

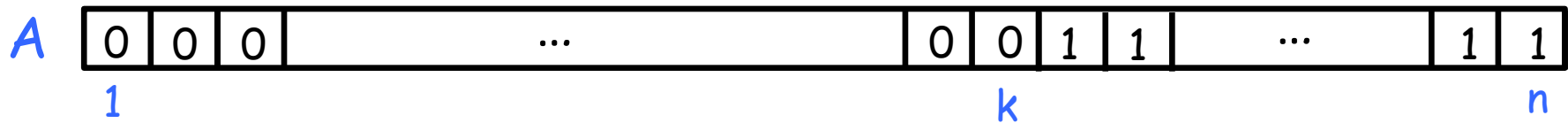
Esercitazione
27 novembre 2024

problema 1

Progettare un algoritmo efficiente per il seguente problema.

Input: vettore ordinato $A[1:n]$ di n bit, ovvero $A[i] \in \{0,1\}$

Output: l'indice k dell'ultimo 0 (numero di zeri)



goal 1: $O(\log n)$

idea: uso l'approccio della ricerca binaria.

Algorithm 2: UltimoZeroRic(A, i, j)

```
if  $i > j$  then
  | return -1
 $m = \lfloor \frac{i+j}{2} \rfloor$  ;
if  $A[m] = 0$  e  $A[m+1] = 1$  then
  | return  $m$ 
if  $A[m] = 1$  then
  | return UltimoZeroRic( $A, i, m-1$ )
else
  | return UltimoZeroRic( $A, m+1, j$ )
```

complessità?

$O(\log n)$

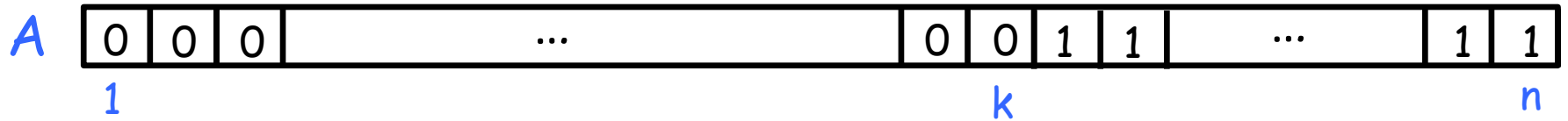
Algorithm 1: UltimoZero(A)

```
 $n =$  lunghezza di  $A$  ;
if  $A[n] = 0$  then
  | return  $n$ 
else
  | return UltimoZeroRic( $A, 1, n-1$ )
```

Progettare un algoritmo efficiente per il seguente problema.

Input: vettore ordinato $A[1:n]$ di n bit, ovvero $A[i] \in \{0,1\}$

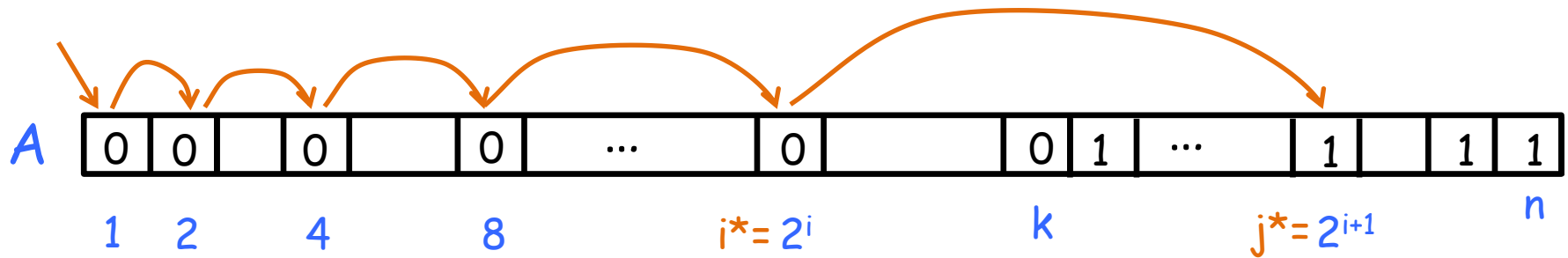
Output: l'indice k dell'ultimo 0 (numero di zeri)



goal 1: $O(\log n)$

goal 2: $O(\log k)$ $\leftarrow$ mai peggiore

idea



idea: trovare in $O(\log k)$ due indici i^* e j^* tale che:

- $A[i^*]=0$ e $A[j^*]=1$
- $|j^*-i^*|=O(k)$ su cui fare ricerca binaria in tempo $O(\log k)$

analisi:

Ho guardato $i+2=O(i)$ elementi

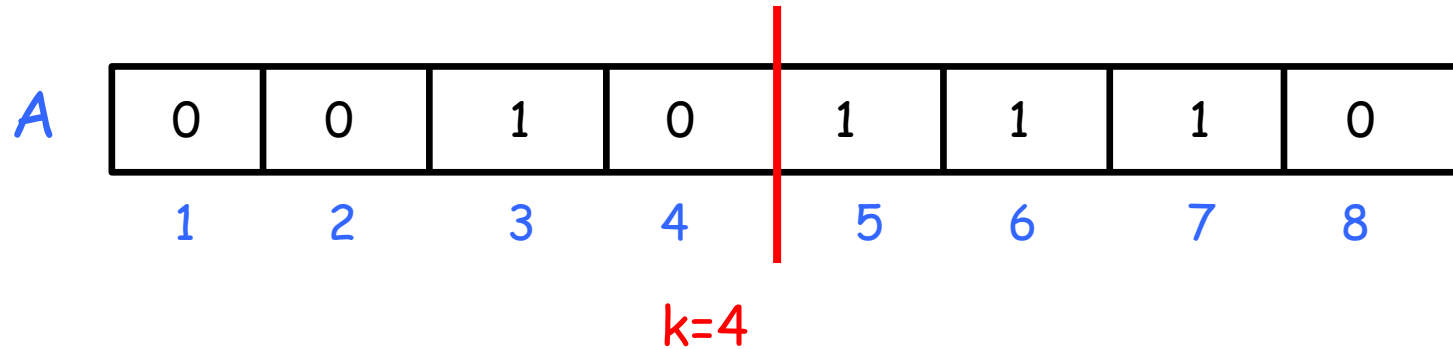
- $2^i \leq k \implies i \leq \log_2 k$
- $j^*-i^* = 2^{i+1} - 2^i = 2^i \leq k$
 $\implies A[i^*;j^*]$ ha j^*-i^*+1 elementi $\implies O(k)$ elementi

problema 2

Progettare un algoritmo efficiente per il seguente problema.

Input: vettore $A[1:n]$ di n bit, ovvero $A[i] \in \{0,1\}$

Output: un indice k tale che $\# \text{di zeri in } A[1:k] = \# \text{di uni in } A[k+1:n]$



goal: $O(n)$

idea: calcolare in tempo $O(n)$ due vettori di dimensione n :

- $Z[1:n]$, con $Z[j] = \#$ di zeri in $A[1:j]$

- $U[1:n]$, con $U[j] = \#$ di uni in $A[j+1:n]$

Calcolo $Z[]$ e $U[]$

if $A[1]=0$ then $Z[1]=1$ else $Z[1]=0$

$U[n]=0$

for $j=2$ to n do

for $j=n-1$ down to 1 do

if $A[j]=0$ then $Z[j]=Z[j-1]+1$

$U[j]=U[j+1]+A[j+1]$

else $Z[j]=Z[j-1]$

A

0	0	1	0	1	1	1	0
---	---	---	---	---	---	---	---

1 2 3 4 5 6 7 8

Z

1	2	2	3	3	3	3	4
---	---	---	---	---	---	---	---

1 2 3 4 5 6 7 8

U

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

1 2 3 4 5 6 7 8

l'algoritmo

Taglia(A)

if A[1]=0 **then** Z[1]=1 **else** Z[1]=0

for j=2 to n **do**

if A[j]=0 **then** Z[j]=Z[j-1]+1

else Z[j]=Z[j-1]

U[n]=0

for j=n-1 down to 1 **do**

 U[j]=U[j+1]+A[j+1]

for j=1 to n **do**

if Z[j]=U[j] **then return** j

return 0

complessità?

$O(n)$

A

0	0	1	0	1	1	1	0
1	2	3	4	5	6	7	8

un altro algoritmo

Taglia(A)

cont=0

for i=1 to n **do**

 cont=cont+A[i]

return cont

complessità?

$O(n)$

correttezza?

N : #di uni in A

Δ_j : (#di uni in $A[j+1:n]$) - (#di zeri in $A[1:j]$) [$=U[j] - Z[j]$]

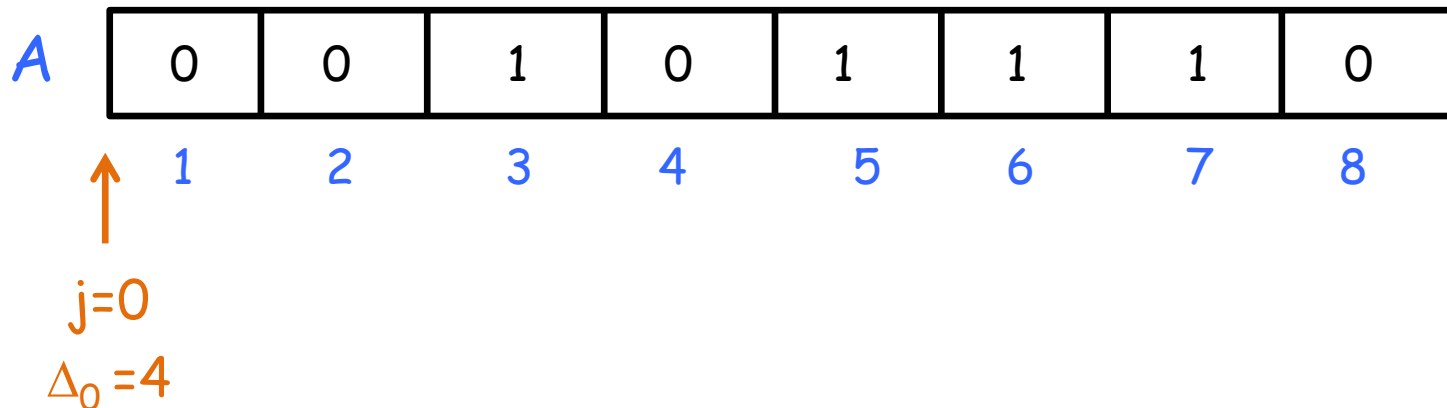
goal: voglio un indice k tale che $\Delta_k=0$

Claim: $\Delta_N = 0$

dim

$$\Delta_0 = N$$

$$\Delta_j = \Delta_{j-1} - 1$$



N : #di uni in A

Δ_j : (#di uni in $A[j+1:n]$) - (#di zeri in $A[1:j]$) [$=U[j] - Z[j]$]

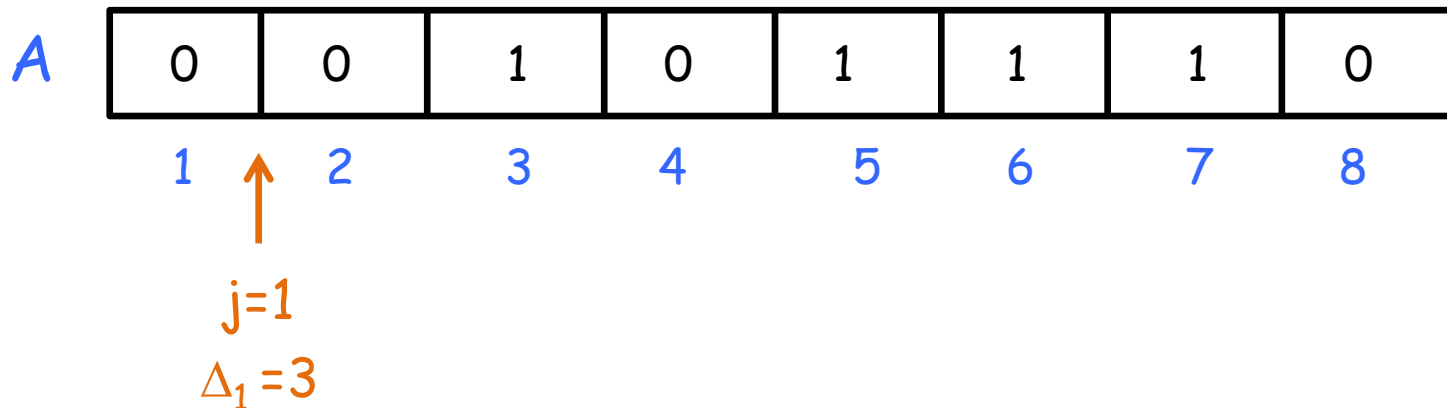
goal: voglio un indice k tale che $\Delta_k=0$

Claim: $\Delta_N = 0$

dim

$$\Delta_0 = N$$

$$\Delta_j = \Delta_{j-1} - 1$$



N : #di uni in A

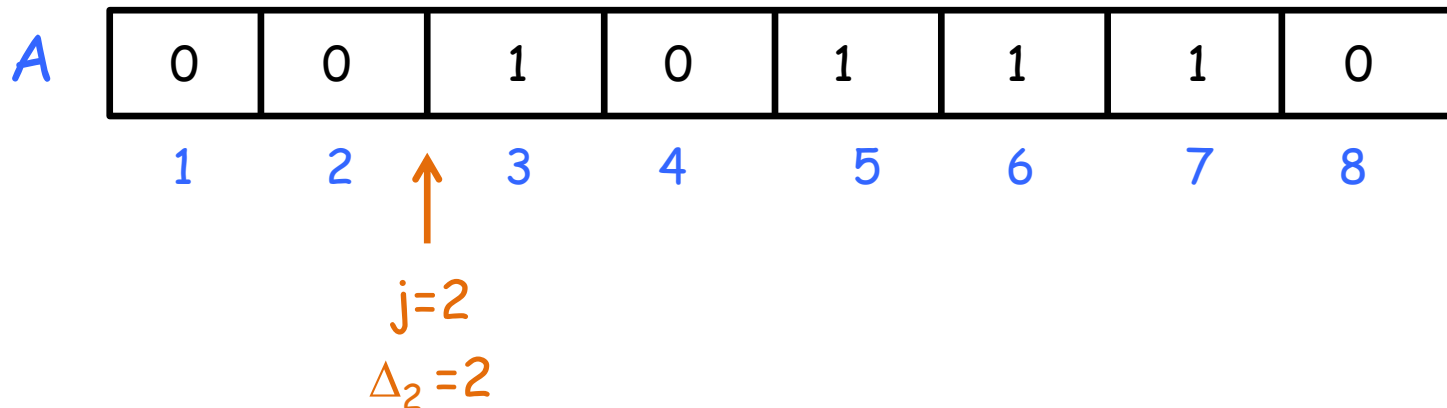
Δ_j : (#di uni in $A[j+1:n]$) - (#di zeri in $A[1:j]$) [$=U[j] - Z[j]$]

goal: voglio un indice k tale che $\Delta_k=0$

Claim: $\Delta_N = 0$
dim

$$\Delta_0 = N$$

$$\Delta_j = \Delta_{j-1} - 1$$



N : #di uni in A

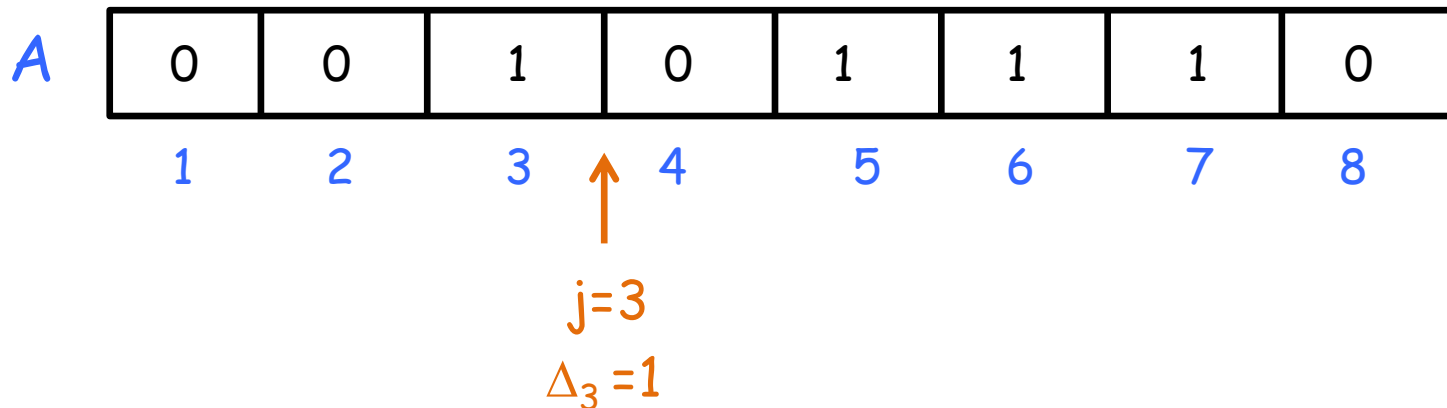
Δ_j : (#di uni in $A[j+1:n]$) - (#di zeri in $A[1:j]$) [$=U[j] - Z[j]$]

goal: voglio un indice k tale che $\Delta_k=0$

Claim: $\Delta_N = 0$
dim

$$\Delta_0 = N$$

$$\Delta_j = \Delta_{j-1} - 1$$



N : #di uni in A

Δ_j : (#di uni in $A[j+1:n]$) - (#di zeri in $A[1:j]$) [$=U[j] - Z[j]$]

goal: voglio un indice k tale che $\Delta_k=0$

Claim: $\Delta_N = 0$

dim

$$\Delta_0 = N$$

$$\Delta_j = \Delta_{j-1} - 1$$

