# ESERCIZIO 1

Un numero intero positivo può sempre essere rappresentato come somma di quadrati di altri numeri interi. Ad esempio, il generico numero $n$ può scomporsi come somma di $n$ addendi tutti uguali a $1^2$ (vale a dire $n = \sum_{i=1}^{n} 1^2$ ).
Dato un intero positivo $n$ vogliamo scoprire qual'è il numero minimo di quadrati necessari ad ottenere $n$.
Ad esempio:

a. per $n = 100$ la risposta è 1 infatti $100 = 10^2$ . Nota che vale anche $100 = 5^2 + 5^2 + 5^2 + 5^2$ ma questa scomposizione usa 4 quadrati e non è minimale. Analogamente non è minimale la scomposizione $100 = 8^2 + 6^2$ che usa 2 quadrati.

b. per $n = 6$ la risposta è 3 infatti $6 = 2^2 + 1^2 + 1^2$ e non è difficile vedere che 6 non può esprimersi come somma di due soli quadrati.

Scrivere lo pseudocodice di un algoritmo che risolve il problema in tempo $O\left(n^{\frac{3}{2}}\right)$
**Motivare BENE la correttezza e la complessità del vostro algoritmo.**

Utilizziamo una tabella $T$ di dimensione $n+1$ dove
$T[i]$ = il numero minimo di quadrati necessari per ottenere la somma $i$.
La soluzione sarà in $T[n]$.

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

$$T[i] = \begin{cases} 0 & \text{se } i = 0 \\ \max\left(T[i-x^2]+1 \mid 1 \leq x \leq \sqrt{i}\right) & \text{altrimenti} \end{cases}$$

Infatti:
- per ottenere 0 bastano 0 quadrati.
- per $i \leq 1$ :
  - serve almeno un quadrato di un numero $x$ dove $1 \leq x^2 \leq i$.
  - Se scelgo il quadrato di x poi devo ottenere tramite ulteriori quadrati il numero $i - x^2$
  - se scelgo il quadrato di $x$ allora il meglio che posso ottenere è $1 + T[i - x^2]$
  - devo quindi trovare il minimo valore $T[i - x^2] + 1$ per tutti i possibili $x$.

$$T[i] = \begin{cases} 0 & \text{se } i = 0 \\ \max\left(T[i - x^2] + 1 \mid 1 \leq x \leq \sqrt{i}\right) & \text{altrimenti} \end{cases}$$

```python
def quadrati(n):
    T = [0 for j in range(n + 1)]
    for i in range(1, n + 1):
        j=1
        T[i]=i
        while j*j<=i:
            T[i]=min(T[i],1+T[i-j*j])
            j+=1
    return T[n]
```

**il** $for$ **viene eseguito** $n$ **volte.**

**Ciascuna iterazione del** $for$ **richiede l'esecuzione del** $while$**.**

**il** $while$ **richiede** $\lfloor\sqrt{i}\rfloor \leq \sqrt{n}$ **iterazioni (ciascuna di costo costante).**

**Il costo totale dell'algoritmo è:** $n \cdot O\left(\sqrt{n}\right) = O\left(n \cdot n^{\frac{1}{2}}\right) = O\left(n^{\frac{3}{2}}\right)$

## ESERCIZIO 2

Progettare un algoritmo che, dato un intero $n$, stampa tutte le stringhe binarie di lunghezza $2n$ tali che il numero di zeri presenti nella prima metà della stringa è inferiore o uguale al numero di uni presenti nella seconda metà.
L'algoritmo proposto deve avere complessità $O(nS(n))$ dove $S(n)$ è il numero di stringhe da stampare.

Ad esempio per $n = 2$ l'algoritmo deve stampare, non necessariamente nello stesso ordine, le seguenti 11 stringhe:

0011 0101 0110 0111 1001 1010 1011 1100 1101 1110 1111

**Motivare BENE la correttezza e la complessità del vostro algoritmo**.

## Algoritmo:

- Utilizziamo un algoritmo di backtracking per la stampa di tutte le stringhe binarie lunghe $2n$ con l'aggiunta di una funzione 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 0 inseriti fino a quel momento.
- dopo l'inserimento dei primi $n$ bit:
  - L'inserimento di uno 0 è soggetto ad una funzione di taglio: lo 0 viene inserito nella stringa solo se restano poi sufficienti bit da settare per portare a zero il valore di $tot$
  - L'inserimento di un 1 è sempre possibile e causa un decremento del contatore $tot$.
- grazie alla funzione di taglio la soluzione parziale via via costruita è sempre un prefisso di una soluzione da stampare.

## Implementazione:

```python
def BT(n,i=0,tot=0,sol=[]):
    if i==2*n:
        print (''.join(sol))
    else:
        if i<n:
            sol.append('0')
            BT(n,i+1,tot+1,sol)
            sol.pop()
            sol.append('1')
            BT(n,i+1,tot,sol)
            sol.pop()
        else:
            if 2*n-(i+1)>=tot:
                sol.append('0')
                BT(n,i+1,tot,sol)
                sol.pop()
            sol.append('1')
            BT(n,i+1,tot-1,sol)
            sol.pop()
```

```
>>> BT(2)
0011
0101
0110
0111
1001
1010
1011
1100
1101
1110
1111
```

- 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

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

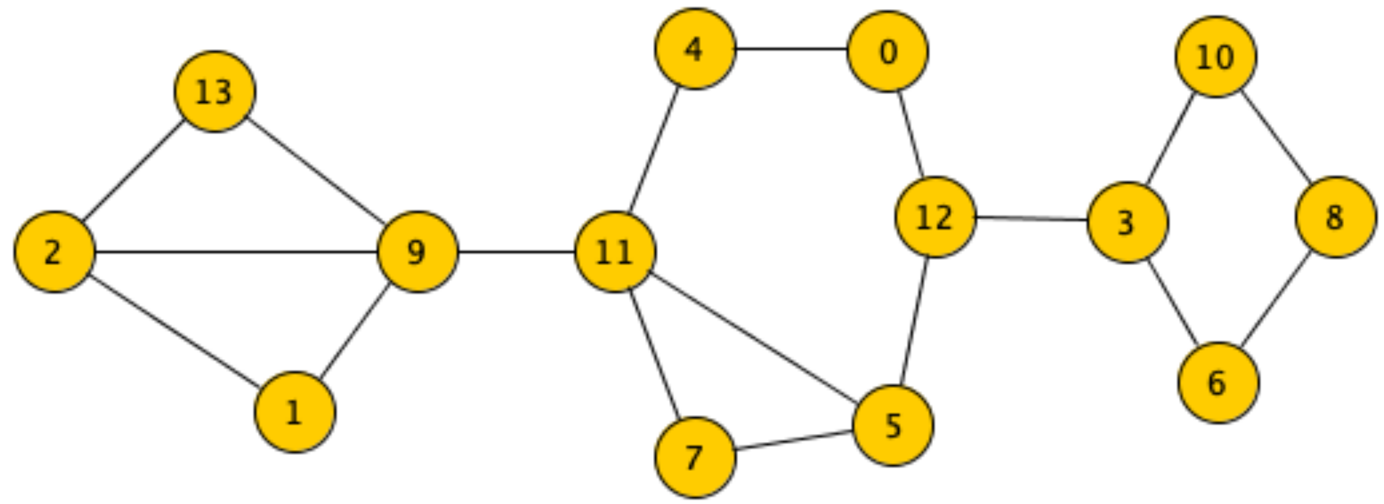
dove:

  - $h = 2 \cdot n$ 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 $O(nS(n))$.

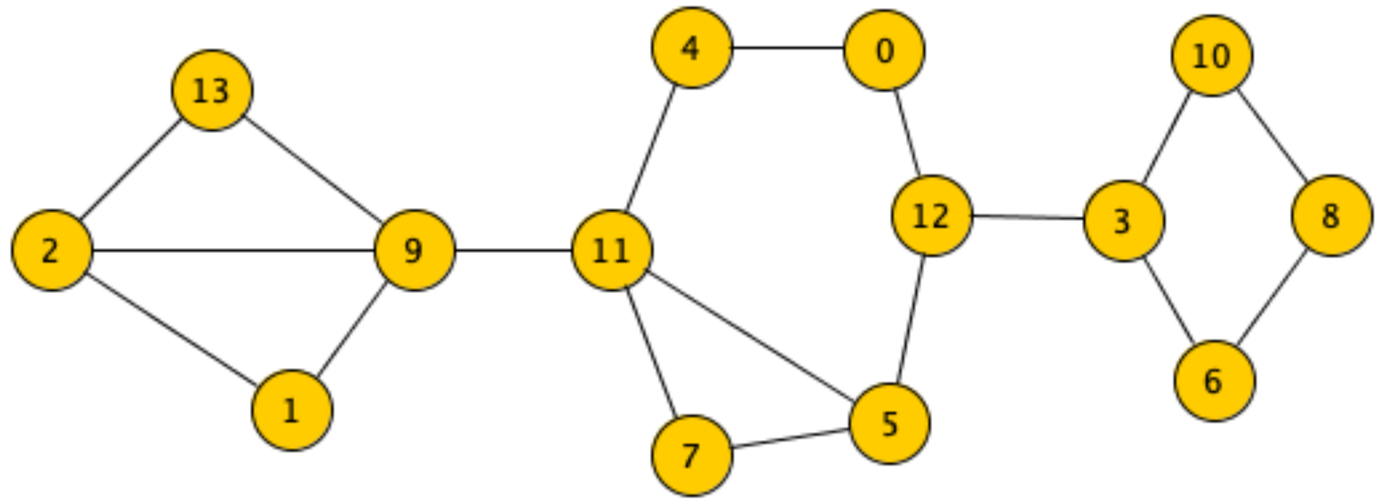
## ESERCIZIO 3

G=




Considerate il grafo $G$ in figura. Determinate il numero minimo di archi che bisogna eliminare da $G$ perché risulti due-colorabile.
**Motivare BENE la vostra risposta**.

G=




- Un grafo è 2-colorabile se e solo se non contiene cicli dispari.
- Il grafo $G$ contiene $4$ cicli dispari: $(2,13,9)$, $(2,9,1)$, $(11,5,7)$ $e$ $(11,4,0,12,5)$
- per eliminarli devo togliere almeno un arco da ciascuno dei $4$ cicli dispari.
- non c'è un arco che appartenga a $3$ cicli ma ci sono archi che appartengono 2 cicli
- l'eliminazione dell'arco $2-9$ permette di eliminare i due cicli $(2,13,9)$ $e$ $(2,9,1)$.
- l'eliminazione dell'arco $5-11$ permette di eliminare i due cicli $(11,5,7)$ $e$ $(11,4,0,12,5)$
- **Il numero minimo di archi da eliminare è 2** (più precisamente i due archi sono $2-9$ e $5-11$.