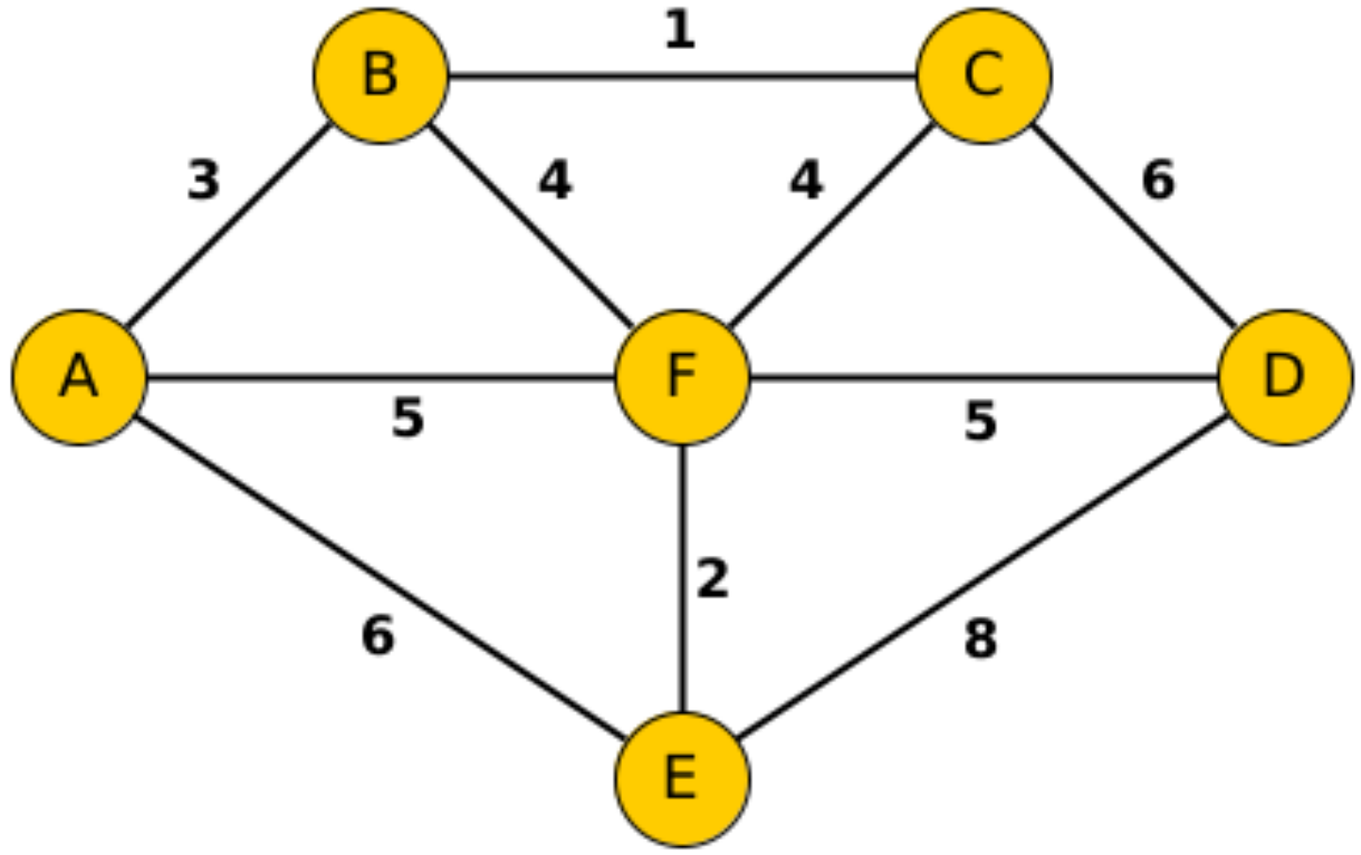
ESERCIZIO 1.

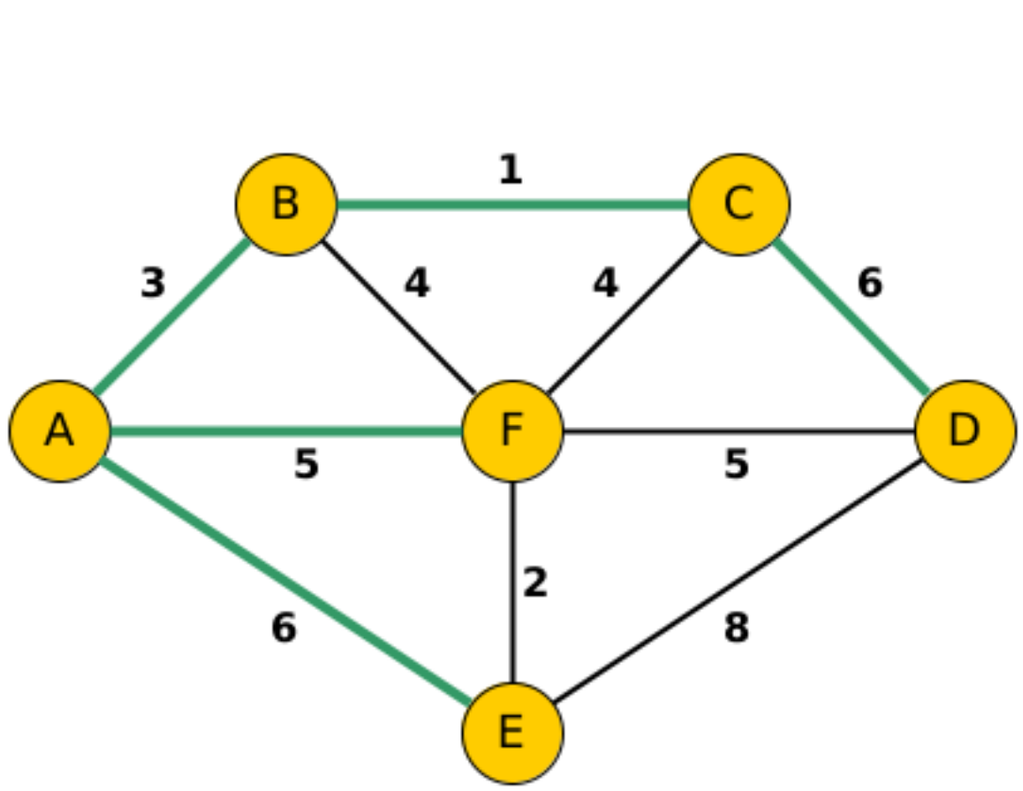
Applicate l'algoritmo di Dijkstra e determinate la distanza minima di ogni vertice del grafo qu sotto, a partite dal vertice etichettato A. Devono essere rappresentati: il vettore dei genitori, che codifica l'albero dei cammini minimi, il vettore delle distanze minime, e un disegno dell'albero dei cammini minimi.

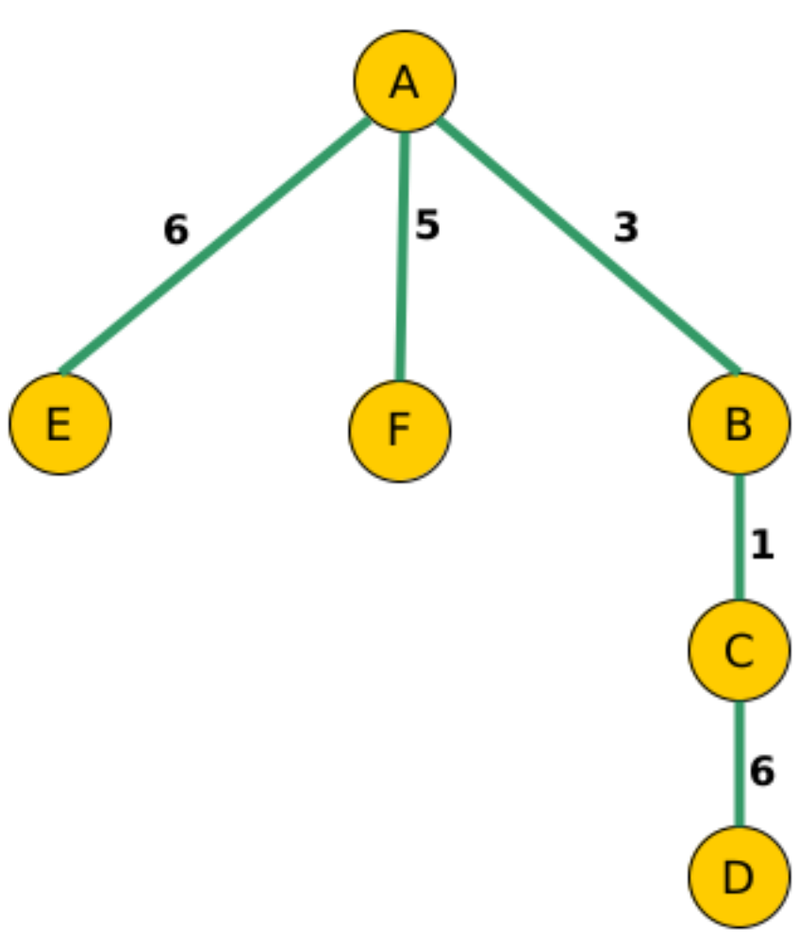
Nel caso in cui l'algoritmo si trovi a scegliere tra due vertici che hanno la stessa priorità, sceglierà prima quello con l'etichetta che viene prima in ordine alfabetico.



Includiamo la descrizione del comportamento dell'algoritmo

vertice	A	B	C	D	E	F
genitore	—	A	B	C	A	A
distanza	0	3	4	10	6	5





ESERCIZIO 2.

Considerate una griglia $n \times n$ con $n > 0$. Un cammino valido in questa griglia deve partire dalla cella di coordinate $(0,0)$ in alto a sinistra ed arrivare alla posizione di coordinate $(n-1,n-1)$ in basso a destra.
E' possibile muoversi solo su celle adiacenti, andando di un passo verso il basso oppure di un passo verso destra. Oltretutto al cammino è vietato toccare le celle di coordinate (i,j) con $i > j$, in altre parole e` vietato toccare celle che si trovino sotto alla diagonale che va da $(0,0)$ a $(n-1,n-1)$.

Ad esempio per $n = 4$ la risposta deve essere 5, abbiamo infatti i seguenti possibili cammini:

$\rightarrow\rightarrow\rightarrow\downarrow\downarrow\downarrow$ $\rightarrow\rightarrow\downarrow\rightarrow\downarrow\downarrow$ $\rightarrow\rightarrow\downarrow\downarrow\rightarrow\downarrow$ $\rightarrow\downarrow\rightarrow\downarrow\rightarrow\downarrow$ $\rightarrow\downarrow\rightarrow\rightarrow\downarrow\downarrow$

Progettare un algoritmo che calcoli il numero di cammini validi e che impieghi tempo $O(n^2)$.

Motivare BENE la correttezza e la complessità dell'algoritmo proposto.

Utilizziamo una tabella di dimensioni $n \times n$.

$T[i][j] =$ **il numero di cammini che dalla cella** $(0,0)$ **arrivano alla cella** (i,j) .
La soluzione al problema sarà $T[n-1][n-1]$.

Resta solo da definire la ricorrenza per il calcolo delle $\Theta(n^2)$ celle della tabella .

Per $i = j = 0$ si ha $T[0][0] = 1$ perché i movimenti non permettono di tornare indietro e si può raggiungere la cella $(0,0)$ solo col cammino "vuoto".

Per $i > j$ si ha $T[i][j] = 0$ perché la cella (i,j) è irraggiungibile

Per $i = j$ e $i > 0$ si ha $T[i][j] = T[i-1][j]$ perché posso raggiungere la cella (i,j) solo con i cammini che provengono dall'alto e fanno un passo verso il basso.

Per $i = 0$ e $j > 0$ si ha $T[i][j] = T[i][j-1]$ perché posso raggiungere la cella (i,j) solo con i cammini che provengono da sinistra e fanno un passo a destra.

In tutti gli altri casi si ha $T[i][j] = T[i-1][j] + T[i-1][j]$ perché posso arrivare da entrambe le direzioni

Implementazione:

```
def Es2(n):
    # tabella
    T = [ [ 0 for _ in range(n)] for _ in range(n) ]
    for i in range(n):
        for j in range(i,n): # ignora j<i
            if   i==j==0: T[i][j] = 1           # partenza
            elif i==0:    T[i][j] = T[i][j-1] # prima riga con j>0
            elif i==j:    T[i][j] = T[i-1][j] # sulla diagonale con i>0
            else:         T[i][j] = T[i-1][j] + T[i][j-1]
    return T[n-1][n-1]
```

```
>>> Es2(4)
5
>>> Es2(5)
14
>>> Es2(6)
42
```

Complessità $\Theta(n^2)$

ESERCIZIO 3.

Fissiamo n e k con $n \geq k \geq 1$. Definiamo **valida** una sequenza di lunghezza n contenente interi da 0 a $k-1$, e che contenga ognuno di questi k valori almeno una volta. Ad esempio se abbiamo $n = 6$ e $k = 4$, allora la sequenza 013212 è valida mentre 020323 non è valida.

Facciamo un altro esempio: tutte le sequenze valide per $n = 4$ e $k = 3$ sono

0012 0021 0102 0112 0120 0121 0122 0201 0210 0211 0212 0221 1002 1012 1020 1021 1022 1102 1120 1200 1201 1202 1210 1220 2001 2010 2011 2012 2021 2100 2101 2102 2110 2120 2201 2210

Trovate un algoritmo che dati n ed k stampi tutte e sole le sequenze valide.
L'algoritmo deve avere complessità $O(n \cdot k \cdot S(n,k))$ dove $S(n,k)$ è il numero di sequenze valide esistenti.

Motivare BENE la correttezza e la complessità dell'algoritmo proposto, calcolando il numero di nodi interni e di foglie nell'albero di computazione, e mettendoli in corrispondenza con $S(n,k)$.

Algoritmo:

Per risolvere questo problema utilizziamo un algoritmo basato sulla costruzione ricorsiva di tutte le sequenze di lunghezza n che contengano tutti i valori da 0 a $k-1$.

Durante la ricorsione la sequenza parziale viene memorizzata in un parametro

`prefix`

mentre viene tenuta traccia degli valori usati nella sequenza

`used`

con una lista di k booleani.

- se il numero di posizioni della sequenza da riempire è maggiore dei valori che non sono stati utilizzati allora si può aggiungere qualunque valore tra 0 e $k-1$.
- altrimenti deve per forza essere aggiunto un elemento tra quelli non utilizzati, tagliando quindi le altre possibilità.

In ogni caso, se alla sequenza viene aggiunto un valore mai usato prima, la funzione deve segnare in `used` che quell'elemento è usato, ma poi deve resettare questa informazione prima di tentare altri rami.

Implementazione

```
def PrintEs3(n,k):
    if n<k:
        return
    Es3rec(n,k,[], [False]*k)

def Es3rec(n,k,prefix,used):
    if len(prefix)==n:
        print(prefix)
        return
    space    = n - len(prefix)
    num_miss = used.count(False)  # costa k passi

    for i in range(k):
        if used[i] and space <= num_miss:
            continue
        prefix.append(i)
        if used[i]:
            Es3rec(n,k,prefix,used)
        else:
            used[i] = True
            Es3rec(n,k,prefix,used)
            used[i] = False
        prefix.pop()
```

```
>>> PrintEs3(4,3)
[0, 0, 1, 2]
[0, 0, 2, 1]
[0, 1, 0, 2]
[0, 1, 1, 2]
[0, 1, 2, 0]
[0, 1, 2, 1]
[0, 1, 2, 2]
[0, 2, 0, 1]
[0, 2, 1, 0]
[0, 2, 1, 1]
[0, 2, 1, 2]
[0, 2, 2, 1]
[1, 0, 0, 2]
[1, 0, 1, 2]
[1, 0, 2, 0]
[1, 0, 2, 1]
[1, 0, 2, 2]
[1, 1, 0, 2]
[1, 1, 2, 0]
[1, 2, 0, 0]
[1, 2, 0, 1]
[1, 2, 0, 2]
[1, 2, 1, 0]
[1, 2, 2, 0]
[2, 0, 0, 1]
[2, 0, 1, 0]
[2, 0, 1, 1]
[2, 0, 1, 2]
[2, 0, 2, 1]
[2, 1, 0, 0]
[2, 1, 0, 1]
[2, 1, 0, 2]
[2, 1, 1, 0]
[2, 1, 2, 0]
[2, 2, 0, 1]
[2, 2, 1, 0]
```

NOTE:

Ognuno dei $S(n,k)$ nodi foglia della ricorsione impiega $O(n)$ per la stampa.

Ogni chiamata che corrisponde ad un nodo interno impiega $O(k)$ per contare quanti valori non sono ancora apparsi nella sequenza. Poi effetta k iterazioni per prendere in considerazione come proseguire la sequenza.

Si può vedere tramite induzione che il numero di posizioni da riempire è sempre maggiore o uguale al numero di valori non apparsi `num_miss`.

- Lo è all'inizio della ricorsione perché $n \geq k$.
- Il numero di elementi da inserire diminuisce di 1 ad ogni passo, mentre `num_miss` o resta uguale o diminuisce di 1. Quindi la differenza può solo diminuire al massimo di 1 per passo.
- Quando la differenza diventa 0, resta 0 ed alla sequenza vengono solo aggiunti valori che non apparivano prima.

Questo dimostra che ogni chiamata (nodo interno dell'albero di ricorsione) produce almeno una sequenza. Quindi #chiamate $\leq nS(n,k)$ e ogni chiamata cosa al massimo $O(k)$. La complessità quindi è quella desiderata.