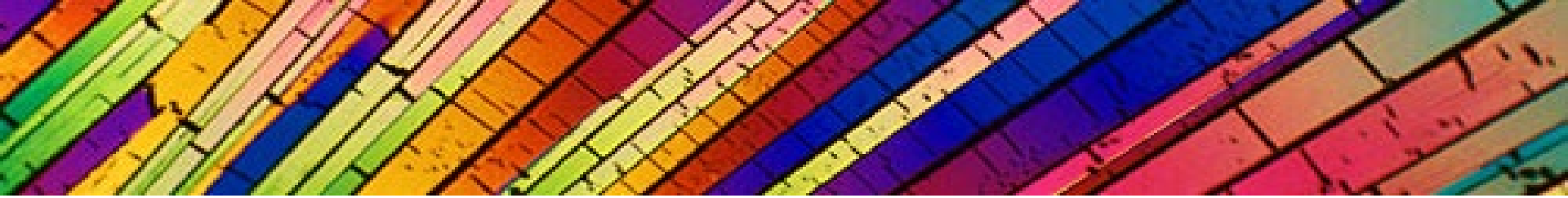


# Algoritmi di base
## e programmazione a stati

Ver. 3

# Algoritmi di ordinamento

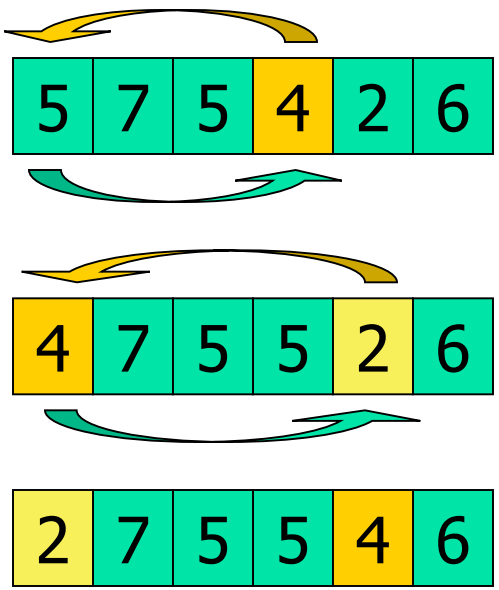
- Lo scopo è ordinare in senso crescente o decrescente il contenuto di un vettore di N elementi, possibilmente senza utilizzare un secondo vettore
- Esiste molta letteratura scientifica a riguardo e molti metodi adatti a vari casi
- In queste slide si vedranno tre semplici algoritmi di esempio per l'ordinamento in senso crescente (o meglio, non-decrescente):
  - Selection sort
  - Bubble sort
  - Insertion sort

# Selection sort semplificato - I

- Si cerca il min degli elementi del vettore e lo si mette nella prima posizione, ossia:
  si confronta il 1° elemento del vettore con tutti i seguenti, ogni volta che si trova un valore minore del 1° li si scambia

*è il successivo a 0*

```
for (j=  1; j<N; j++)
   if (v[j] < v[0])
   {
      tmp = v[j];
      v[j] = v[0];
      v[0] = tmp;
   }
```

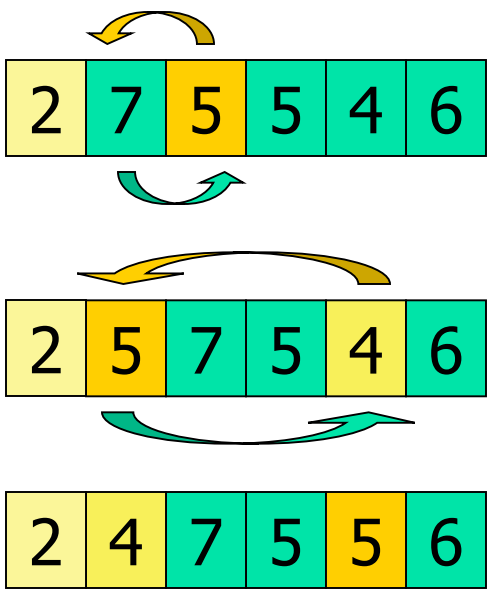


# Selection sort semplificato - II

- Si ripete lo stesso identico procedimento per tutti gli elementi a partire dal secondo, ossia: si confronta il 2° elemento del vettore con tutti i seguenti, ogni volta che si trova un valore minore del 2° li si scambia

*è il successivo a 1*

```
for (j=  2; j<N; j++)
   if (v[j] < v[1])
   {
      tmp = v[j];
      v[j] = v[1];
      v[1] = tmp;
   }
```



# Selection sort semplificato - III

- In generale, si applica questo procedimento a tutti gli elementi a partire dall'i-esimo, così si determina di questi il valore più piccolo che dopo gli scambi si trova alla posizione i-esima

*è il successivo a* **i**

```
for (j=i+1; j<N; j++)
   if (v[j] < v[i])
   {
      tmp = v[j];
      v[j] = v[i];
      v[i] = tmp;
   }
```

# Selection sort semplificato -IV

- Per ordinare completamente il vettore si ripete questo procedimento per tutti i valori di **i** da 0 fino al *penultimo* (l'ultimo va a posto da sé): ogni iterazione trova il valore più piccolo tra i restanti e lo colloca nella posizione i-esima

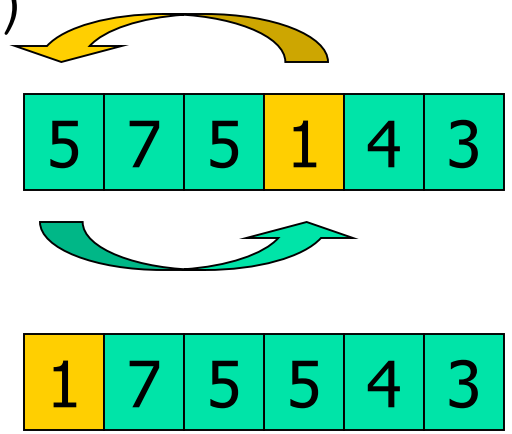
```
for (i=0; i<N-1; i++)
   for (j=i+1; j<N; j++)
      if (v[j] < v[i])
      {
         tmp = v[j];
         v[j] = v[i];
         v[i] = tmp;
      }
```

# Selection sort - I

- Il primo passo è individuare la *posizione* del valore minimo tra gli elementi del vettore e scambiare questo con quello nella prima posizione, il primo valore è ora al posto giusto

*è il successivo a 0*

```
jmin = 0;
for (j=  1; j<N; j++)
   if (v[j] < v[jmin])
      jmin = j;
tmp = v[jmin];
v[jmin] = v[0];
v[0] = tmp;
```

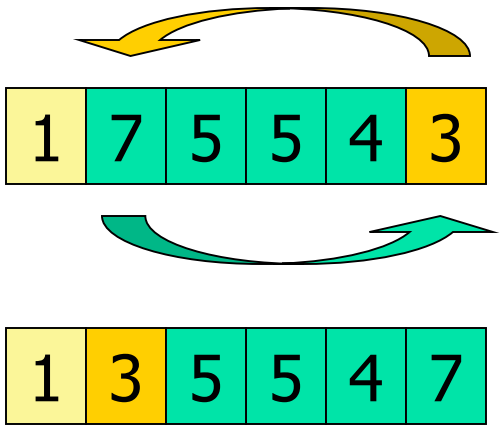


# Selection sort - II

- Si ripete lo stesso identico procedimento per tutti gli elementi a partire dal secondo: si determina la posizione del secondo valore più piccolo e lo si colloca al secondo posto

*è il successivo a 1*

```
jmin = 1;
for (j=  2; j<N; j++)
   if (v[j] < v[jmin])
      jmin = j;
tmp = v[jmin];
v[jmin] = v[1];
v[1] = tmp;
```



# Selection sort - III

- In generale, si applica questo procedimento a tutti gli elementi a partire dall'i-esimo, così si determina di questi il valore più piccolo che dopo gli scambi si trova alla posizione i-esima

*è il successivo a* **i**

```
jmin = i;
for (j=i+1; j<N; j++)
   if (v[j] < v[jmin])
      jmin = j;
tmp = v[jmin];
v[jmin] = v[i];
v[i] = tmp;
```

# Selection sort - IV

- Se si ripete questo procedimento per tutti i valori di **i** da 0 fino al penultimo (l'ultimo va a posto da sé) si ottiene l'ordinamento in senso crescente di tutto il vettore

```
for (i=0; i<N-1; i++)
{ jmin = i;
  for (j=i+1; j<N; j++)
    if (v[j] < v[jmin])
      jmin = j;
  tmp = v[jmin];
  v[jmin] = v[i];
  v[i] = tmp;
}
```

*Se non viene trovato un min lo scambio avviene comunque*

# Selection sort - V

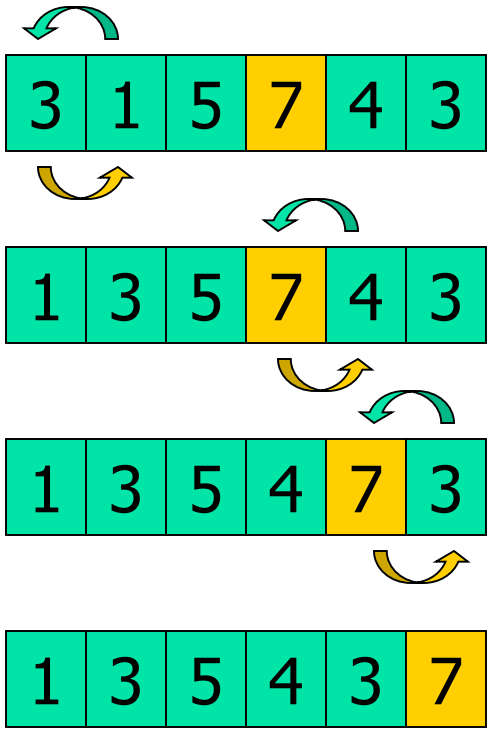
- Quando non viene trovato un min, lo scambio avviene comunque, per evitarlo:

```
for (i=0; i<N-1; i++)
{ jmin = i;
  for (j=i+1; j<N; j++)
    if (v[j] < v[jmin])
      jmin = j;
  if (i != jmin)
  {
    tmp = v[jmin];
    v[jmin] = v[i];
    v[i] = tmp;
  }
}
```

# Bubble sort - I

- Se si scorrono tutti gli elementi di un vettore e ogni volta che si trovano due valori ADIACENTI non in ordine (il più piccolo a destra del più grande) li si scambia: il più grande di tutti risale a destra

```
for (j=0; j<N-1; j++)
   if (v[j] > v[j+1])
   {
      tmp = v[j];
      v[j] = v[j+1];
      v[j+1] = tmp;
   }
```



# Bubble sort - II

- Ripetendo N-1 volte questa operazione, tutti i valori risalgono verso destra fino ad occupare la posizione corretta e quindi vengono ordinati in senso crescente

```
for (i=0  ; i<N-1; i++)
   for (j=0; j<N-1; j++)
      if (v[j] > v[j+1])
      {
         tmp = v[j];
         v[j] = v[j+1];
         v[j+1] = tmp;
      }
```

# Bubble sort - III

- *Inefficienza:* i valori già sistemati a destra vengono comunque confrontati; per evitarlo si anticipa la fine del ciclo interno, sfruttando il ciclo esterno: a ogni iterazione si valuta un valore in meno

```
for (i=N-1; i>0  ; i--)
   for (j=0; j<i  ; j++)
      if (v[j] > v[j+1])
      {
         tmp = v[j];
         v[j] = v[j+1];
         v[j+1] = tmp;
      }
```

# Bubble sort - IV

- *Inefficienza:* se pochi valori sono fuori posto, l'ordinamento si ottiene con meno di N-1 passate, per non fare passate inutili si fa terminare il ciclo se non ci sono stati scambi

```
scambi = SI;
for (i=N-1; i>0 && scambi==SI ;i--)
{ scambi = NO;
  for (j=0; j<i  ; j++)
    if (v[j] > v[j+1])
    { tmp = v[j];
      v[j] = v[j+1];
      v[j+1] = tmp;
      scambi = SI; }
}
```

continua solo se ci sono stati scambi

# Bubble sort - V

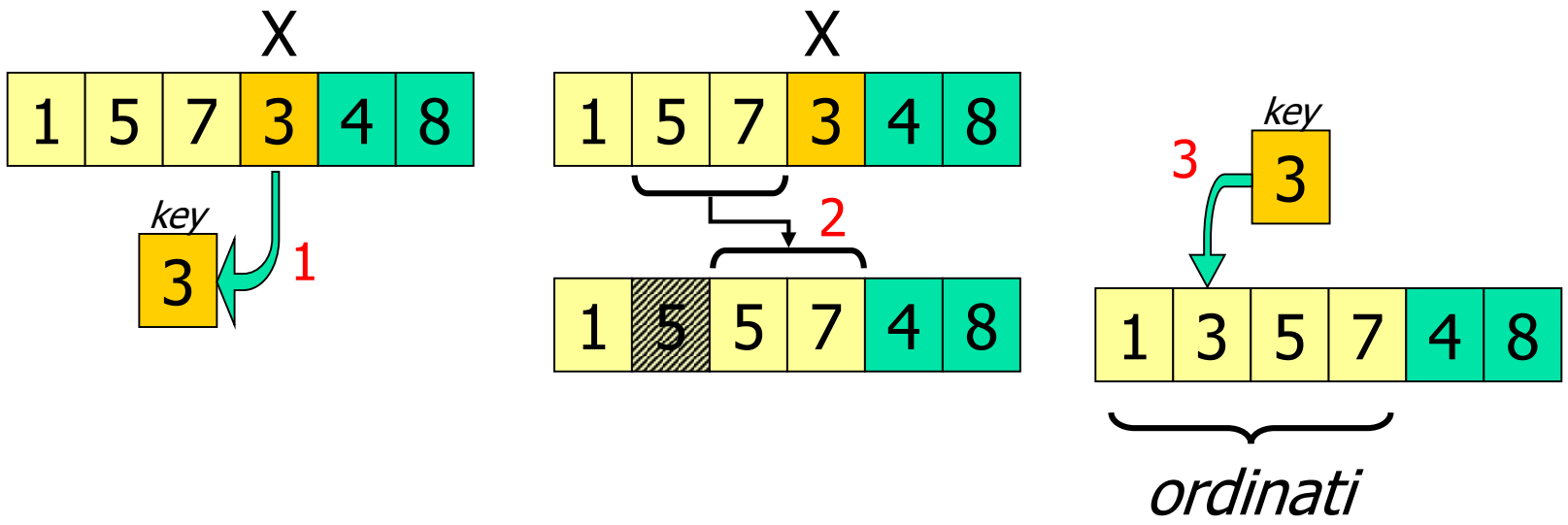
- Un ulteriore miglioramento si può avere ricordando la posizione dell'ultimo valore sistemato: i valori successivi sono già ordinati
- Si memorizza quindi la posizione dell'`ultimo` scambio e si fa far fermare il ciclo interno a quella posizione-1 (basta impostare il valore `i` a `ultimo`)
- `ultimo` funge anche da `scambio` in quanto quando è 0 indica che non ci sono stati scambi e il ciclo esterno termina

# Bubble sort - V

```
for (i=N-1; i>0  ; i=ultimo)
{
   ultimo=0;
   for (j=0; j<i  ; j++)
      if (v[j] > v[j+1])
      {
         tmp = v[j];
         v[j] = v[j+1];
         v[j+1] = tmp;
         ultimo = j;
   }
}
```

# Insertion sort - I

- Per ciascuno degli elementi X, a partire dal secondo a sinistra fino all'ultimo a destra:
  1. copia l'elemento X in una variabile (*key*)
  2. scala verso destra di una posizione tutti gli elementi del vettore a sinistra di X maggiori di *key*
  3. copia *key* nella posizione più a sinistra di quelle appena scalate



# Insertion sort - I

- Il punto 2 si realizza con un ciclo che parte dall'elemento precedente X al primo:
  - se l'elemento `v[i]` è maggiore di *key* lo copia nella posizione seguente: `v[i+1] = v[i]`
  - altrimenti copia *key* al posto dell'ultimo elemento spostato: `v[i+1] = key`
- Si noti che gli elementi a sinistra di ciascuno degli X sono già stati ordinati in senso crescente dalle passate precedenti, quindi basta fermarsi al primo elemento di valore inferiore (o uguale) a X alla sua sinistra

# Insertion sort - II

```
for (j=1; j<N; j++)
{
   key = v[j];
   for (i=j-1; i>=0 && v[i]>key; i--)
      v[i+1] = v[i];
   v[i+1] = key;
}
```

# Esercizi

1. Per ciascuno degli algoritmi di ordinamento precedentemente descritti (variazioni incluse), si scriva un programma completo che ordini in senso **crescente** (o meglio, non-decrescente) un vettore di N numeri interi.
2. Se ne misurino i tempi di esecuzione usando la funzione `clock` in `<time.h>` (vedere la soluzione se non si conosce la funzione `clock`)
3. Per ciascuno degli algoritmi di ordinamento precedentemente descritti (variazioni incluse), si scriva un programma completo che ordini in senso **non-crescente** un vettore di N interi.

# Algoritmi di ricerca

- Si vuole cercare un dato valore `val` in un vettore `vett` di N elementi
- Si possono considerare i due casi:
  - vettore non ordinato
  - vettore ordinato

# Ricerca in vettore non ordinato

- Si deve scorrere tutto il vettore in quanto `val` può essere in qualsiasi posizione

```
for (i=0; i<N && vett[i]!=val; i++)
   ;
if (i == N)
   printf("Non trovato\n");
else
   printf("Trovato\n");
```

- Finché non lo trova, continua a cercare
- Se lo trova, è alla posizione `i`

# Ricerca in vettore non ordinato

- Il ciclo termina in 2 casi:
  - è stato trovato il valore
  - ha finito di cercare (invano) in tutti gli elementi
- Deve stabilire per quale motivo il ciclo è finito
  - se trova: `i` < N
  - se non lo trova: `i` = N
- Non si deve controllare se `val` è stato trovato verificando la condizione `vett[i]=val` perché se non lo trova `i` vale N e `vett[N]` identifica un elemento inesistente
- Al massimo (nel caso peggiore) fa N controlli

# Ricerca in vettore ordinato
## Ricerca lineare

- Non serve scorrere tutto il vettore, ci si ferma non appena si supera il valore cercato

```
for (i=0; i<N && vett[i]<val; i++)
   ;
if (i<N && val == vett[i])
  printf("Trovato\n");
else
  printf("Non trovato\n");
```

- Quando `vett[i] >= val` il ciclo viene interrotto e si controlla se `val` è stato trovato
- `i<N` perché a fine ciclo si ha `i==N`
- Al massimo fa N controlli

# Ricerca in vettore ordinato
## Ricerca dicotomica (o binaria)

```
left=0; right=N-1;
while (right>=left)
{ m=(left+right)/2;
  if (val==vett[m])
    break;          → trovato
  if (val<vett[m])
    right=m-1;      → elimina la metà destra
  else
    left=m+1;       → elimina la metà sinistra
}
if (val==vett[m])
  printf("Trovato\n");
else
  printf("Non trovato\n");
```

# Ricerca in vettore ordinato
## Dicotomica (o binaria)

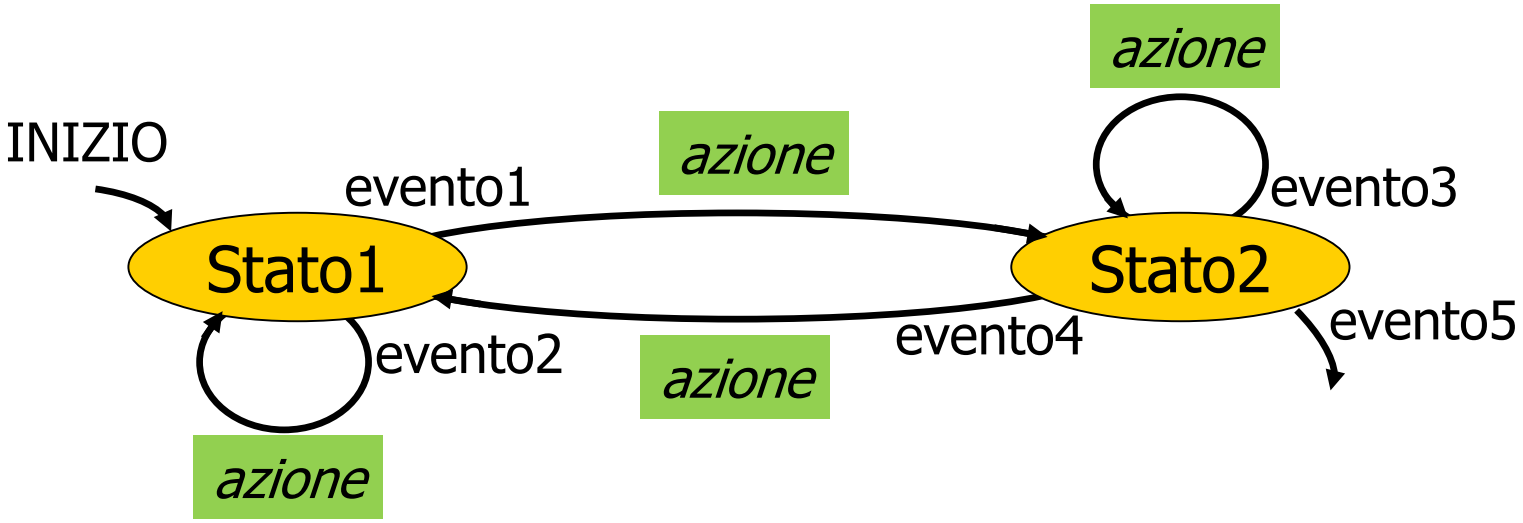
- Inizializza due indici `left` e `right` al primo e all'ultimo indice del vettore (0 e N-1)
- Calcola l'indice del valore centrale: `m`
- Se `vett[m]` è il valore cercato, termina
- Altrimenti se `val` è minore del valore centrale, arretra l'indice destro `right` al centro `m` (così dimezza il numero di valori in cui cercare)
- Altrimenti se `val` è maggiore del valore centrale, avanza l'indice sinistro `left` al centro
- Ripetendo questa operazione, si dimezza ogni volta il vettore
- Veloce: al massimo fa $\lceil \log_2(N+1) \rceil$ controlli

# Programmazione a stati

- Per *stato* si intende la situazione del programma in un dato istante (es. "sta leggendo i caratteri di una parola")
- Mentre il programma viene eseguito esso cambia di stato in seguito al verificarsi di eventi (es. lettura di un carattere da analizzare), la transizione può portare allo stesso stato di partenza (anello)
- Ad esempio quando si analizza una stringa, si possono avere gli stati DENTRO (ha letto un carattere alfabetico) e FUORI (ha letto un carattere non alfabetico)

# Programmazione a stati

- Graficamente:
  - gli stati sono disegnati come cerchi o ovali
  - le transizioni sono disegnate come archi orientati
  - nei punti di partenza degli archi si indicano gli eventi che causano le transizioni di stato
  - sugli archi si indicano le azioni da intraprendere durante le transizioni



# Programmazione a stati

- Nel codice, l'informazione sullo stato viene mantenuta tramite una *variabile di stato*, in genere di tipo `enum` o `int`
- Quindi i valori che la variabile di stato può assumere sono in genere costanti definite da `enum` o con `#define`
- Se i valori sono costanti e dagli stati partono diverse possibili transizioni, lo `switch` è spesso il costrutto di elezione

# Programmazione a stati

- Esempio
  Si scriva un programma che chieda una frase e conti quante sono le parole che iniziano con ciascuna lettera dell'alfabeto.
  Esempio di output:

```
Parole che iniziano con A: 4
Parole che iniziano con B: 1
Parole che iniziano con C: 0
…
Parole che iniziano con Z: 0
```

# Programmazione a stati

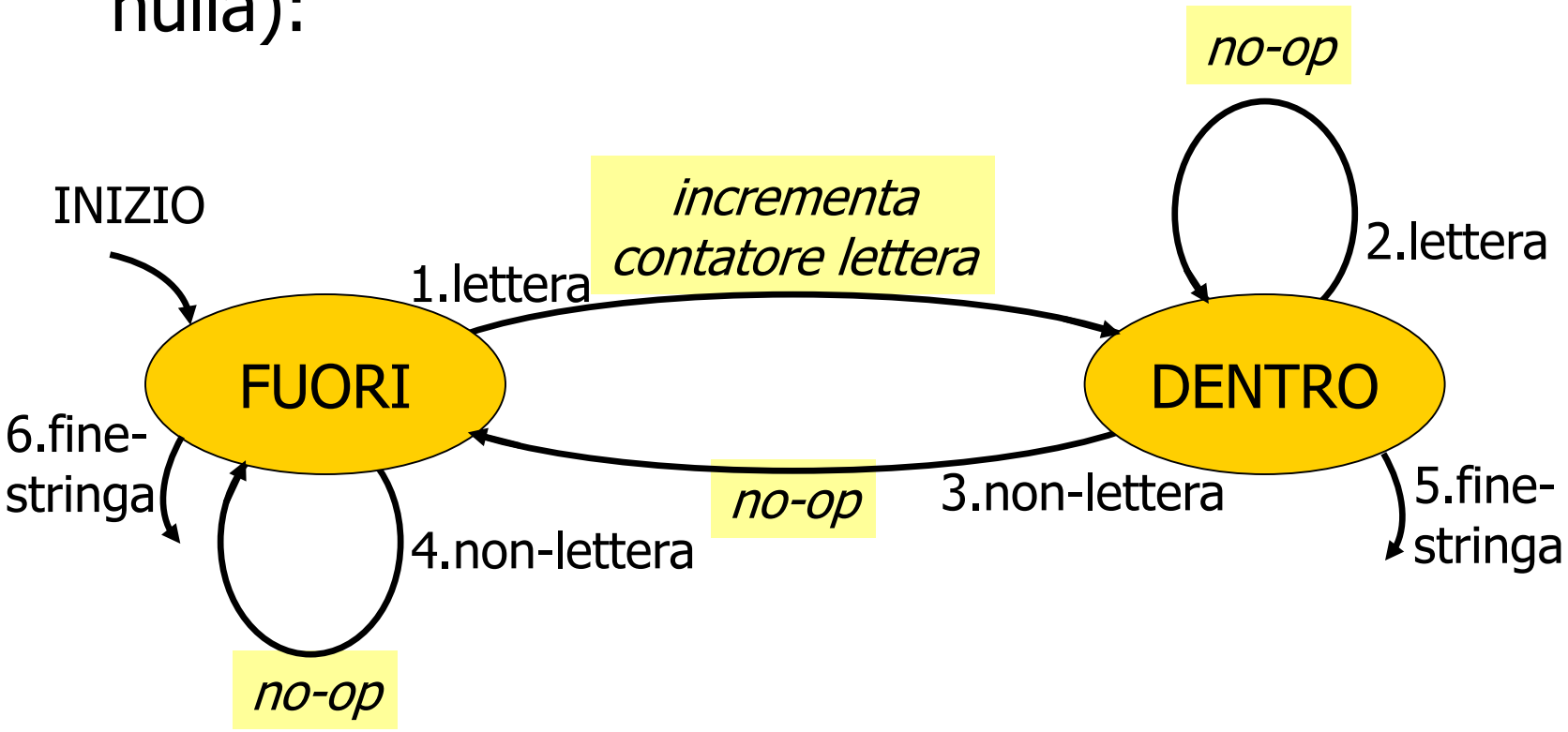
- La variabile di stato indicherà se si stanno leggendo i caratteri di una parola (si è "dentro" una parola) o no (si è "fuori" da una parola) e quindi assumerà 2 possibili valori:
  - DENTRO – FUORI
- L'evento che può causare il cambiamento di stato è la lettura del carattere successivo
- In questo esempio ci sono solo 4 casi:
  - stato FUORI e si legge un carattere alfabetico
  - stato DENTRO e si legge un carattere alfabetico
  - stato DENTRO e si legge un carattere non-alfabetico
  - stato FUORI e si legge un carattere non-alfabetico

# Programmazione a stati

- Più in dettaglio, quando si legge un carattere:
  1. se stato FUORI ed è carattere alfabetico:
     entra DENTRO (cambia lo stato in DENTRO) e incrementa il contatore di quel carattere
  2. se stato DENTRO ed è carattere alfabetico:
     resta DENTRO (non cambia lo stato) e non fa nulla
  3. se stato DENTRO e non è un carattere alfabetico:
     va FUORI dalla parola (cambia lo stato in FUORI) e non serve che faccia nulla
  4. se stato FUORI e non è un carattere alfabetico:
     resta FUORI (non cambia lo stato) e non fa nulla
  5. 6. se è \0:
     termina

# Programmazione a stati

- Rappresentazione grafica corrispondente all'esempio (no-op=no operation, non fa nulla):



# Programmazione a stati

```
stato = FUORI;
for (i=0; frase[i]!='\0'; i++) /* 5. e 6. quando ha finito esce */
  switch (stato)
  {
    case FUORI:
      /* 1. se stato FUORI e car alfab: entra DENTRO e incr il cont. */
      if (isalpha(frase[i]))
      {
         stato = DENTRO;
         lettere[toupper(frase[i]) - 'A']++;
      }
      /* 4. se stato FUORI e car non alfab: resta FUORI, non fa nulla */
      break;
    case DENTRO:
      /* 3. se stato DENTRO e car non alfab: va FUORI, non fa nulla */
      if (!isalpha(frase[i]))
         stato = FUORI;
      /* 2. se stato DENTRO e car alfab: resta DENTRO, non fa nulla */
      break;
  }
```

# Esercizi

4. Si scriva un programma che chieda una frase e conti quante sono le parole che iniziano con ciascuna lettera dell'alfabeto.
   *Lo si risolva con una soluzione a stati.*
   Esempio di output:

```
Parole che iniziano con A: 4
Parole che iniziano con B: 1
Parole che iniziano con C: 0
…
Parole che iniziano con Z: 0
```

# Esercizi

5. (Es. 4 sui file) Si scriva un programma che chieda il nome di un file contenente un testo qualsiasi e di questo conti quante sono le parole che iniziano con ciascuna lettera dell'alfabeto.
   *Lo si risolva con una soluzione a stati.*

   Esempio di output:

```
Parole che iniziano con A: 45
Parole che iniziano con B: 12
Parole che iniziano con C: 27
…
Parole che iniziano con Z: 3
```

# Esercizi

6. Un'immagine è composta da al max 256x256 punti di colore bianco o nero, è rappresentata da una matrice di caratteri letta da file. Ogni elemento della matrice rappresenta un punto:
   - il carattere '1' rappresenta un punto nero
   - il carattere '0' rappresenta un punto bianco

   Si scriva un programma a stati che codifichi l'immagine sostituendo, per ciascuna riga di punti, le sequenze consecutive dello stesso carattere ('0' o '1') con il car. stesso seguito dal numero delle sue ripetizioni + spazio.
   Si salvi l'output su un altro file.
   Si rifaccia l'es. codificando invece le colonne.

# Esercizi

*Continuazione:*

Ad es. l'immagine seguente (qui 3x10):

0000000000 *(10 zeri)*
0111111100 *(1 zero, 7 uno, 2 zeri)*
0100000100 *(1 zero, 1 uno, 5 zeri, 1 uno, 2 zeri)*

viene codificata per righe come segue:

010
01 17 02
01 11 05 11 02

e per colonne come segue:

03
01 12
01 11 01
. . .