Partial Sum

Vogliamo implementare una struttura dati che supporti le seguenti operazioni:

`Init` che prende una lista A e restituisce una rappresentazione `Repr` di A.

`Lookup` che prende in input $i < j$ e `Repr` restituisce la somma $A[i] + A[i+1] + \ldots + A[j-1]$.

`Set` che prende in input k e `val` ed aggiorna la rappresentazione `Repr`.

Partial Sum

Descriviamo un'implementazione che, per una lista di N elementi, usa spazio $O(N)$ e le operazioni prendono tempo

`Init`	$O(N)$
`Lookup`	$O(\log N)$
`Set`	$O(\log N)$

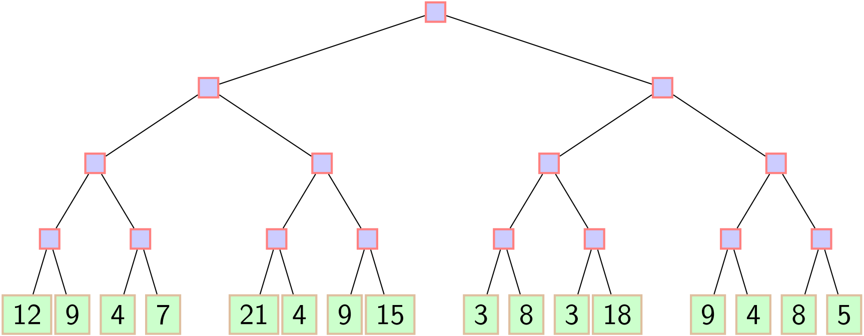
Init: costruire la rappresentazione

Precomputiamo le risposte alle seguenti query:

- $(0,1),(1,2),\ldots,(N-1,N)$:
  tutte le query $(i, i+1)$ *di lunghezza* 1*, per* i *multiplo di* 1*.*
- $(0,2),(2,4),\ldots,(N-2,N)$:
  tutte le query $(i, i+2)$ *di lunghezza* 2*, per* i *multiplo di* 2*.*
- $(0,4),(4,8),\ldots(N-4,N)$:
  tutte le query $(i, i+4)$ *di lunghezza* 4*, per* i *multiplo di* 4*.*
- $\ldots\ldots\ldots$
- $(0,N)$:
  tutte le query $(i, i+N)$ *di lunghezza* N*, per* i *multiplo di* N*.*

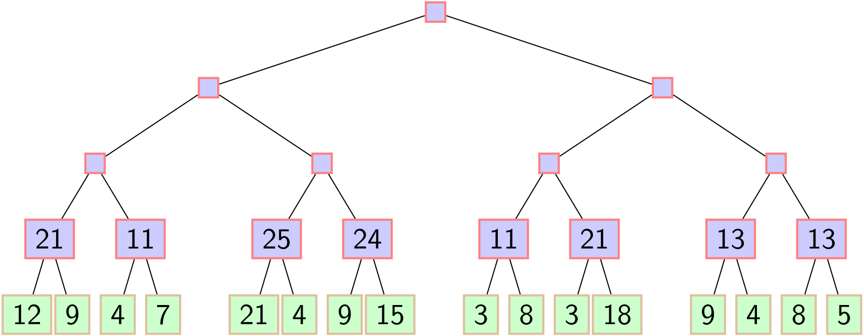
$$[12, 9, 4, 7, 21, 4, 9, 15, 3, 8, 3, 18, 9, 4, 8, 5]$$

Risposte a Lookup($i, i+1$), per i multiplo di 1



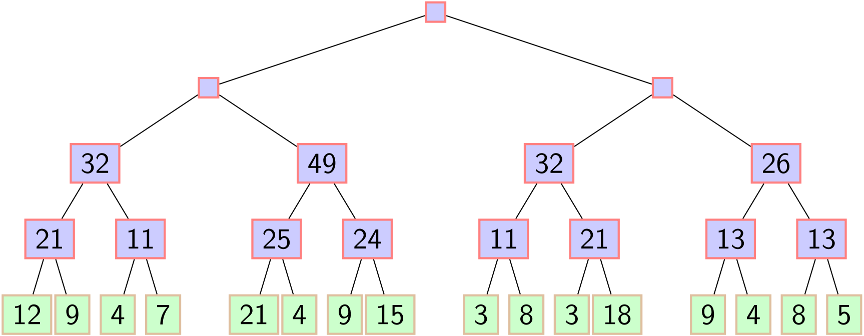
$$[12, 9, 4, 7, 21, 4, 9, 15, 3, 8, 3, 18, 9, 4, 8, 5]$$

Risposte a Lookup$(i, i+2)$, per i multiplo di 2



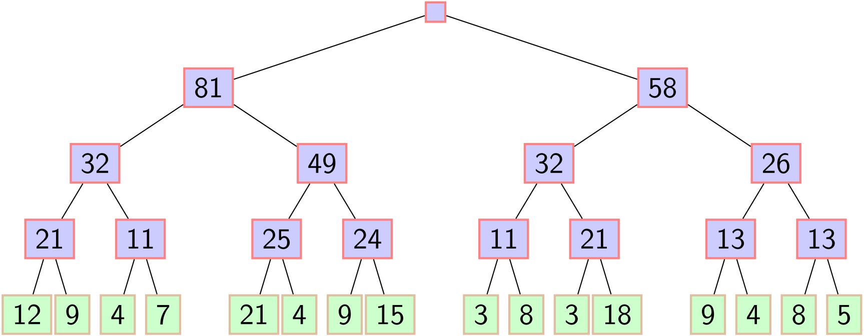
$$[12, 9, 4, 7, 21, 4, 9, 15, 3, 8, 3, 18, 9, 4, 8, 5]$$

Risposte a Lookup$(i, i+4)$, per i multiplo di 4



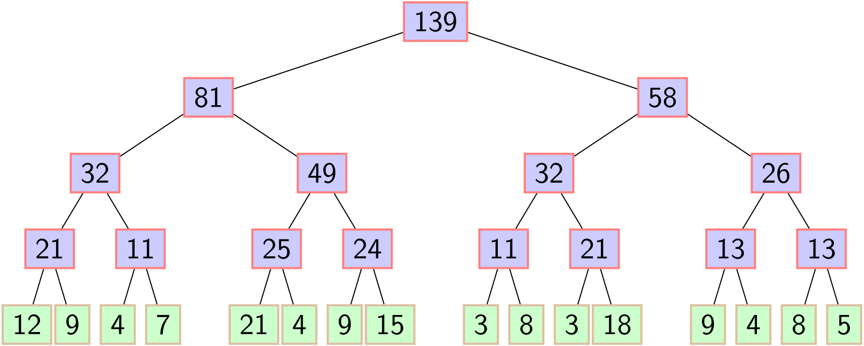
$[12, 9, 4, 7, 21, 4, 9, 15, 3, 8, 3, 18, 9, 4, 8, 5]$

Risposte a Lookup$(i, i + 8)$, per i multiplo di 8

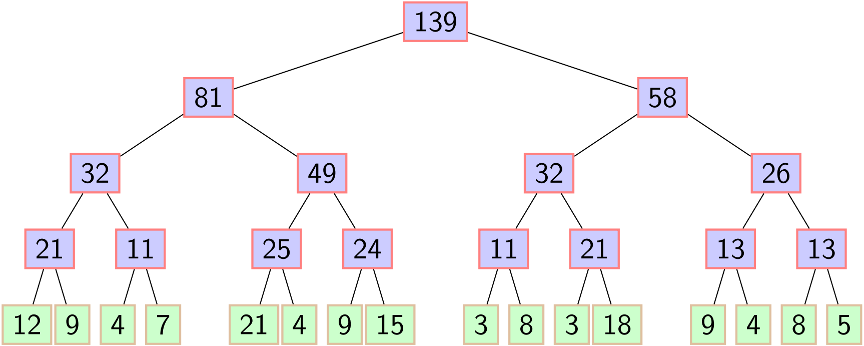


$$[12, 9, 4, 7, 21, 4, 9, 15, 3, 8, 3, 18, 9, 4, 8, 5]$$

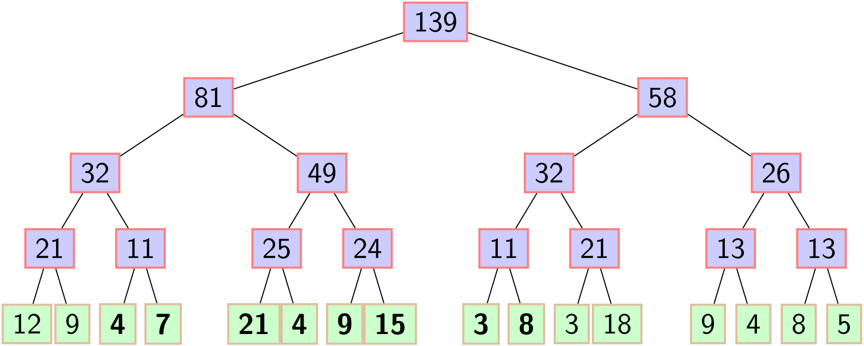
Risposte a Lookup$(i, i + 16)$, per i multiplo di 16



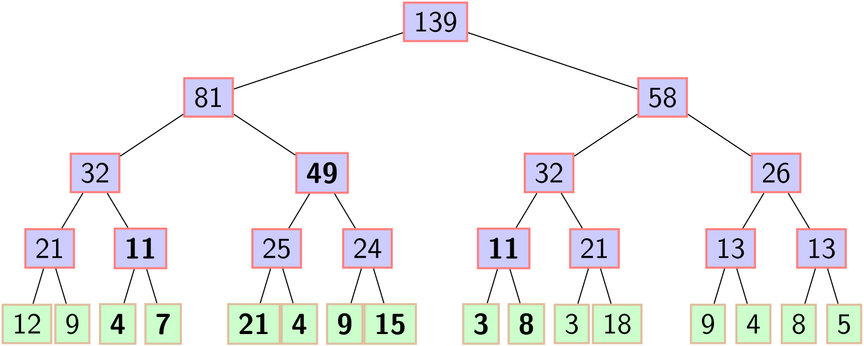
$$\mathrm{Lookup}(2,10) = \mathrm{Lookup}(2,4) + \mathrm{Lookup}(4,8) + \mathrm{Lookup}(8,10)$$



$$\mathrm{Lookup}(2,10) = \mathrm{Lookup}(2,4) + \mathrm{Lookup}(4,8) + \mathrm{Lookup}(8,10)$$



$$\mathrm{Lookup}(2,10) = \mathrm{Lookup}(2,4) + \mathrm{Lookup}(4,8) + \mathrm{Lookup}(8,10)$$



Implementazione di Lookup

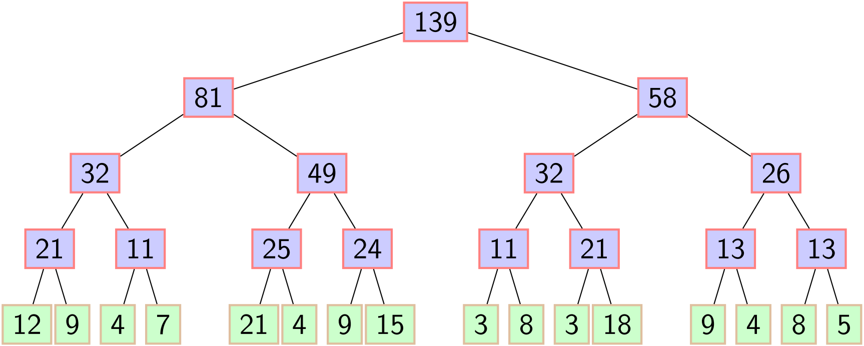
```
def _lookup(self,x,i,j,s,t):   #want answer to [i,j)
                               #[i,j) included in [s,t)
                               #x index of the element with answer to [s,t)
    if i==j:        #sum of the empty interval
        return 0

    if i==s and j==t:          #[i,j)=[s,t) answer found at index x
        ##print(x,i,j,s,t,"finisce")
        return self.A[x]

    m=(t+s)//2
    if j<m:
        ##print(x,i,j,s,t,"diventa",2*x+1,i,j,s,m)
        return self._lookup(2*x+1,i,j,s,m)
    if i>m:
        ##print(x,i,j,s,t,"diventa",2*x+2,i,j,m,t)
        return self._lookup(2*x+2,i,j,m,t)
    ##print(x,i,j,s,t,"diventa",2*x+1,i,m,s,m,"+",2*x+2,m,j,m,t)
    return self._lookup(2*x+1,i,m,s,m)+self._lookup(2*x+2,m,j,m,t)
```

- Intervallo $[i, j)$ incluso in intervallo $[s, t)$
- x è l'indice che contiene la soluzione per l'intervallo $[s, t)$
- Chiamata iniziale

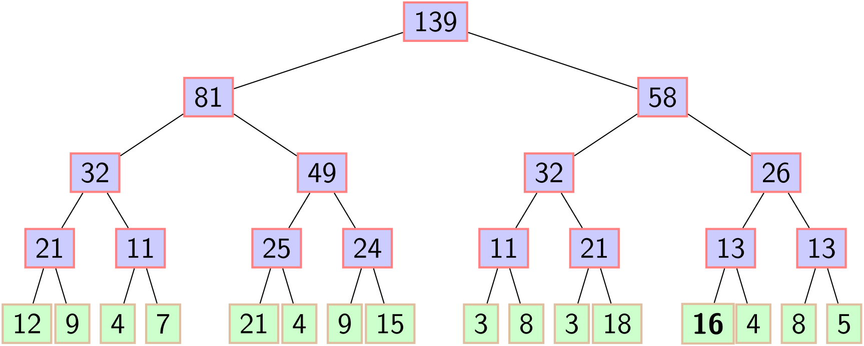
$$\_\mathtt{lookup}(self, 0, i, j, 0, self.N)$$

  - $x = 0$
  - $[s, t) = [0, N)$

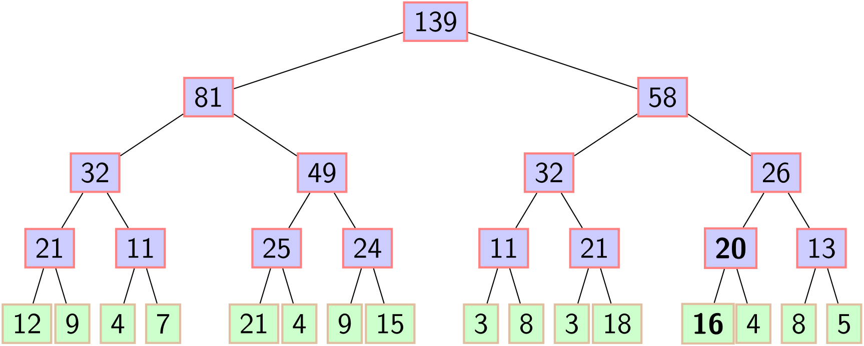
$\mathrm{Set}(12, 16)$



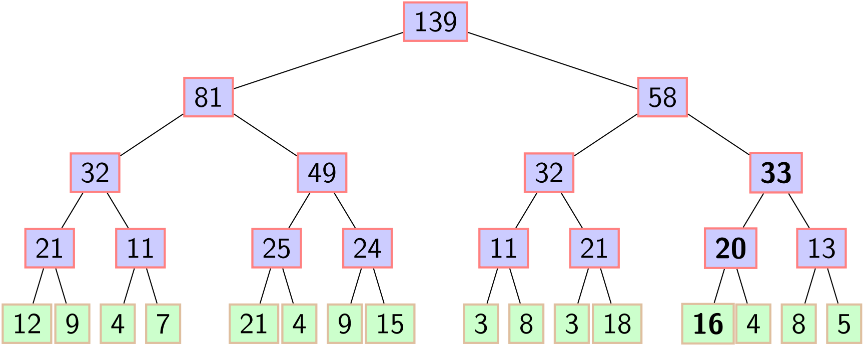
$\mathsf{Set}(12, 16)$



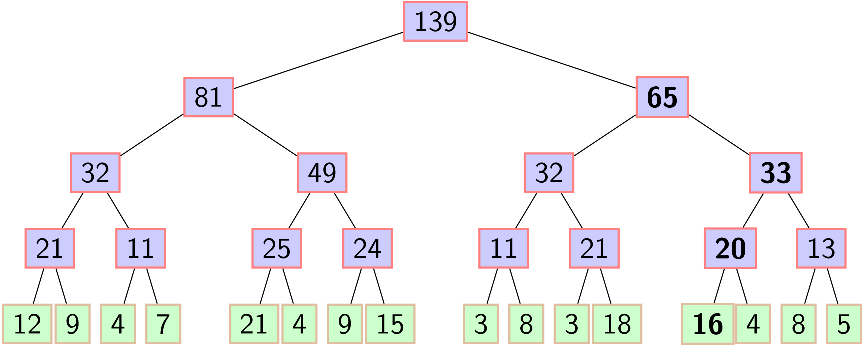
$\mathsf{Set}(12, 16)$



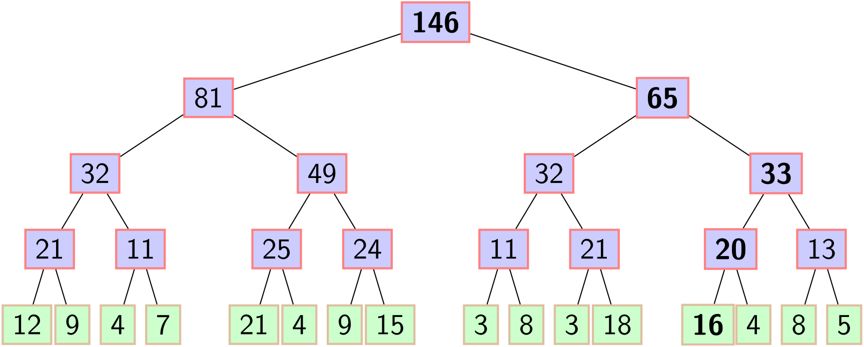
Set(12, 16)


139
81
58
32
49
32
33
21
11
25
24
11
21
20
13
12
9
4
7
21
4
9
15
3
8
3
18
16
4
8
5

Set(12, 16)

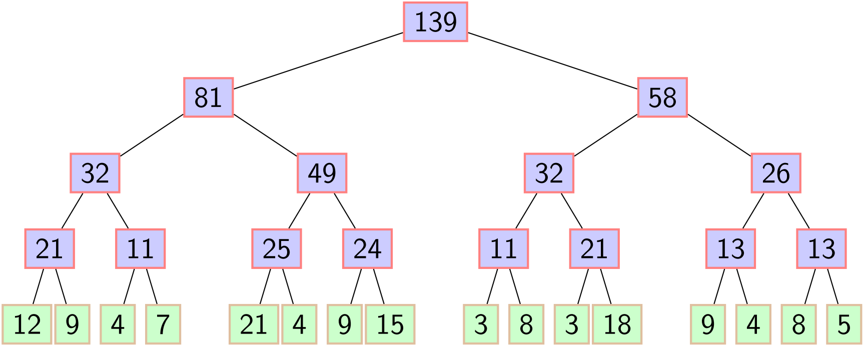

139
81
65
32
49
32
33
21
11
25
24
11
21
20
13
12
9
4
7
21
4
9
15
3
8
3
18
16
4
8
5

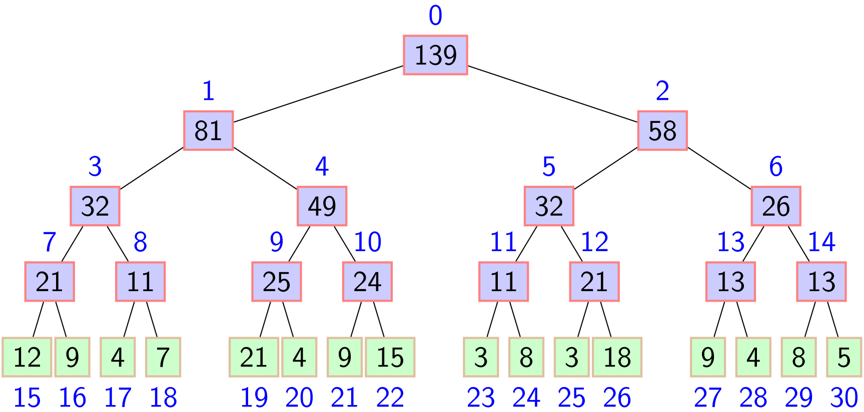
$\mathrm{Set}(12, 16)$



Implementazione di Set

```python
def __setitem__(self,idx,val):
    i=idx+self.N-1 #idx indice in lista originale
                   #i indice nella rappresentazione
    diff=val-self.A[i] ##diff=nuovo valore - vecchio valore
    while(i>=0):
        ##print(f'{"Posizione:":10s}{i:3d}{"  "}{self.A[i]:3d}{"--->"}{self.A[i]+diff:3d}')
        self.A[i]+=diff
        i=(i-1)//2
```


139
81
58
32
49
32
26
21
11
25
24
11
21
13
13
12 9 4 7 21 4 9 15 3 8 3 18 9 4 8 5


0
139
1
81
2
58
3
32
4
49
5
32
6
26
7
21
8
11
9
25
10
24
11
11
12
21
13
13
14
13
12 9 4 7 21 4 9 15 3 8 3 18 9 4 8 5
15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

Construire Repr

Primo tentativo: Insert in lista

```
def __init__(self,B):
    self.N=len(B)
    self.A=B.copy()
    start=len(B)-1
    while start>0:
        end=0
        while start>end:
            t=self.A[start]+self.A[start-1]
            start=start-2  #consumo due elementi
            self.A.insert(0,t)
            start=start+1
            end=end+1 #annullo l'effetto della insert su start e end
```

`insert` takes linear time

Construire Repr

Secondo tentativo: Append in lista

```
def __init__(self,B):
    self.A=B.copy()
    self.A.reverse()
    self.N=len(B)
    start=0
    end=self.N
    while start<end-1:
        for i in range(start,end,2):
            self.A.append(self.A[i]+self.A[i+1])
        l=(end-start)//2
        start=end
        end=end+l
    self.A.reverse()
```

append takes constant time

Approccio generale

- Precalcolare la soluzione ad un sottoinsieme S delle possibili query
- La risposta ad ogni lookup si ottiene combinando alcune soluzioni di S
  Sia k il numero massimo di soluzioni da combinare per una lookup
- Ogni valore $A[i]$ influenza un sottoinsieme di soluzioni precomputate che devono essere modificati come effetto di una set.
  Sia m il numero massimo di soluzioni da aggiornare per una set.

Implementazioni

- **Nessuna precomputazione**
  - S contiene tutti problemi di grandezza 1; $|S| = N$
  - $k = N, m = 1$
- **Precomputazione tutte le soluzioni**
  - S contiene tutti i problemi; $|S| = O(N^2)$
  - $k = 1,\ m = N$
- **Precomputazione dei prefissi**
  - S contiene tutti i prefissi; $|S| = N$
  - $k = 2,\ m = N$
- **Algoritmo efficienti**
  - $|S| = O(N)$
  - $k, m = O(\log N)$