodo u bisogna controllare se esso ha archi uscenti e per ogni arco uscente in v, bisogna controllare che v ha archi uscenti e per ogni arco uscente di v in x, lo si aggiunge a G^2 in quanto questo significa che esiste un cammino da u in x che passa per v e quindi usa due archi.

- La complessità nel caso peggiore di ogni nodo avente arco uscente verso gli altri nodi è $\Theta(|V|^3)$

13. *Esercizio*: Descrivere un algoritmo per il seguente problema. Dato in input un grafo orientato $G = (V, E)$ rappresentato tramite liste di adiacenza e un nodo $s \in V$, restituire il numero dei nodi in V raggiungibili da s che si trovano alla massima distanza da s.

```
Alg(G, s)
  count = 0
  Q ← s
  d[s] = 0
  Scoperto[s] = True
  maxD = 0
  while Q non è vuota
      Estrai u da Q
      Per ogni nodo x adiacente a u
          if (Scoperto[x] == False)
              Scoperto[x] = True
              d[x] = d[u] + 1
              Q ← x
              if (d[x] > maxD)
                  maxD = d[x]
  ∀u∈V
      if (d[u] == maxD)
          count = count + 1
  return count
```

Si potrebbe eliminare il ciclo finale e fare dei controlli e l'aumento o il reset di count direttamente dentro la BFS.

Si è scelto di non farlo perché si farebbero $|E|$ controlli e aumenti/reset di count.

Mentre il ciclo finale fa $|V|$ controlli e nel peggiore dei casi $|V|$ aumenti di count.

- L'algoritmo semplicemente effettua una BFS in cui a ogni esplorazione si salva la distanza dei nodi da s e salva nella variabile maxD la massima distanza da s a cui è riuscito ad arrivare.
  - Alla fine controlla il vettore delle distanze d[·] e conta quanti nodi sono alla massima distanza.

- La complessità di BFS è $\Theta(|V|+|E|)$.
  - Scorrere il vettore delle distanze richiede $\Theta(|V|)$.
  - La complessità dell'algoritmo è dominata dalla BFS e quindi è $\Theta(|V|+|E|)$

14. *Esercizio*: Descrivere un algoritmo per il seguente problema:

Input: un grafo diretto $G = (V, E)$ rappresentato mediante liste di adiacenza.

Output: per ogni nodo $v \in V$ il numero dei nodi w raggiungibili da v (ovvero il numero dei nodi $w \in V$ per i quali esiste un cammino orientato da v a w).

```
Alg(G)
  count = [0, ..., 0], |count| = |V|
  ∀s ∈ V
    Q = [ ]
    Q ← s
    Scoperto[u] = False, ∀u ∈ V
    Scoperto[s] = True
    while Q non è vuota
      Estrai u da Q
      Per ogni nodo x adiacente a u
        if(Scoperto[x] == False)
          Scoperto[x] = True
          Q ← x
          count[s] = count[s] + 1
  return count
```

- L'algoritmo semplicemente effettua una BFS per ogni nodo s e ogni volta che viene esplorato un nuovo nodo si aumenta il numero di nodi raggiungibili da s.

- La complessità di BFS è $\Theta(|V|+|E|)$, essa viene effettuata $|V|$ volte e perciò la complessità dell'algoritmo è $\Theta(|V|(|V|+|E|)) = \Theta(|V|^2 + |V||E|)$

15. *Esercizio*: Descrivere un algoritmo per il seguente problema:

Input: un grafo diretto $G = (V, E)$ rappresentato mediante liste di adiacenza, un nodo $v \in V$ ed un intero k

Output: il numero di nodi del grafo a distanza $\leq k$ da v.

```
Alg( G, v, k )
  Q ← v
  Scoperto[v] = True
  d[v] = 0
  count = 0
  while Q non è vuota
    Estrai v da Q
    Per ogni nodo x adiacente a v
      if( Scoperto[x] == False )
        Scoperto[x] = True
        Q ← x
        d[x] = d[v] + 1
        if( d[x] ≤ k )
          count = count + 1
        else
          break
return count
```

← se $d[x] > k$, essendo una BFS, tutti i successivi archi considerati avranno $d[x] + i > k$, $i \geq 0$ e quindi non ha senso andare avanti

- L'algoritmo effettua una BFS in G su v. A ogni nuovo nodo esplorato ne calcola la distanza da v, se essa è $\leq k$ allora aumenta il contatore. Se essa è $> k$, l'algoritmo ferma il ciclo poiché i successivi archi considerati avranno distanze $> k$, essendo una BFS.

- La complessità dell'algoritmo è data dalla BFS che è $\Theta(|V|+|E|)$.

16. *Esercizio*: Descrivere un algoritmo per il seguente problema:
Input: un grafo non diretto $G = (V, E)$ rappresentato mediante liste di adiacenza
Output: il numero di componenti connesse di G.

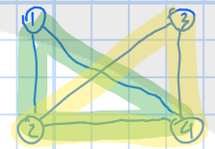
```
Alg ( G(V,E))
  count = 0
  ∀s ∈ V
    if (Scoperto[s] == False)      ← importante per non avere una complessità Θ(|V|(|V|+|E|))
      Scoperto[s] = True
      count = count + 1
      Q ← s
      while Q non è vuota
        Estrai u da Q
        Per ogni nodo x adiacente a u
          if (Scoperto[x] == False)
            Scoperto[x] = True
            Q ← x

  return count
```

- L'algoritmo effettua una BFS per ogni nodo che non è stato scoperto e a ogni BFS aumenta il contatore. Se alla fine della prima BFS vi sono nodi ancora non scoperti, allora vuol dire che il grafo ha più di una componente connessa.

- La complessità è data dalla BFS, la quale anche se eseguita più volte, non considera mai più di una volta i nodi e due volte gli archi.
Quindi la complessità dell'algoritmo è $\Theta(|V|+|E|)$

17. *Esercizio*: Dato un grafo non diretto $G = (V, E)$, un triangolo in esso è una tripla di vertici $a, b, c \in V$ tali che $(a, b), (b, c), (c, a) \in E$. Descrivere un algoritmo per il seguente problema:

Input: un grafo non diretto $G = (V, E)$ rappresentato mediante matrice di adiacenza;
Output: il numero di triangoli in G.



	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	0	1	0	1
4	1	1	1	0

```
Alg( a[|V|][|V|])
  count=0
  for( i=1 ; i≤|V| ; i++)
    for( j=i+1 ; j≤|V| ; j++)
      if( a[i][j]==1)
        for( k=j+1 ; k≤|V| ; k++)
          if ( a[j][k]==1 and a[k][i]==1)
            count = count+1

  return count
```

si usa $j=i+1$ e $k=j+1$ per evitare di contare gli elementi più volte.

- L'algoritmo controlla la matrice di adiacenza e la prima volta che un nodo u ha un arco verso un nodo v, $a[u][v]==1$, passa al nodo v alla ricerca dei suoi adiacenti, senza però considerare gli archi già visitati (perciò di $j=i+1$ e $k=j+1$).

  N.B: usando una matrice di adiacenza, si devono cercare gli archi esistenti in G, perciò la complessità non può essere $\Theta(|E|)$.

  Non appena trova un arco $(v,x) \in E$, ossia $a[v][x]==1$, controlla che $(x,u) \in E$, ossia $a[x,u]==1$. Nel caso esista tale arco, allora i nodi u, v, x formano un triangolo e quindi incrementa di 1 la variabile count.

- L'algoritmo ha complessità $\Theta(|V|^3)$ in quanto abbiamo 3 cicli innestati con condizione di uscita su $|V|$.

18. *Esercizio*: Dato un grafo non diretto $G = (V, E)$, con $V = \{1, \dots, n\}$ ed un array di interi $A[1 \dots n]$, G è detto ben colorato se per ogni $\{i, j\} \in E$ vale che $A[i] \neq A[j]$. Si descriva un algoritmo per il seguente problema:

Input: $G = (V, E)$ rappresentato mediante matrice di adiacenza, vettore A
Output: "SI", se G è ben colorato, "NO", altrimenti.

Si risolva l'esercizio anche sotto l'ipotesi che di G sia rappresentato mediante liste di adiacenze.

```
Alg(a[|V|][|V|], A[n])
  for(i=1; i≤|V|; i++)
    for(j=i+1; j≤|V|; j++)
      if(a[i][j]==1 and A[i]==A[j])
        return "NO"

  return "SI"
```

- L'algoritmo controlla la lista di adiacenze e controlla gli archi una volta sola, $j=i+1$. N.B: usando una matrice di adiacenza, si devono cercare gli archi esistenti in G, perciò la complessità non può essere $\Theta(|E|)$.
  Quando trova un arco, controlla se i suoi nodi hanno lo stesso colore, se è True allora restituisce "NO" per indicare che G non è ben colorato.
  Se non si verifica mai che $(i,j) \in E \wedge A[i] == A[j]$, allora ritorna "SI".

- La complessità è data dal controllare tutti gli archi in una matrice di adiacenza, che è $\Theta(|V|^2)$.

```
Alg(L1, ..., L|V|, A)
  for(i=1; i≤|V|; i++)
    ∀u∈Li
      if(A[i] == A[u])
        return "NO"

  return "SI"
```

- L'algoritmo per ogni lista di adiacenza del nodo $u \in V$ controlla che i nodi $v \in V$ nella sua lista se esistono nodi dello stesso suo colore, ossia $A[u] == A[v]$, e nel caso si verifica, restituisce "NO" a indicare che G non è ben colorato.
  Se questa situazione non si verifica mai, allora restituisce "SI".

- La complessità è data dal controllare tutti gli archi in una lista di adiacenza, che è $\Theta(|V|+|E|)$

19. *Esercizio*: Dato un grafo non diretto $G = (V, E)$, con $V = \{1, \dots, n\}$ ed un sottoinsieme $S \subseteq V$, S è detto un VERTEX COVER per G se per ogni $\{i, j\} \in E$ vale $\{i, j\} \cap S \neq \emptyset$. Si descriva un algoritmo per il seguente problema:

Input: $G = (V, E)$ rappresentato mediante matrice di adiacenza, sottoinsieme $S \subseteq V$
Output: "SI", se S è un VERTEX COVER "NO", altrimenti.

Si risolva l'esercizio anche sotto l'ipotesi che di G sia rappresentato mediante liste di adiacenze.

Vertex-Cover: un sotto insieme S di vertici di V per cui non vi sono archi, ossia $\forall u,v \in S, (u,v) \notin E$

```
Alg(a[|V|][|V|], S)
    ∀u,v ∈ S
        if(a[u][v]==1)
            return "NO"

    return "SI"
```

- L'algoritmo prende ogni volta due nodi u,v da S e controlla se $(u,v) \in E$, ossia a[u][v]==1. Nel caso tale arco esista, allora S non è un VertexCover e quindi restituisce "NO". Se questa situazione non si verifica mai, allora restituisce "SI".

- La complessità è data dallo scorrimento dell'insieme S che è $\Theta(|S|)$. Controllare se un arco esiste in una matrice di adiacenze richiede $\Theta(1)$.

```
Alg(L1,...,L|V|, S)
    ∀u,v ∈ S
        scorri i nodi x in Lu
            if(x==v)
                return "NO"

    return "SI"
```

- L'algoritmo prende ogni volta due nodi u,v in S e controlla se nella lista di adiacenze di u, ossia L_u, è presente il nodo v, se è presente questo significa che $(u,v) \in E$ e quindi S non è un VertexCover e quindi restituisce "NO". Se questa situazione non si verifica mai, allora restituisce "NO".

- Lo scorrimento di S richiede $\Theta(|S|)$, tuttavia a ogni scorrimento controlliamo una lista di adiacenti che nel peggiore dei casi $|L_i| = |V|$ e quindi la complessità dell'algoritmo è $\Theta(|S||V|)$.

20. *Esercizio*: Descrivere un algoritmo per il seguente problema:

Input: un grafo diretto $G = (V, E)$ rappresentato mediante liste di adiacenza, due nodi $v, w \in V$.

Output: Il numero di nodi raggiungibili da v che si trovano alla stessa distanza sia da v che da w.

Alg(G, v, w)

$d_v, T(V_v, E_v) = BFS(G, v)$ $\quad \Theta(|V|+|E|)$

$d_w = BFS(G, w)$ $\quad \Theta(|V|+|E|)$

$\forall x \in V_v$ $\quad \Theta(|V|)$ poiche $V_v \subseteq V$

if $(d_v[x] == d_w[x])$ $\quad$ se $x \notin V_w \Rightarrow d[x] = \infty$

count = count + 1

return count

SPIEGAZIONE DI BFS

- L'algoritmo esegue BFS su G e v e si fa ritornare l'albero costruito e il vettore delle distanze.

Dopodichè esegue BFS su G e w e si fa restituire il suo vettore delle distanze.

Per ogni nodo x in V_v, ossia i nodi raggiungibi da v, controlla che x dista ugualmente da v e w e se è vero incrementa il contatore di 1. Se x non è raggiungibile da w allora $d_w[x] = \infty$ e quindi x non sarà considerato.

- L'algoritmo ha complessità $\Theta(|V|+|E|)$ in quanto vengono eseguite 2 BFS e alla fine viene fatto scorrere l'insieme V_v, il quale è $V_v \subseteq V$ e quindi $|V| \leq |V|$ e quindi $\Theta(|V|)$.

La complessità totale è $\Theta(|V|+|E|)$

21. *Esercizio*: Si descriva l'algoritmo per il calcolo dell'ordinamento topologico di un DAG, argomentando la correttezza e valutandone la complessità di tempo Giustificare con *precisione* le relative affermazioni effettuate.

Input: Grafo $G=(V,E)$, G è un DAG.
Un numero i di partenza per l'ordinamento topologico

Output: vettore $n[u], u \in V$ che contiene l'ordinamento topologico, ossia il numero assegnato a ogni nodo

Alg($G(V,E)$, i)
  if ($V == \emptyset$)
    return n
  Trova il nodo u senza archi entranti, ossia il nodo u che non appare in nessuna lista di adiacenza
  $n[u]=i$
  Alg($G(V-\{u\}, E-\{(u,x)\in E : x \in V\}, i+1$)

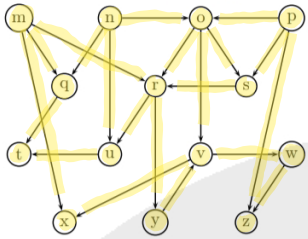
- Essendo G un DAG, esso non ha cicli e quindi esiste un nodo v che non ha archi entranti, ossia $\forall x \in V, (x,v) \notin E$.
Prendiamo il primo che troviamo, in quanto può essere presente più di 1, e gli assegniamo un valore numerico i.
Dopodicchè richiamiamo l'algoritmo su $G-\{v\}$ e incrementiamo i.
L'algoritmo si ferma quando a tutti i nodi sarà assegnato un valore numerico per l'ordinamento topologico.

- La correttezza dell'algoritmo è data dal fatto che essendo G un DAG, esso non ha cicli e almeno un nodo u senza archi entranti. Rimuovere il nodo u da G significa rimuovere i suoi archi uscenti e rimuoverlo dall'insieme V.
Rimuovendo archi, questo non può creare un ciclo e quindi esisterà un altro nodo y che non ha archi entranti. Se tale y non esiste dopo aver rimosso gli archi (u,x), allora G conteneva un ciclo, ma questo è impossibile poiché G è un DAG.

- La complessità è data dal trovare il nodo u senza archi entranti ed eliminare gli archi $(u,x) \in E, \forall x \in V$.
  - La prima operazione richiede tempo $\Theta(|V|+|E|)$ in quanto si deve controllare ogni lista di adiacenza.
  - La seconda operazione può richiede tempo $\Theta(1)$ in quanto si tratta solamente di eliminare la lista di adiacenza di u.
  Non bisogna andare a controllare le altre liste di adiacenza, in quanto u per definizione non ha archi entranti.

  - La prima operazione viene effettuata $|V|$ volte quindi la complessità dell'algoritmo è $\Theta(|V|(|V|+|E|)) = \Theta(|V|^2+|V||E|)$

22. *Esercizio*: Dato il grafo rappresentato in figura



determinarne un ordinamento topologico, applicando passo passo l'algoritmo visto a lezione.

$n(m)=1$

$n(n)=2$

$n(q)=3$

$n(p)=4$

$n(o)=5$

$n(s)=6$

$n(r)=7$

$n(u)=8$

$n(t)=9$

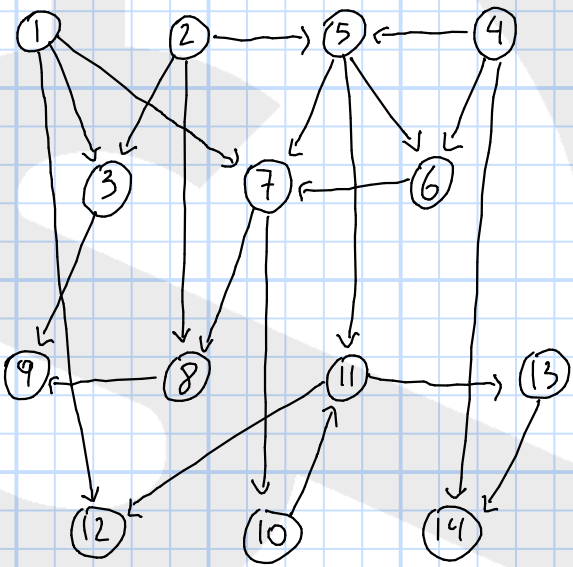
$n(y)=10$

$n(v)=11$

$n(x)=12$

$n(w)=13$

$n(z)=14$



23. *Esercizio*: Si provino le seguenti affermazioni:

(a) Se G ammette un ordinamento topologico allora G è necessariamente senza cicli

(b) Se G è un DAG allora ha un vertice senza archi entranti.

a) Per definizione di ordinamento topologico si ha che un nodo v ha archi uscenti solamente verso nodi con ordinamento topologico più grande, ossia $\forall x \in V$, se $(v,x) \in E$ allora $n(v) < n(x)$

Se avessimo un ciclo, ad esempio se $\{(u,v),(v,z),(z,u)\} \subseteq E$ e per assurdo supponessimo che esiste un ordinamento topologico, allora avremo che:

$n(u) < n(v)$

$n(v) < n(z)$

$n(z) < n(u) \rightarrow n(z) < n(v)$ ASSURDO

$n(z) < n(u)$ implica che $n(z) < n(v)$ poichè $n(u) < n(v)$, ma questo è un assurdo poiché $(v,z) \in E$ e quindi $n(v) < n(z)$.

- L'assurdo è nato dal fatto di aver supposto che un grafo con un ciclo avesse un ordinamento topologico, quindi affinchè ci sia un ordinamento topologico in un grafo, esso <u>non</u> deve avere cicli.

b) Per definizione di DAG, esso è un grafo diretto senza cicli.

Per assurdo supponiamo che ogni nodo $v \in V$ abbia un arco entrante in un DAG, ossia $\forall v \in V, \exists x \in V : (x,v) \in E$.

Ad esempio $\{(u,v),(v,z),(z,u)\} \subseteq E$

Ogni nodo ha un arco entrante, ma notiamo anche che gli archi formano un ciclo, ma questo è un ASSURDO.

Avevamo supposto che fosse un DAG, cioè senza cicli.

- L'assurdo è nato dal fatto di aver supposto che in un DAG tutti i nodi abbiano un arco entrante.

Quindi affinchè un DAG sia tale, deve esistere almeno un nodo senza archi entranti.

24. *Esercizio*: Siano $C_1, \ldots, C_k$ le componenti fortemente connesse di un grafo diretto $G = (V, E)$. Dato il grafo $G' = (V', E')$, dove $V' = \{C_1, \ldots, C_k\}$ e c'è un arco diretto (C_i, C_j) dal "vertice" C_i al "vertice" C_j se e solo se, *nel grafo* $G = (V, E)$ esiste un qualche arco della forma (u, v), dove $u \in C_i$ e $v \in C_j$.

Si provi che $G' = (V', E')$ è un DAG, ovvero non ha cicli.

Se per assurdo G' avesse un ciclo allora un ciclo minimo sarebbe della forma: $\{(C_u, C_v), (C_v, C_z), (C_z, C_u)\} \subseteq E'$.

Se ciò fosse vero, allora ci sarebbe un cammino da C_u a C_v e da C_v a C_u passando per C_z, ossia $C_v \to C_z \to C_v$.

Per definizione di componente fortemente connessa, se c'è un cammino da u a v e da v a u, allora $u, v \in C(u) = C(v)$.

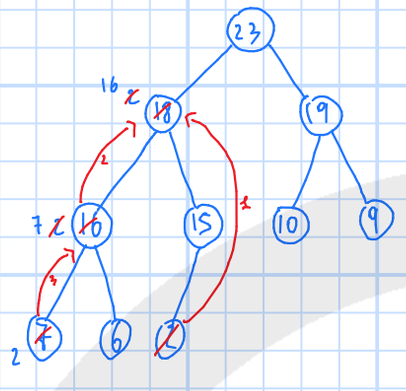
Quindi C_u e C_v sono la stessa componente fortemente connessa, ma questo è ASSURDO poiché per ipotesi erano componenti fortemente connesse di G e quindi disgiunte.

- L'assurdo è nato dal fatto di aver supposto che ci fosse un ciclo in G' che è un DAG.

Quindi in un DAG non ci può essere un ciclo.

25. *Esercizio*: Sia dato un MAX-HEAP rappresentato da un array $A[1 \dots n]$. Si scriva un procedura (*in pseudocodice, per favore*) che, in tempo $O(\log n)$, avendo in input A ed un indice i, $1 \leq i \leq n$, cancelli l'elemento $A[i]$ dallo heap. L'array in output dovrá ancora essere un MAX-HEAP. Si illustri la esecuzione dell'algoritmo sul seguente heap, con $i = 2$ (*L'algoritmo puó usare, al suo interno, chiamata a procedure note viste a lezione, purché se ne spieghi chiaramente l'utilizzo e la funzione.* Come suggerimento, si proceda in modo analogo alla cancellazione dell'elemento nella radice dello Heap, così come visto a lezione).

1	2	3	4	5	6	7	8	9	10
23	18	19	16	15	10	9	7	6	2



$i \in \mathbb{N}$ indica l'ordine delle operazioni

Alg(A, i)

Sostituisci A[i] con A[n]

Heap-size = Heap-size - 1

Max-Heapify (A, i)

return A

N.B.: non c'è bisogno di controllare che sostituendo A[i] con A[n], l'elemento A[i] durante il Max-Heapify possa salire in A[1] ... A[i-1] poiché essendo uno Max-Heap, $A[n] \leq A[i] \dots A[\lfloor n/2 \rfloor]$ poiché è la proprietà di base.

- Max-Heapify prende in input l'array che rappresenta il Max-Heap e l'indice dell'elemento che potrebbe aver interrotto la proprietà di base di uno Max-Heap, ossia che il nodo padre ha valore maggiore uguale ai suoi figli, ossia $A[i] \geq A[2i] \wedge A[i] \geq A[2i+1]$.
  - Per fare ciò, esso confronta $A[i]$ con $A[2i]$ e $A[2i+1]$ e se la proprietà è valida non fa niente, altrimenti scambia $A[i]$ con $\max\{A[2i], A[2i+1]\}$.

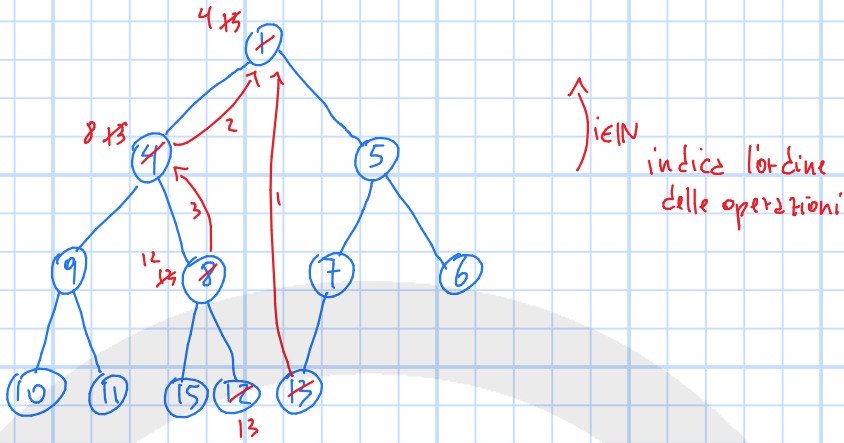
  E segue questa operazione fin quando la proprietà di base dello Max-Heap viene ristabilita.

- La complessità è $O(\log n)$ poiché al massimo si scende per h volte, dove h è l'altezza dell'albero ed è stato provato che $h = \log n$

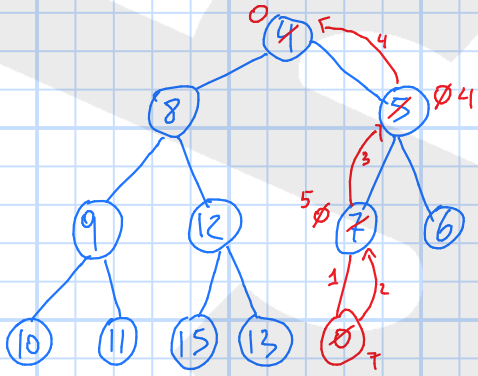
26. *Esercizio*: Si dia la rappresentazione mediante albero del MIN-HEAP:

$A = [1, 4, 5, 9, 8, 7, 6, 10, 11, 15, 12, 13]$.

Si ridisegni sia l'albero che il vettore corrispondente dopo l'esecuzione delle seguenti operazioni:
(a) HEAP-EXTRACT-MIN(A), seguita da (b) HEAP-INSERT$(A, 0)$

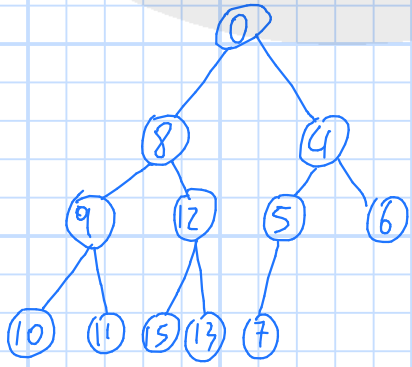


Dopo Heap-Extract-Min(A)



$A = [4, 8, 5, 9, 12, 7, 6, 10, 11, 15, 13]$

Dopo Heap-Insert(A, 0)



$A = [0, 8, 4, 9, 12, 5, 6, 10, 11, 15, 13, 7]$

27. *Esercizio*: Si assuma che n numeri diversi tra di loro siano memorizzati in un MIN-HEAP. Si dia un algoritmo che in tempo $O(1)$ trovi il terzo elemento piú piccolo di A. *Sotto l'ipotesi che i sia costante*, si dica come l'i-esimo elemento piú piccolo di A possa essere trovato in tempo $O(1)$.

- Prendo il minimo degli elementi tra il 1° e il 2° livello (altezza)

  Confronto di $2^1 + 2^2 = 6$ elementi $\Rightarrow \Theta(6) \Rightarrow$ costante.

- Faccio i Extract-Min.

  Complessità $= \Theta(i \log n)$

  Esiste una soluzione in tempo $\Theta(i \log i)$ che fa uso di priority queue.

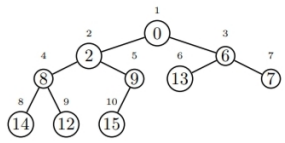
28. *Esercizio*: Dato un array di n numeri $A[1 \dots n]$, si presenti un algoritmo che, avendo in input l'array A, ritorni TRUE se A rappresenta un MIN-HEAP, ritorni il piú piccolo indice i su cui la proprietá dello heap é violata, altrimenti. Si valuti la complessitá dell'algoritmo proposto.

```
Alg(A)
  i=1
  while(True)
    if 2i>n                    ← si fermerà dopo l'ultimo nodo interno/padre se è un Min-Heap
      return True
    else if (A[i]>A[2i] or A[i]>A[2i+1])
      return i
    i=i+1
```

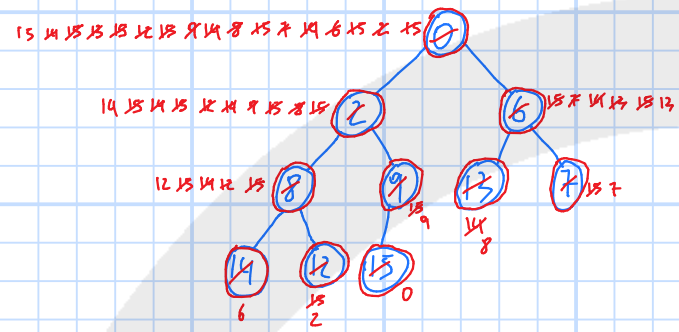
- L'algoritmo controlla che se il valore del padre è maggiore di uno dei figli, allora la proprietà di base di un Min-Heap è stata violata e ritorna l'indice del padre.
  - Vi è un controllo se $2i>n$ o $2i+1>n$ poiché vogliamo controllare solamente i nodi che hanno figli e non le foglie.
  - Se arriviamo che nell'iterazione precedente abbiamo controllato l'ultimo padre ed esso rispettava la proprietà, allora restituiamo True.

- L'algoritmo viene eseguito per il numero di padri nello Heap, nel caso peggiore il numero è $n-2^{\log n}$. La complessità è quindi $\Theta(n)$

29. *Esercizio*: Si esegua l'algoritmo HEAPSORT(A) sullo heap



descrivendo esplicitamente tutti i passi dell'algoritmo.



L'algoritmo scambia la radice con l'ultima foglia e decrementa la size dello Heap di 1 per non considerare più quella ultima foglia. Dopodiché esegue Min-Heapify(A, 1) per ristabilire la proprietà di base dello Min-Heap, ossia il padre deve avere valore minore o uguale a quello dei suoi due figli.

Queste operazioni vengono eseguite |V|-1 volte e alla fine A sarà un array ordinato in senso decrescente.

#	Elementi scambiati	A (dopo Min-Heapify) [heap-size | ..]
1	0 ↔ 15	[2, 8, 6, 12, 9, 13, 7, 14, 15 | 0]
2	2 ↔ 15	[6, 8, 7, 12, 9, 13, 15, 14 | 2, 0]
3	6 ↔ 14	[7, 8, 13, 12, 9, 14, 15 | 6, 2, 0]
4	7 ↔ 15	[8, 9, 13, 12, 15, 14 | 7, 6, 2, 0]
5	8 ↔ 14	[9, 12, 13, 14, 15 | 8, 7, 6, 2, 0]
6	9 ↔ 15	[12, 14, 13, 15 | 9, 8, 7, 6, 2, 0]
7	12 ↔ 15	[13, 14, 15 | 12, 9, 8, 7, 6, 2, 0]
8	13 ↔ 15	[14, 15 | 13, 12, 9, 8, 7, 6, 2, 0]
9	14 ↔ 15	[15, 14, 13, 12, 9, 8, 7, 6, 2, 0]

30. *Esercizio*: Dato un array A di n elementi, tutti distinti tra di loro, rappresentante un MAX-HEAP.

(a) In che posizione di A compare l'elemento massimo?

(b) In quali posizioni di A può comparire il secondo massimo?

(c) Per $k = 3, 4$, in che posizioni di A può comparire il k-esmo elemento piú grande di A?

(d) In che posizioni di A può comparire l'elemento minimo di A?

Giustificare le risposte.

a) Essendo un Max-Heap, l'elemento con valore massimo è la radice, ossia $A[1]$

b) Il secondo massimo può comparire in $A[2]$ o in $A[3]$. Non può comparire in $A[1]$ poiché essendo un Max-Heap, il padre $A[i]$ è maggiore o uguale dei figli $A[2i]$ e $A[2i+1]$. Non può essere uguale in quanto per ipotesi gli elementi in A sono distinti.

- Non può comparire in $A[4] .. A[n]$ poiché hanno valori minori rispetto a $A[2]$ e $A[3]$ per costruzione.

c) Il k-esimo elemento più grande in A per $k=3$ compare in $A[2]$ o in $A[3]$ perché essendo un Max-Heap, la radice è il 1° valore più grande in A e uno dei suoi figli sarà il 3°. Non può essere un nipote della radice poiché la proprietà di base di uno Max-Heap è che i padri hanno valore maggiore dei figli.

- Il 4° elemento più grande in A comparirà tra $A[4]$ e $A[7]$ per le stesse motivazioni espresse precedentemente.

probabilmente sbagliato

d) L'elemento minimo può comparire tra gli elementi che si trovano alla penultima e ultima altezza di A, ossia tra $A[\sum_{i=0}^{\log n - 2} 2^i] \dots A[n]$ — forse corretto

ESERCIZI 6

1. *Esercizio*: Si presenti l'algoritmo di Dijkstra per il calcolo dei cammini minimi in un grafo, si argomenti la sua correttezza e se ne valuti la sua complessità di tempo.

$S = \{s\}$, $d[s] = 0$

$\forall u \in V-S$, $d'[u] = \infty$

while $S \neq V$

tra tutti i nodi $w \in V-S$ per qualche $v \in S$ tale che $(v,w) \in E$

lunghezza arco e\ costo arco e

seleziona tra i nodi w, il nodo x tale che $d'[x] = \min_{e=(v,x)} (d[v] + \ell(e))$ è il più piccolo possibile

$d[x] = d'[x]$

$S \leftarrow x$

- La correttezza dell'algoritmo si basa sul fatto che il cammino che va da s, passa per v e arriva in x sia quello di costo minore.

  Prima di tutto, vale che $d'[x] \leq d'[y]$, se non fosse vero, allora tra tutti i $w \in V-S$ avremmo selezionato y e non x.

  Poiché $d'[x]$ è il più piccolo, allora vuol dire che se esistesse un altro percorso minimo da s a x, questo debba passare per $d'[y] + \ell((y,x))$. Ma sappiamo già $d'[y] \geq d'[x] \Rightarrow d'[y] + \ell((y,x)) \geq d'[x]$ e quindi non esiste un altro cammino di costo inferiore e la nostra soluzione è ottima.

- Per $V-S$ si può usare una coda a priorità min che possiamo chiamare Q, i cui elementi verranno disposti in base al valore $d'[\cdot]$

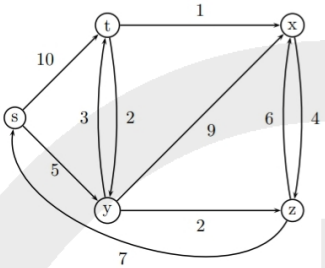
  A ogni estrazione di w da Q, ossia $d'[w] = \min_{e=(v,w)} (d[v] + \ell(e))$, bisognerà aggiornare al più $i \leq |E|$ valori in Q, ossia tutti i nodi $(w,z) \in E : d'[w] + \ell((w,z)) < d'[z]$, $z \in V-S$. Poiché all'inizio $d'[\cdot] = \infty$, si faranno solo Decrease-Key.

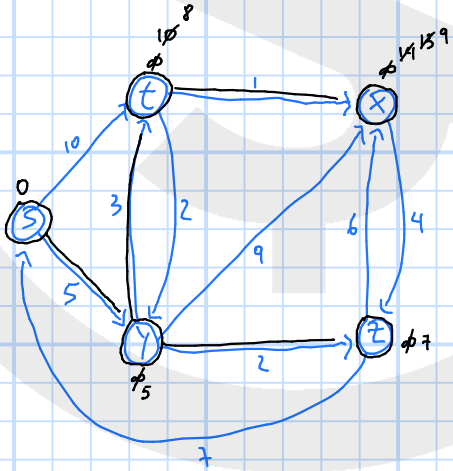
- L'operazione di Extract-Min richiede $\Theta(\log |V|)$, Decrease-Key idem.

  Extract-Min verrà effettuato $|V|-1$ volte, Decrease-Key viene effettuato al più 1 volta per ogni arco e quindi nel peggiore dei casi $|E|$ volte.

- La complessità totale dell'algoritmo è data dalle Decrease-Key effettuate $|E|$ volte e quindi è $\Theta(|E| \log |V|)$

2. *Esercizio*: Si esegua l'algoritmo di Dijkstra per il calcolo dei cammini minimi dal nodo s sul seguente grafo. Si esplicitino in dettaglio i passi che l'algoritmo esegue (ovvero, NON è sufficiente riportare semplicemente l'albero dei cammini minimi risultanti dall'esecuzione dell'algoritmo).





3. *Esercizio*: Sia $G = (V, E)$ un grafo diretto, con lunghezze $\ell(e) > 0$ per ogni arco $e \in E$. Si progetti e si analizzi (e si argomenti la correttezza) un algoritmo che presi in input un vertice $s \in V$ ed un arco $(u, v) \in E$, determini: **1.** se tra tutti i cammini di lunghezza minima da s a v ne esiste almeno uno che usi l'arco (u, v); **2.** se tra tutti i cammini di lunghezza minima da s a v ne esiste almeno uno che non usi l'arco (u, v).

FARE IL PUNTO 2

```
Alg(G=(V,E), s, (u,v))
    S={s}, d[s]=0, parent[s]=Nil
    ∀z ∈ V-S
        d'[z]=∞,
        parent[z]=Nil
    while S≠V
        tra tutti i nodi w∈V-S, per qualche y∈S
        seleziona tra i nodi w, il nodo x tale che d'[x]= min_{e=(y,x)} (d[y]+ℓ(e)) è il più piccolo possibile
        d[x]=d'[x]
        S ← x
        parent[x]=y

    if (parent[v]==u)
        return True
    else
        return False
```

- La correttezza dell'algoritmo prima di tutto è data dalla correttezza dell'algoritmo di Dijkstra nel calcolo dei cammini minimi.
- Ossia che il cammino da s che passa per v e arriva a x è di costo minimo.

  Vale innanzitutto che $d'[y] \geq d'[x]$, altrimenti avremo scelto $d'[y]$.

  Se esistesse un altro cammino da s in x di costo min., allora vuol dire che questo passa per $d'[y]$ e poi va in x, ma $d'[y] \geq d'[x] \Rightarrow d'[y] + \ell((y,x)) \geq d'[x]$ e quindi tale cammino non è di costo minimo.

- Nell'algoritmo viene usato un vettore in cui si memorizza il parent di ogni nodo, ossia $y = parent[x]$ indica che per andare da s in x, l'ultimo arco attraversato è (y,x).
- Nello specifico del nostro problema, se parent[v]==u, allora per andare da s in v si attraversa l'arco (u,v) e quindi si ritorna True, False altrimenti.

- Il vettore parent non aggiunge nessun overhead assintotico, se non di spazio $\Theta(|V|)$, quindi la complessità dell'algoritmo è data dalla complessità di Dijkstra, ossia $\Theta(|E| \log |V|)$

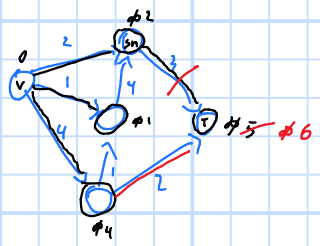
4. *Esercizio*: Dato un grafo orientato $G = (V, E)$, con lunghezze $\ell(e) > 0$ per per ogni arco $e \in E$, e due nodi $v, t \in V$.

(a) Si proponga un algoritmo che determini, oltre ad un cammino minimo p_1 da v a t che supponiamo essere $p_1 = (v, s_1, ...s_n, t)$, un secondo cammino p_2 che sia il cammino di lunghezza minima tra quelli che congiungono v con t e non contengono l'arco (s_n, t).

(b) Si provi la correttezza e si valuti la complessità dell'algoritmo proposto.

(c) Si proponga un algoritmo che determini, oltre ad un cammino minimo p_1 da v a t, un secondo cammino p_2 che sia il cammino di lunghezza minima tra tutti i cammini che congiungono v con t e sono distinti da p_1.

a)



Alg($G=(V,E)$, v, t)

$S = \{v\}$, $d[v]=0$, p_1-parent$[v]$= Nil

$\forall u \in V-S$

$d'[u] = \infty$

p_1-parent$[u]$= Nil

while $S \neq V$

tra tutti i nodi $w \in V-S$, per qualche $y \in S$

selezione tra i nodi w, il nodo x tale che $d'[x] = \min_{e=(y,x)} (d[y] + \ell(e))$ sia il più piccolo possibile

$d[x] = d'[x]$

$S \leftarrow x$

p_1-parent$[x] = y$

$E = E - \{(p_1\text{-parent}[t], t)\}$ p_1-parent$[t]$ è s_n

$S = S - \{t\}$

$d[t] = d'[t] = \min_{e=(u,t)} (d[x] + \ell(e))$, $u \in S$ seleziono il nodo $u : (u,t) \in E$ con costo $d[u] + \ell((u,t))$ minimo

p_2-parent $= p_1$-parent

p_2-parent$[t] = u$

b)

- La correttezza dell'algoritmo nel calcolo di p_1, ossia p_1-parent, è data dalla correttezza dell'algoritmo di Dijkstra.

  Spiegare correttezza di Dijkstra

- p_2 anch'esso viene calcolato correttamente in quanto quello che si fa è eliminare l'arco (s_n, t), ossia $(p_1\text{-parent}[t], t)$, e si trova un nuovo arco $u \in S : (u,t) \in E$ di costo minimo, ossia $d[u] + \ell((u,t))$ più piccolo possibile.

  Non è necessario rifare tutto Dijkstra, in quanto gli altri cammini non vengono influenzati poiché si elimina l'arco che va da (s_n, t).

- La complessità nel calcolo di p_1 è la complessità di Dijkstra, ossia $\Theta(|E| \log |V|)$.

  p_2 viene calcolato in tempo $\Theta(M)$ nel peggiore dei casi dato dalla copia di p_1 in p_2. Il calcolo di $d[t]$ in p_2 viene fatto in tempo $\Theta(\log n)$ in quanto si fa una Extract-min e Decrease-key.

- La complessità totale dell'algoritmo è data dal calcolo di p_1, ossia $\Theta(|E| \log |V|)$.

c)

p_1-parent = Dijkstra($G=(V,E)$, v, t)

$u = t$

while ($u \neq$ Nil)

$E = E - \{(p_1\text{-parent}[u], u)\}$ — rimuovo gli archi del cammino minimo p_1

$u = p_1$-parent$[u]$

p_2-parent = Dijkstra($G=(V,E)$, v, t) — calcolo p_2 su $G=(V,E)$, dove da E sono stati rimossi gli archi di p_1

- L'algoritmo calcola p_1, ossia p_1-parent.

  p_2 deve essere diverso da p_1, perciò ogni arco in p_1 viene tolto da E e dopo viene calcolato dall'inizio p_2.

- La complessità è $\Theta(|E| \log |V|)$ in quanto si fanno due chiamate a Dijkstra.

5. *Esercizio*: Sia $G = (V, E)$ un grafo diretto in cui ad ogni nodo $u \in V$ è associato un peso $w(u) > 0$. Descrivere un algoritmo per calcolare i cammini di peso minimi da un nodo sorgente $s \in V$ a tutti i nodi $v \in V$. In questa situazione, il peso di un cammino è definito come la somma dei pesi dei nodi che appaiono nel cammino. (Sugg.: si riduca il problema a quello di calcolare i cammini di peso minimi in un opportuno grafo con pesi su archi).

```
Alg( G=(V,E), s, w)
  Q ← V
  ∀u ∈ Q
    d[u] = ∞
    parent[u] = Nil
  while Q ≠ ∅
    u = Extract-Min(Q)
    per ogni nodo x adiacente a u
      if x ∈ Q and w((u,x)) < d[u]
        parent[x] = u
        d[x] = w((u,x))
        Decrease-Key(Q, x, d[x])
```

serve per evitare di formare cicli nel MST perché $u \in S$, $x \in Q = V - S$

- L'algoritmo è l'algoritmo di Prim per il calcolo dei cammini di peso totale minimo a partire da un nodo s di partenza

- $d[v] = \min_{e(u,v)} (w(e)),\ u \in Q \Rightarrow u \in S,\ v \in Q \Rightarrow v \in V - S$

- Q è una coda a priorità min sui valori $d[v]$

Prim è simile a Dijkstra, ma differisce nel calcolo di $d[v]$. Mentre in Dijkstra $d[v] = d[u] + \ell((u,v))$, in Prim $d[v] = \ell((u,v))$.

- La complessità è quella dell'algoritmo di Prim, ossia $\Theta(|E| \log |V|)$

6. *Esercizio*: Descrivere l'algoritmo di Prim per la costruzione di un MST e provarne con precisione la correttezza.

```
Prim( G=(V,E), c, r)
    Q ← V
    ∀u ∈ Q
        d[u] = ∞
        parent[u] = Nil
    d[r]=0
    while Q ≠ ∅
        u = Extract-Min(Q)
        per ogni nodo x adiacente a u
            if x ∈ Q and c(u,x) < d[x]
                d[x] = c(u,x)
                parent[x] = u
                Decrease-key(Q, x, d[x])
```

- L'algoritmo di Prim è simile a quello di Dijkstra, ma differisce nel calcolo di $d'[v]$.

  Mentre in Dijkstra $d'[v] = \min_{\substack{e=(u,v):\\ u\in S,\\ v\in V-S}} (d[u] + \ell(e))$, in Prim si ha $d[v] = \min_{\substack{e=(u,v):\\ u\in S\\ v\in V-S}} (c(e))$.

  Prim a ogni istante prende l'arco $(x,y): x\in S, y\in V-S$ di costo minimo, aggiunge y in S, nell'algoritmo è l'equivalente di rimuovere y da Q, $u\in Q \to u\in V-S$.

  Quando $Q=\emptyset$, ossia $V-S=\emptyset \Rightarrow S=V$, allora l'algoritmo termina e in parent[·] vi sarà il MST prodotto da Prim.

- Essendo Prim simile a Dijkstra, la complessità è la stessa, ossia $\Theta(|E| \log |V|)$.

- La correttezza di Prim è data dalla dimostrazione che esso produca un MST.
  - Prim produce un albero in quanto aggiunge archi da S in $V-S$ e quindi non vi possono essere cicli
  - Prim aggiunge a ogni iterazione sempre l'arco di costo minore da S in $V-S$.

  Per i due punti di sopra, Prim produce un albero di costo <u>totale</u> minimo, ossia un MST.

7. *Esercizio*: Descrivere l'algoritmo di Kruskal per la costruzione di un MST e provarne con precisione la correttezza.

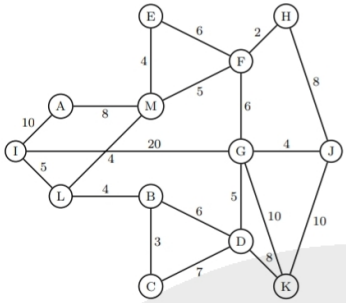
- Kruskal procede selezionando a ogni passo l'arco $(u,v) \in E : (u,v) = \min\{c(x,y) \in E\}$ e che non formi un ciclo.

  Inoltre, così come per Prim e Dijkstra, deve valere che $u \in S$ e $v \in V-S$, questo serve per evitare la formazione di cicli, in quanto Kruskal deve produrre un MST, il quale è un albero e quindi non può contenere cicli.

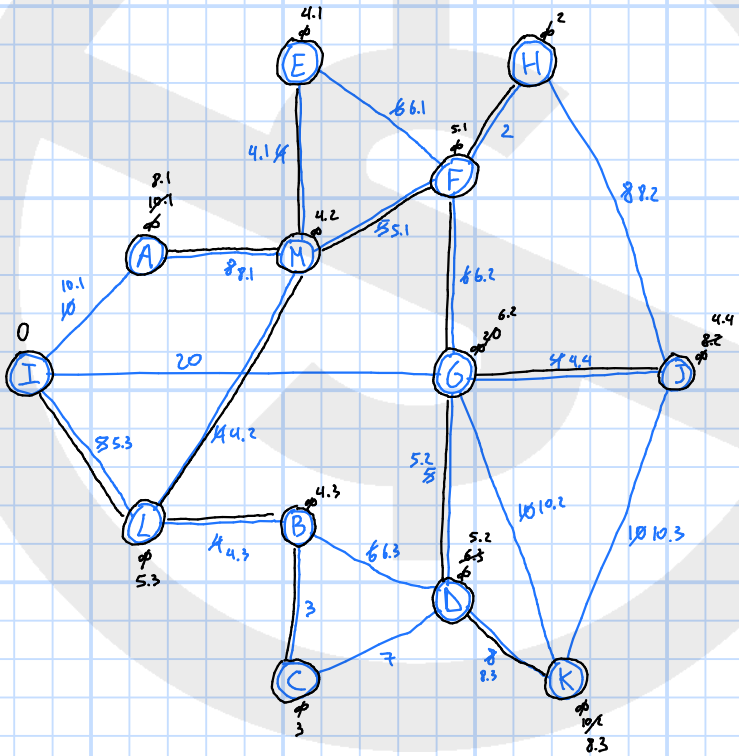
- Essendo che non devono esserci cicli, Kruskal selezionerà $|V|-1$ archi e questi ci garantiranno che gli archi selezionati siano un grafo connesso poiché non vi sono cicli e vi sono esattamente $|V|-1$ archi.

- Per quanto Kruskal produce un MST, ossia un albero con costo totale minimo.

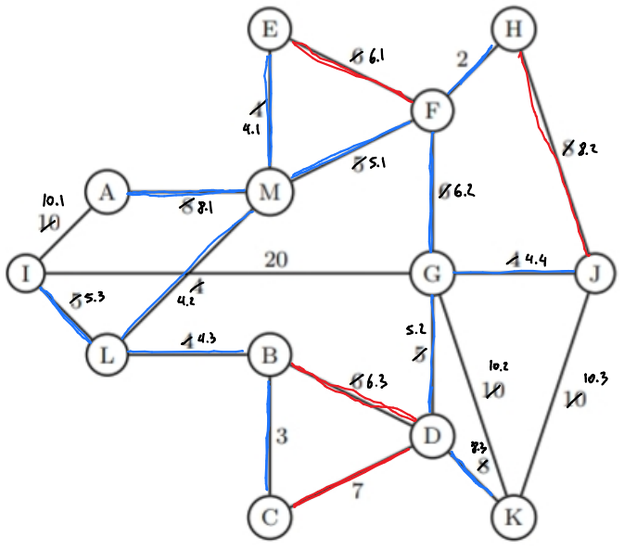
8. *Esercizio*: Dato il grafo rappresentato in figura, calcolare un MST di esso applicando l'algoritmo di Prim a partire dal nodo I. Si descrivano con precisione le computazioni effettuate dall'algoritmo passo dopo passo.



RICORDARSI DI CAMBIARE LEGGERMENTE I VALORI DEGLI ARCHI CHE HANNO COSTO UGUALE



9. *Esercizio*: Dato il grafo rappresentato in figura, calcolare un MST di esso applicando l'algoritmo di Kruskal. Si descrivano con precisione le computazioni effettuate dall'algoritmo passo dopo passo.

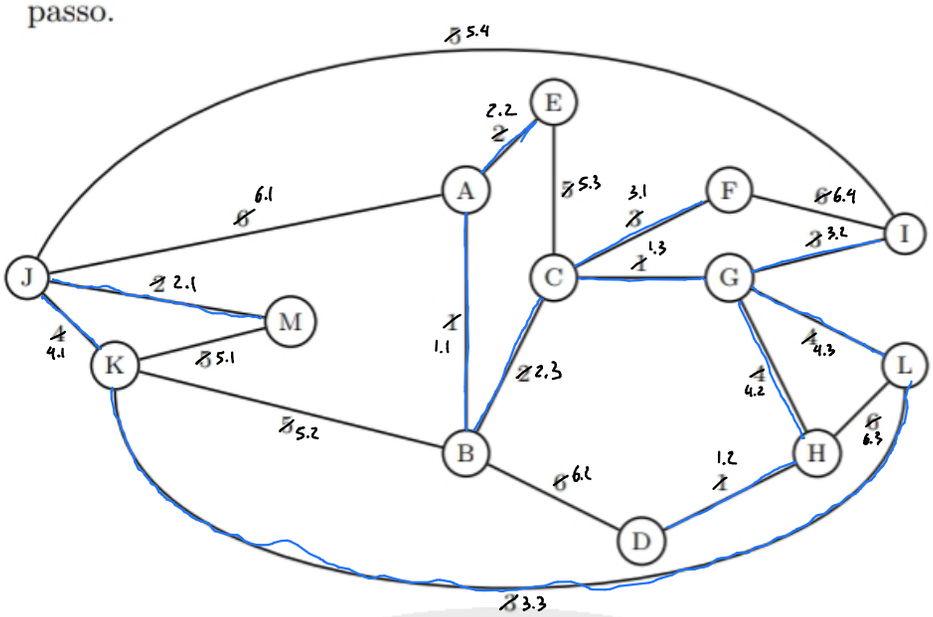


RICORDARSI DI CAMBIARE LEGGERMENTE I VALORI DEGLI ARCHI CHE HANNO COSTO UGUALE

archi selezionati ↓	arco considerato ↓	arco scelto? (formerebbe un ciclo) ↓
T	(u, v)	SI \ NO
{ }		
{(f,h)}	(f, h)	NO
{(f,h), (b,c)}	(b, c)	NO
{(f,h), (b,c), (e,m)}	(e, m)	NO
T ∪ {(l,m)}	(l, m)	NO
T ∪ {(l,b)}	(l, b)	NO
T ∪ {(g,j)}	(g, j)	NO
T ∪ {(m,f)}	(m, f)	NO
T ∪ {(g,d)}	(g, d)	NO
T ∪ {(i,l)}	(i, l)	NO
T	(e, f)	SI
T ∪ {(f,g)}	(f, g)	NO
T	(b, d)	SI
T	(c, d)	SI
T ∪ {(z,m)}	(z, m)	NO
T	(h, g)	SI
T ∪ {(d,k)}	(d, k)	NO

Non c'è bisogno di controllare gli archi rimanenti perchè $|T| = |V| - 1$, qualsiasi arco in $E - T$ aggiunto a T formerebbe un ciclo e per questo l'algoritmo può fermarsi quando $|T| = |V| - 1$.

10. *Esercizio*: Dato il grafo rappresentato in figura, calcolare un MST di esso applicando l'algoritmo di Kruskal. Si descrivano con precisione le computazioni effettuate dall'algoritmo passo dopo passo.

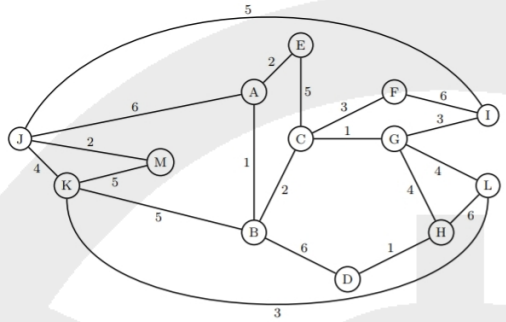


RICORDARSI DI CAMBIARE LEGGERMENTE I VALORI DEGLI ARCHI CHE HANNO COSTO UGUALE

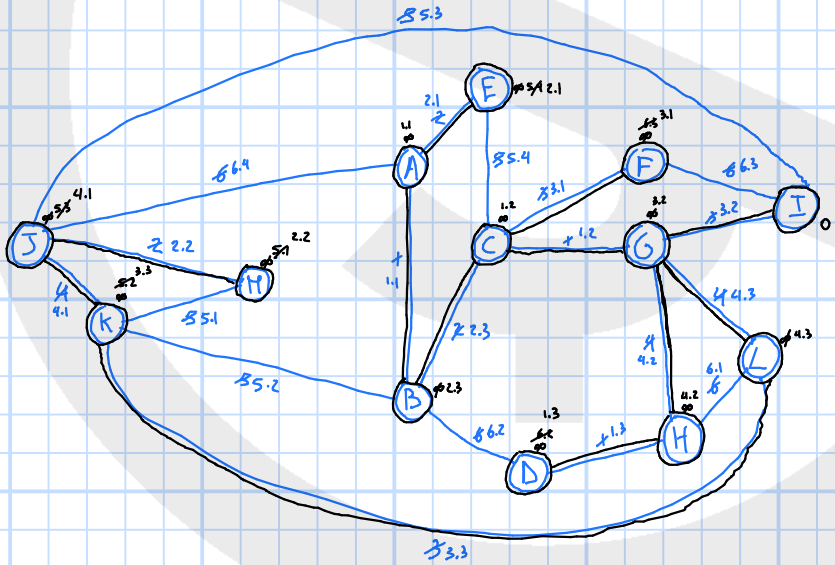
T	arco considerato	scartato? (formerebbe un ciclo)
$\{\ \}$		
$\{(a,b)\}$	(a,b)	NO
$T \cup \{(d,h)\}$	(d,h)	NO
$T \cup \{(c,g)\}$	(c,g)	NO
$T \cup \{(j,m)\}$	(j,m)	NO
$T \cup \{(a,e)\}$	(a,e)	NO
$T \cup \{(b,c)\}$	(b,c)	NO
$T \cup \{(c,f)\}$	(c,f)	NO
$T \cup \{(g,i)\}$	(g,i)	NO
$T \cup \{(k,l)\}$	(k,l)	NO
$T \cup \{(j,k)\}$	(j,k)	NO
$T \cup \{(g,h)\}$	(g,h)	NO
$T \cup \{(g,l)\}$	(g,l)	NO

Non c'è bisogno di controllare gli archi rimanenti perché $|T| = |V|-1$, qualsiasi arco in $E-T$ aggiunto a T formerebbe un ciclo e per questo l'algoritmo può fermarsi quando $|T| = |V|-1$.

11. *Esercizio*: Dato il grafo rappresentato in figura, calcolare un MST di esso applicando l'algoritmo di Prim a partire dal vertice I. Si descrivano con precisione le computazioni effettuate dall'algoritmo passo dopo passo.



RICORDARSI DI CAMBIARE LEGGERMENTE I VALORI DEGLI ARCHI CHE HANNO COSTO UGUALE



12. *Esercizio*: Si proponga un algoritmo che, dato in input un grafo non non orientato $G = (V, E)$, pesi $w(e)$ per ogni arco $e \in E$, ed un arco $(u, v) \in E$, determini l'esistenza o meno di un MST per G non contenente l'arco (u, v). Si motivi la correttezza e si calcoli la complessit'a dell' algoritmo proposto. Si riconsiderino i punti precedenti nel caso in cui si voglia determinare l'esistenza di un MST per il grafo G contenente l'arco (u, v).

```
Alg( G=(V,E), w)
  Q ← V
  ∀u ∈ Q
      d[u] = ∞
  E = E - {(u,v)}
  d[v] = 0
  while Q ≠ ∅
      x = Extract-Min(Q)
      per ogni nodo y adiacente a x
          if x ∈ Q and w(x,y) < d[x]
              d[x] = w(x,y)
              Decrease-Key(Q, x, d[x])

  ∀v ∈ V
      if d[v] == ∞
          return "Non esiste un MST per G in quanto G non connesso"
  return "Esiste MST per G senza arco (u,v)"
```

Esiste un MST passante per $(u,v) \Leftrightarrow w(u,v) <$ di altri $n-1$ archi e non forma un ciclo

Complessità data dall'algoritmo di Prim, ossia $\Theta(|E| \log |V|)$

13. *Esercizio*: Sia $G = (V, E)$ un grafo non orientato e pesato, con archi di costo $c(e)$, per ogni $e \in E$.

(a) Si descriva brevemente un algoritmo efficiente (per esempio tra quelli visti a lezione) che determini un Minimum Spanning Tree per G e se ne valuti la complessità.

(b) Si provi o si refuti la seguente proprietà: dato un Minimum Spanning Tree per G questo contiene almeno un arco tra quelli aventi peso minimo in G.

(c) Si provi o si refuti la seguente proprietà: dato un Minimum Spanning Tree per G questo contiene un cammino minimo per ogni coppia di nodi in G.

a) Prim $\Theta(|E| \log |V|)$

Spiegazione di Prim

b) Se per assurdo MST non contenesse un arco tra quelli di costo minimo, allora esisterebbe un altro MST con costo inferiore usando qualche di codesti archi.

Se per ipotesi la scelta dei primi n-1 archi di costo minore formasse un ciclo, allora basterebbe scartarne alcuni, ma sicuramente ci sarà almeno uno degli archi nel MST poiché un solo arco non forma un ciclo.

c) Questo avviene per costruzione nei diversi algoritmi. In generale ad ogni passo si sceglie un arco che tra un sottoinsieme di archi ha costo minore e non forma un ciclo. Quindi si sceglieranno $|V|-1$ archi e questi formeranno il cammino minimo per ogni coppia di nodi

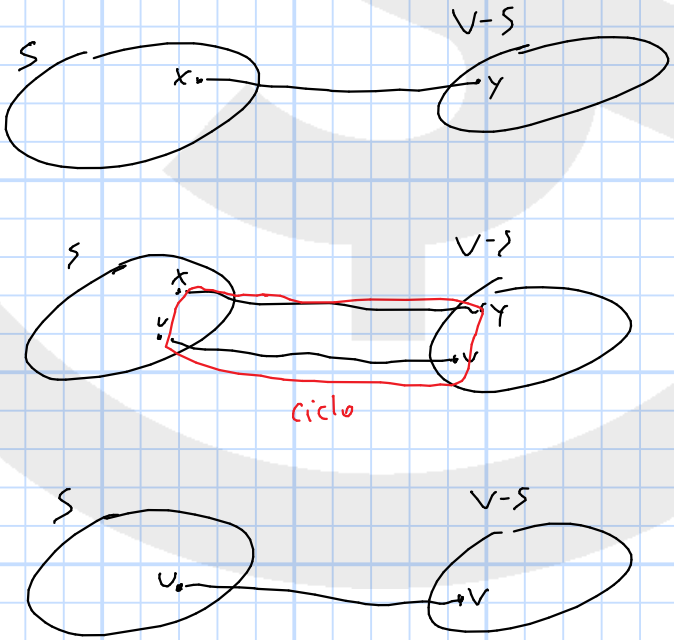
14. *Esercizio*: Sia $G = (V, E)$ un grafo non orientato e pesato, con archi di costo $c(e)$, per ogni $e \in E$. Si provi la seguente affermazione:

sia $\emptyset \neq S \subset V$ un sottoinsieme dei nodi, e sia $e = (u, v)$ l'arco di costo minimo *con un estremo in S e l'altro in $V - S$. Allora* ogni *MST contiene l'arco e.*

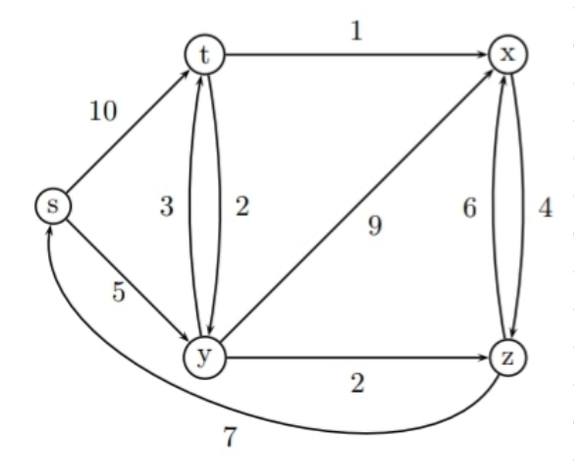
Se per assurdo si scegliesse un arco (x,y), $x \in S$, $y \in V-S$ in cui $c(x,y) > c(u,v)$, prima o poi sceglieremo l'arco (u,v), in quanto minimizza la nostra soluzione.

Ma se scegliamo (u,v), si forma un ciclo in quanto si può andare da S in $V-S$ con (x,y) e lo si può fare anche con (u,v).

Bisogna eliminare il ciclo e quindi, poiché minimizziamo, eliminiamo l'arco con costo maggiore, ossia (x,y) e quindi $(u,v) \in MST$



15. *Esercizio*: Si esegua l'algoritmo di Bellman-Ford per il calcolo del cammino minimo da s a t nel grafo di sotto riportato. Si descrivano con precisione le computazioni effettuate dall'algoritmo passo dopo passo.



$$OPT(i,u) = \min\left(OPT(i-1,u),\ \min_{w \in V}\left(c(u,w) + OPT(i-1,u)\right)\right)$$

$n-1=4$ ci si ferma a $OPT(4,u)$ poichè con $s=n=|V|$ si formano cicli

$\forall u \in S,\ OPT(0,u) = \infty$

$OPT(0 \dots n-1, t) = 0$

$OPT(1,s) = \min(\infty, \min(10+0, 5+\infty)) = 10$

$OPT(1,y) = \min(\infty, \min(3+0, 9+\infty, 2+\infty)) = 3$

$OPT(1,z) = \min(\infty, \min(6+\infty, 7+\infty)) = \infty$

$OPT(1,x) = \min(\infty, \min(4+\infty)) = \infty$

$OPT(2,s) = \min(10, \min(5+3) = 8$

$OPT(2,y) = \min(3, \min(9+\infty, 2+\infty)) = 3$

$OPT(2,z) = \min(\infty, \min(6+\infty, 7+10)) = 17$

$OPT(2,x) = \min(\infty, \min(4+\infty)) = \infty$

$OPT(3,s) = \min(8, \min(5+3)) = 8$

$OPT(3,y) = \min(3, \min(9+\infty, 2+17)) = 3$

$OPT(3,z) = \min(17, \min(4+\infty, 7+8)) = 15$

$OPT(3,x) = \min(\infty, \min(4+17)) = 21$

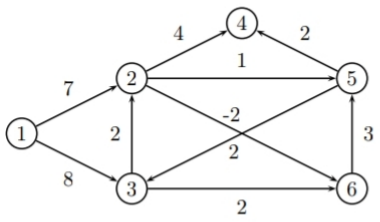
$OPT(4,s) = \min(8, \min(5+3)) = 8$

$OPT(4,y) = \min(3, \min(9+17, 2+15)) = 3$

$OPT(4,z) = \min(15, \min(4+17, 7+8)) = 15$

$OPT(4,x) = \min(17, \min(4+15)) = 19$

16. *Esercizio*: Si esegua l'algoritmo di Bellman-Ford per il calcolo dei cammini minimi da 1 al nodo 6. Si descrivano con precisione le computazioni effettuate dall'algoritmo passo dopo passo.



(L'arco da 5 a 3 ha peso -2 e l'arco da 2 a 6 ha peso 2.)

$$OPT(i,v) = \min\left(OPT(i-1,v),\ \min_{w \in V}\left(c(v,w) + OPT(i-1,w)\right)\right)$$

$n-1 = 5$

$\forall v \in V,\ OPT(0,v) = \infty$

$OPT(0 \dots n-1, 6) = 0$

$OPT(1,1) = \min(\infty, \min(7+\infty, 8+\infty)) = \infty$

$OPT(1,2) = \min(\infty, \min(4+\infty, 1+\infty, 2+0)) = 2$

$OPT(1,3) = \min(\infty, \min(2+\infty, 2+0,)) = 2$

$OPT(1,4) = \infty$

$OPT(1,5) = \min(\infty, \min(-2+\infty, 2+\infty)) = \infty$

$OPT(2,1) = \min(\infty, \min(7+2, 8+2)) = 9$

$OPT(2,2) = \min(2, \min(4+\infty, 1+\infty)) = 2$

$OPT(2,3) = \min(2, \min(2+2)) = 2$

$OPT(2,4) = \infty$

$OPT(2,5) = \min(\infty, \min(-2+2, 2+\infty)) = 0$

$OPT(3,1) = \min(9, \min(7+2, 8+2)) = 9$

$OPT(3,2) = \min(2, \min(4+\infty, 1+0)) = 1$

$OPT(3,3) = \min(2, \min(2+2)) = 2$

$OPT(3,4) = \infty$

$OPT(3,5) = \min(0, \min(-2+2, 2+\infty)) = 0$

$OPT(4,1) = \min(9, \min(7+1, 8+2)) = 8$

$OPT(4,2) = \min(1, \min(4+\infty, 1+0)) = 1$

$OPT(4,3) = \min(2, \min(2+1)) = 2$

$OPT(4,4) = \infty$

$OPT(4,5) = \min(0, \min(-2+2, 2+\infty)) = 0$

$OPT(5,1) = \min(8, \min(7+1, 8+2)) = 8$

$OPT(5,2) = \min(1, \min(4+\infty+1+0)) = 1$

$OPT(5,3) = \min(2, \min(2+1)) = 2$

$OPT(5,4) = \infty$

$OPT(5,5) = \min(0, \min(-2+2, 2+\infty)) = 0$

17. *Esercizio*: Derivare l'equazione di ricorrenza per il costo di cammini minimi in grafi, argomentando i relativi passi. Descrivere un algoritmo per il calcolo dell'equazione prima ottenuta.

$$
\begin{array}{l}
\text{Archi nodo} \\
\downarrow \quad \downarrow \\
OPT(i,v) = \\
\uparrow \\
\text{costo}
\end{array}
\begin{cases}
0 & \text{se } v = t \\
\min\left(OPT(i-1,v),\ \min\limits_{w \in V}\left(c(v,w) + OPT(i-1,w)\right)\right) & \text{altrimenti} \\
\infty & \text{se } i = 0
\end{cases}
$$

- $v = t = 0$ poiché per andare da t a t costa 0
- ∞ se $i=0$ poiché senza archi non si può andare da v a t
- Quello che si cerca di fare è di vedere se esiste un cammino con più archi che costa di meno. Questo può succedere poiché gli archi possono avere costi negativi.

Algoritmo di Bellman e Ford

```
Alg( G=(V,E), s, t, c)
  M[i,v] = ∞,      0 ≤ i ≤ n-1
  M[i,t] = 0
  for i=1 to n-1
     ∀v ∈ V
        per ogni w adiacente a v
           M[i,v] = M[i-1,v]
           if c(v,w)+M[i-1,w] < M[i,v]
              M[i,v] = c(v,w)+M[i-1,w]

  return M[n-1,t]
```

$$OPT(i,v) = \min\left(OPT(i-1,v),\ \min_{\substack{w \in V: \\ (v,w) \in E}}\left(c(v,w) + OPT(i-1,w)\right)\right)$$

La matrice $M[\cdot,\cdot]$ avrà $n \cdot m = |V| \cdot |E|$ entry e quindi la complessità è $\Theta(|V||E|)$.

18. *Esercizio*: Derivare l'algoritmo per scoprire se un grafo diretto con lunghezze sugli archi contiene cicli di costo totale negativo e provarne la correttezza.

1. Eseguo l'algoritmo di Bellman e Ford per calcolare la matrice $M[i,v]$, $0 \le i \le n-1$, $v \in V$

2. $\forall v \in V$ calcolo $M[n,v]$ e confronto il valore con $M[n-1,v]$.

   Se $M[n,v] < M[n-1,v]$ allora esiste un ciclo negativo. n archi sicuramente formano un ciclo.

Se G non avesse cicli negativi, allora l'aggiunta di un arco farebbe aumentare o lascierebbe invariata la soluzione.

19. *Esercizio*: Derivare l'algoritmo per trovare in un grafo diretto un ciclo di costo totale negativo (se esso esiste) e provarne la correttezza.

1. Eseguo l'algoritmo di Bellman e Ford per calcolare la matrice $M[i,v]$, $0 \leq i \leq n-1$, $v \in V$

2. $\forall v \in V$, calcolo $M[n,v]$ e lo confronto con $M[n-1,v]$, se $M[n,v] < M[n-1,v]$, allora esiste un ciclo con costo negativo.

3. Sia w : $M[n,w] < M[n-1,w]$, ossia il cammino da w in t con n archi ha un ciclo negativo.

4. Se il cammino da w a t non avesse un ciclo di costo negativo, allora avremmo potuto eliminare l'arco che crea il ciclo e avere un cammino di $n-1$ archi $< M[n,w]$, ma ciò è assurdo poiché $M[n,w] < M[n-1,w]$.

ESERCIZI 7

1. *Esercizio*: Scrivere in pseudo codice una procedura che, preso in input un intero n, stampi tutte le stringhe di lunghezza minore o uguale ad n sull'alfabeto $\{a, b, c\}$.

```
Alg(sol, k, n)
  if k == n
    print(sol)
  else
    print(sol[:k+1])   stampa i primi k+1 elementi in sol
    for x in {a,b,c}
      sol[k+1] = x
      Alg(sol, k+1, n)
```

L'algoritmo verrà chiamato su Alg(sol, 0, n)

- L'algoritmo procede alla costruzione di tutte le stringhe $w \in \{a,b,c\}^* : |w| \leq n$ e le stampa.

  Una volta che $|w| = n$, viene effettuato il backtracking per la stampa delle altre stringhe.

- La complessità è data dalla stampa di ogni sol che richiede $O(n)$ e dalla generazione delle $|w| \leq n$.

  La complessità della generazione è uguale a $|S| = \{w \in \{a,b,c\}^* : |w| \leq n\}$

  $|S|$ può essere indicato anche come $D(n)$, dove $D(n)$ è il numero di elementi da stampare, nell'esercizio successivo viene utilizzato

  Quindi la complessità totale dell'algoritmo è $O(n|S|)$.

2. *Esercizio*: Scrivere in pseudo codice una procedura che, preso in input interi n e c, stampi tutti i sottinsiemi di $\{1, 2, \ldots, n\}$ di cardinalità $\leq c$. La complessità dell'algoritmo deve essere $O(nD(n))$, dove $D(n)$ è il numero di elementi da stampare.

```
Alg(SOL, k, c, n, tot)
    if k==n
        print(SOL)
    else
        SOL[k+1]=0
        Alg(SOL, k+1, c, n, tot)
        if tot<c
            SOL[k+1]=1
            Alg(SOL, k+1, c, n, tot+1)
```

non $\leq c$ perchè altrimenti aggiungiamo elementi e se $tot \leq c \Rightarrow tot = c+1$ nella ricorsione successiva e questo non va bene

- Usiamo una funzione di taglio per evitare di calcolare i sottoinsiemi di cardinalità $> c$.

- La complessità è data dalla stampa di <u>ogni</u> SOL che richiede tempo $O(n)$.

· Il numero delle soluzioni è $\sum_{j=0}^{c} \binom{n}{j} = D(n)$.

Quindi la complessità totale dell'algoritmo è $O(nD(n))$.

3. *Esercizio*:

Si consideri la seguente istanza del problema dello Zaino: Vettore dei valori $v = [12, 26, 8, 6]$, vettore dei pesi $p = [6, 26, 10, 10]$, capacità di trasporto $P = 26$.

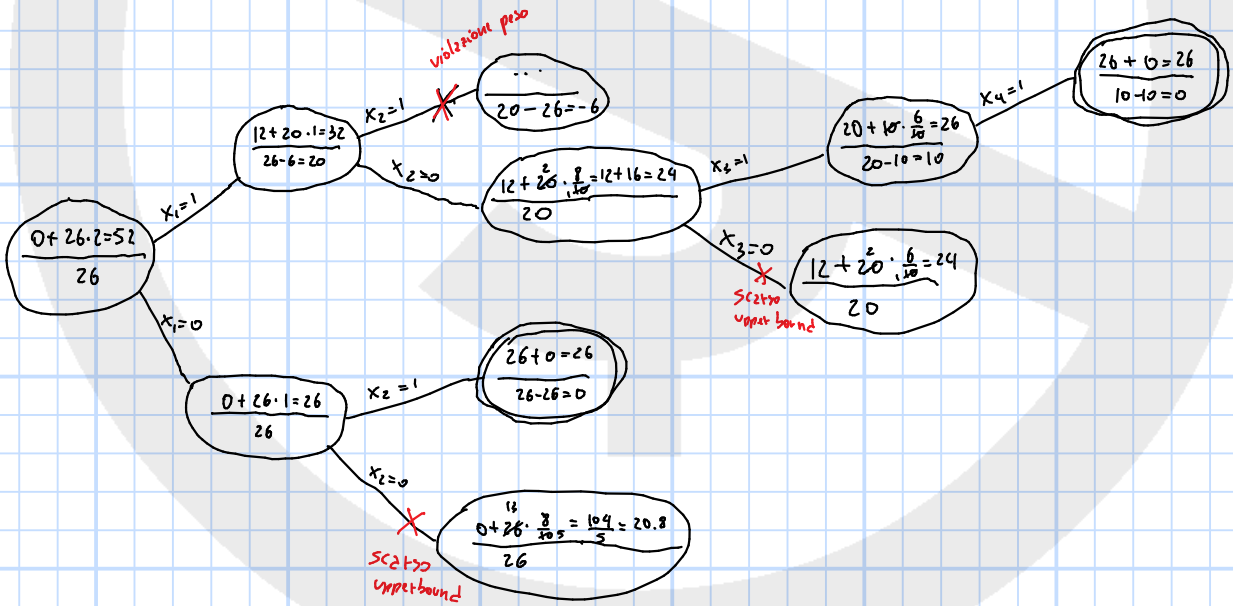
Si disegni l'albero delle soluzioni dell'algoritmo di Branch e Bound, usando la stessa funzione di taglio vista a lezione.

$\frac{v_i}{p_i} = [2, 1, \frac{8}{10} = 0.8, \frac{6}{10} = 0.6]$ per ipotesi v è ordinato t.c. $\frac{v_i}{p_i} \geq \frac{v_{i+1}}{p_{i+1}}$

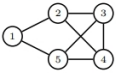
$$\text{Upper bound} = \underbrace{\sum_{i=1}^{k} x_i v_i}_{\text{valore oggetti già considerati}} + \underbrace{\left(P - \sum_{i=1}^{k} x_i p_i\right)}_{\text{capacità rimanente}} \cdot \underbrace{\frac{v_{k+1}}{p_{k+1}}}_{\text{oggetto con maggiore valore non ancora considerato}}$$

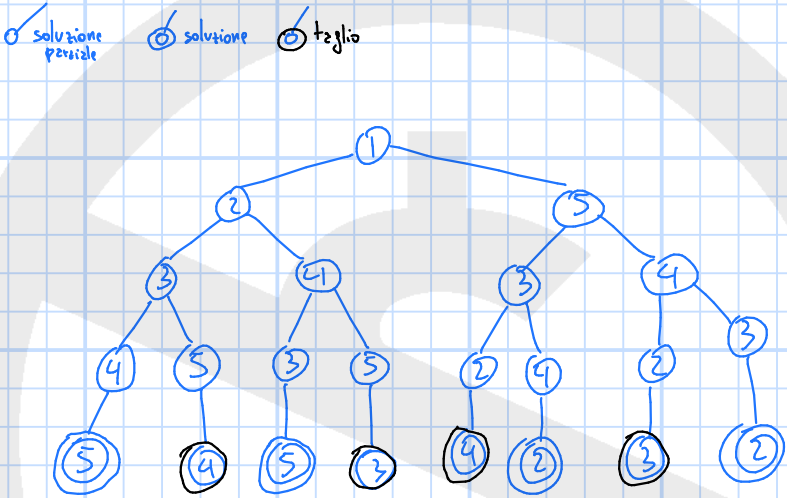
questo vale 0 all'ultimo calcolo di x_n, poiché v_{n+1} non esiste

$\frac{\text{upper bound}}{\text{capacità rimanente}}$



4. *Esercizio*: Dato il grafo di sotto riportato, si disegni l'albero delle soluzioni dell'algoritmo `CICLI-HAMILTONIANI`, spiegando le operazioni effettuate.



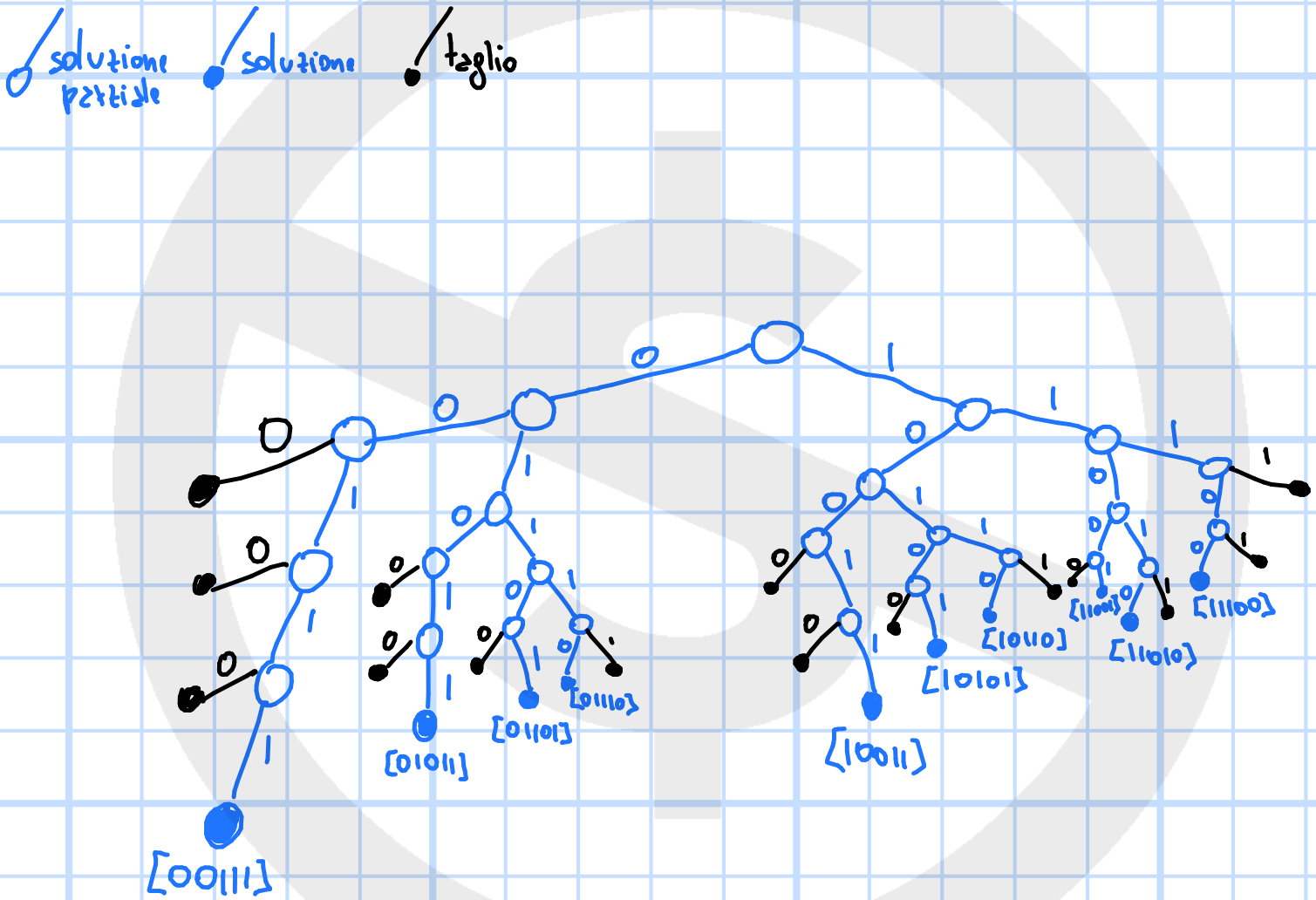


L'operazione fatta è quella di seguire gli archi a partire da un nodo.

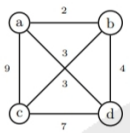
Se si riesce a visitare tutti i nodi e l'ultimo nodo ha un arco al nodo di partenza, allora è un ciclo Hamiltoniano.

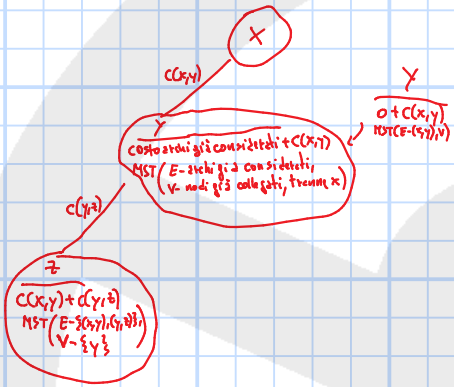
La funzione di taglio è quella che si seguono solo i nodi u e v tale che $(u,v) \in E$.

5. *Esercizio*: Si disegni l'albero delle soluzioni dell'algoritmo SOTTOINSIEMI-VINCOLATI($SOL, 0, c, n$), per $c = 3$ e $n = 5$

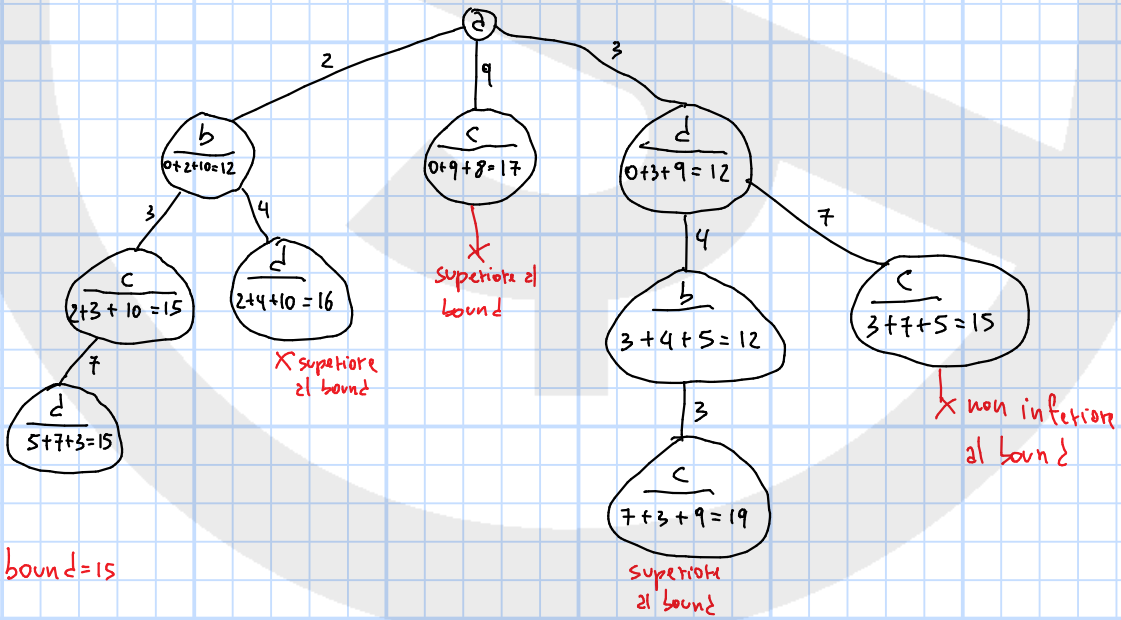


6. *Esercizio*: Si disegni l'albero delle soluzioni per il problema del Commesso Viaggiatore sul seguente grafo:





scelgo un nodo e un arco e poi continuo in ampiezza, scegliendo quello con Bound minore. N.B: ciò non significa che sia la scelta giusta. È solo un criterio di scelta



bound = 15