ESERCIZIO1

Abbiamo due interi k ed n con $1 \leq k \leq n$. Vogliamo sapere quanti diversi modi ci sono di partizionare l'insieme dei primi n interi in k sottoinsiemi.

Ad esempio per $k = 2$ ed $n = 3$ l'algoritmo deve restituire 3, possiamo infatti partizionare i 3 interi in due sottoinsiemi nei modi seguenti:

- $\{\{1,2\}, \{3\}\}$
- $\{\{1\}, \{2,3\}\}$
- $\{\{1,3\}, \{2\}\}$

Progettare un algoritmo che risolve il problema in tempo $O(n \cdot k)$.

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

Utilizziamo una tabella T di dimensione $(n+1) \times (k+1)$ dove $T[i][j] =$ il numero partizioni per i primi i interi in j sottoinsiemi .
La soluzione sarà in $T[n][k]$.

La formula ricorsiva che permette di ricavare $T[i][j]$ dalle celle precedentemente calcolate è la seguente:

$$T[i][j] = \begin{cases} 1 & \text{se } j = 1 \\ 0 & \text{se } i = 1 \\ j * T[i-1][j] + T[i-1][j-1] & \text{altrimenti} \end{cases}$$

Infatti:

- c'è un solo modo di partizione gli elementi in un solo sottoinsieme: $T[i][1] = 1$
- non c'è modo di partizione in più insiemi un unico elemento $T[1][j] = 0$ per $j > 1$
- per $i, j > 1$ possono accadere due cose:
  - l'intero i finisce in una partizione da solo. Questo può accadere in $T[i-1][j-1]$
  - l'intero i finisce in un insieme con altri elementi. Questo può accadere in $j * T[i-1][j]$

$$T[i][j] = \begin{cases} 1 & \text{se } j = 1 \\ 0 & \text{se } i = 1 \\ j * T[i-1][j] + T[i-1][j-1] & \text{altrimenti} \end{cases}$$

```
def conta(n, k):
    T = [[0 for j in range(k + 1)] for i in range(n + 1)]
    for i in range(1, n + 1):
        for j in range(1, k + 1):
            if j==1: T[i][j]=1
            elif i==1: T[i][j]=0
            else:
                T[i][j] = j * T[i - 1][j] + T[i - 1][j - 1]
    return T[n][k]
```

Complessità $\Theta(n)$

ESERCIZIO 2

Progettare un algoritmo che, dato un intero n, stampa tutte le stringhe binarie di lunghezza $2 \cdot n$ tali che il numero di uni presenti nella prima metà della stringa è lo stesso del numero di uni presenti nella seconda metà.

Ad esempio per $n = 2$ l'algoritmo deve stampare, non necessariamente nello stesso ordine, le seguenti 6 stringhe:

$1111 \quad 1010 \quad 1001 \quad 0110 \quad 0101 \quad 0000$

L'algoritmo proposto deve avere complessità $O(nS(n))$ dove $S(n)$ è il numero di stringhe da stampare.

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

Algoritmo:

- Utilizziamo un algoritmo di backtracking per la stampa di tutte le stringhe binarie lunghe $2n$ con l'aggiunta di funzioni di taglio.
- per la generazione dei primi n bit della stringa tutti i nodi vengono generati e viene anche utilizzato un contatore *tot* che riporta il numero di 1 inseriti fino a quel momento.
- dopo l'inserimento dei primi n bit il contatore *tot* viene decrementato ad ogni inserimento di un simbolo 1 e la generazione dei nodi è soggetta a due funzioni di taglio:
  - uno 0 viene inserito nella stringa solo se restano poi sufficienti bit da settare per portare a zero il valore di *tot*
  - un 1 viene inserito nella stringa solo se il contatore *tot* è positivo.
- grazie alle due funzioni di taglio la soluzione parziale via via costruita è sempre un prefisso di una soluzione da stampare.

Implementazione:

```
def BT(n,i,tot,sol):
    if i==2*n:
        print (''.join(sol))
    else:
        if i<n:
            for x in {'0','1'}:
                sol.append(x)
                BT(n,i+1,tot+int(x),sol)
                sol.pop()
        else:
            if tot>0:
                sol.append('1')
                BT(n,i+1,tot-1,sol)
                sol.pop()
            if 2*n-(i+1)>=tot:
                sol.append('0')
                BT(n,i+1,tot,sol)
                sol.pop()
```

```
>>> BT(3,0,0,[])
111111
110110
110101
110011
101110
101101
101011
100100
100010
100001
011110
011101
011011
010100
010010
010001
001100
001010
001001
```

- Nota che nell'albero di ricorsione prodotto dall'esecuzione di BT un nodo viene generato solo se porta ad una foglia da stampare.
- Possiamo quindi dire che la complessità dell'esecuzione di $bk(n, 0, 0, sol)$ richiederà tempo

$$O(S(n) \cdot h \cdot f(n) + S(n) \cdot g(n))$$

  dove:

  - $h = 2 \cdot n$ è l'altezza dell'albero.
  - $f(n) = O(1)$ è il lavoro di un nodo interno.
  - $g(n) = O(n)$ è il lavoro di una foglia

- Quindi il costo di $BT(n, 0, 0, [\,])$ è $O(nS(n))$.

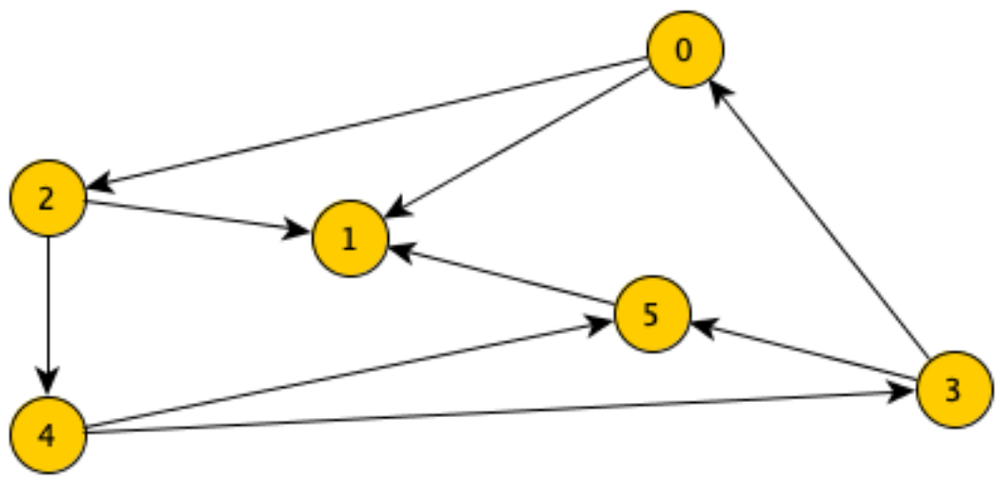
ESERCIZIO 3

Considerate il grafo G in figura.

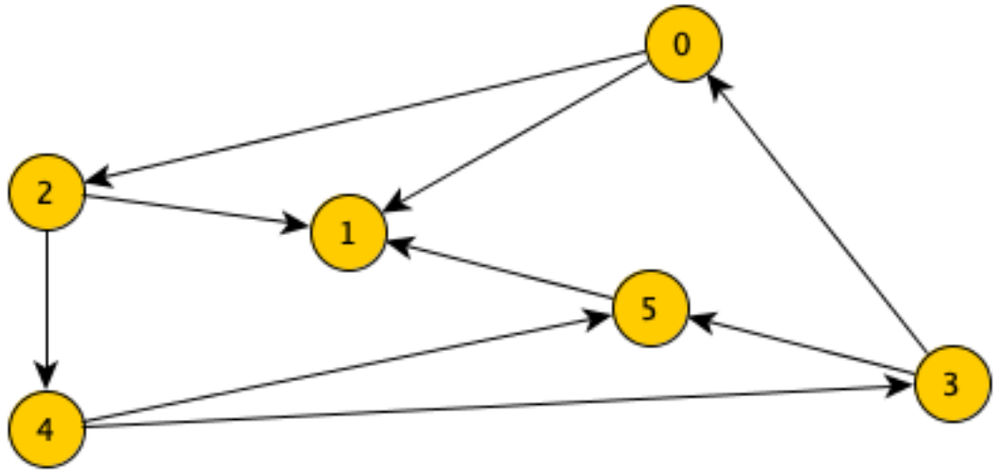
1. Riportate l'albero dfs che si ottiene eseguendo una visita di G a partire dal nodo 0.
2. dire quali sono gli archi di attraversamento, quali gli archi in avanti e quali gli archi all'indietro che si incontrano nel corso della visita.

Nel corso della visita ogni qualvolta un nodo ha più vicini non visitati scegliete sempre quello di indice minimo (in altre parole assumete il grafo rappresentato tramite liste di adiacenza dove i nodi di ciascuna lista sono ordinati per indice crescente).

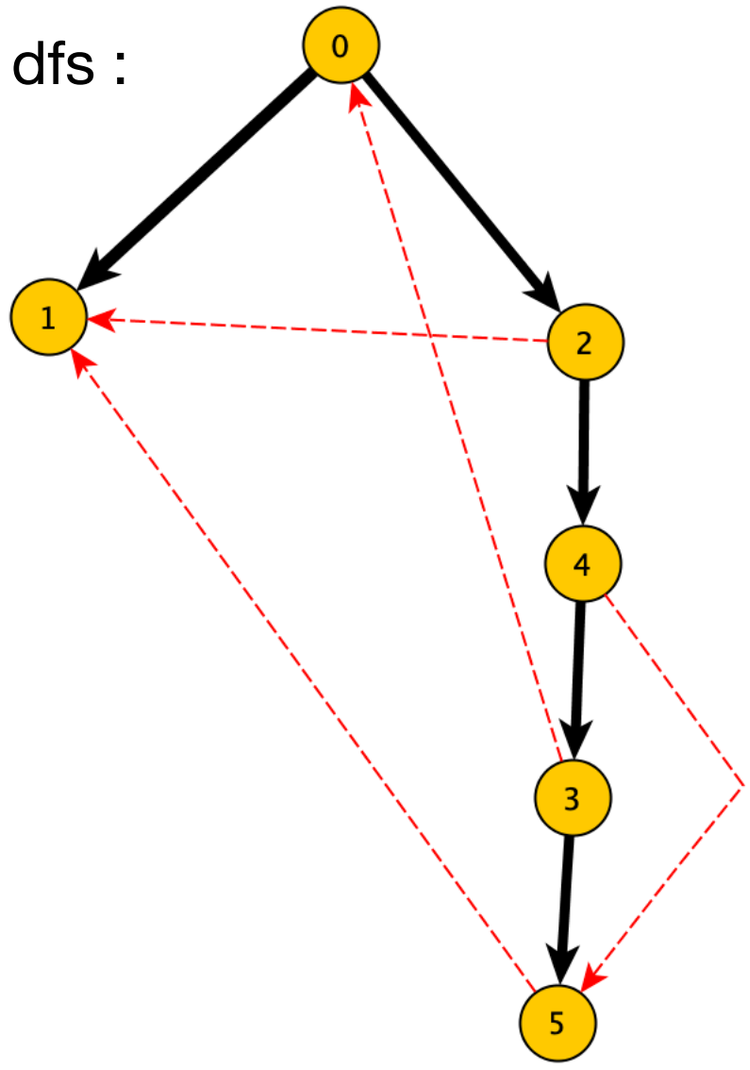
G=



G=



albero dfs :



archi all'indietro: $3 \longrightarrow 0$

archi in avanti: $4 \longrightarrow 5$

archi di attraversamento: $2 \longrightarrow 1, 5 \longrightarrow 1$