Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Tesi di Laurea

COMPOSIZIONE DI OPERATORI DI ORDINAMENTO CON CONTENITORI

COMPOSITION OF SORTING OPERATORS WITH CONTAINERS

ALESSIO SANTORO

Relatore: *Relatore*
Correlatore: *Correlatore*

Anno Accademico 2023-2024

"Inserire citazione"
— Inserire autore citazione

1

INTRODUZIONE

2

ALGORITMI DI ORDINAMENTO E CLASSI DI PATTERN

In questo capitolo verranno introdotti alcuni tra i algoritmi di ordinamento che utilizzano opportune strutture dati come contenitori intermedi per gli elementi della permutazione in input, in particolare stacksort, queuesort e bubblesort, e altri concetti necessari per l'analisi della composizione tra due o piú esecuzioni di tali algoritmi.

Durante la loro esecuzione questi algoritmi possono salvare gli elementi in un contenitore (la diversa struttura dati adottata definisce i diversi algoritmi) dalla quale poi vengono prelevati per essere aggiunti all'output.

In questa tesi gli algoritmi prendono in input permutazioni di interi, sebbene sia possibile estendere la trattazione a parole arbitrarie su un insieme totalmente ordinato (anche se non sempre in modo immediato). Per rappresentare le permutazioni si usa la notazione lineare, quindi una permutazione π di lunghezza n è una parola sull'alfabeto $\{1, 2, \ldots, n\}$ in cui ogni lettera compare una e una sola volta.

INVERSIONI In un permutazione $\pi = \pi_1\pi_2 \ldots \pi_n$ un'inversione è una coppia di valori (π_i, π_j), dove $i < j, \pi_i > \pi_j$. Una coppia di valori che non compone una inversione si dice una non inversione.

Una singola iterazione degli algoritmi che prenderemo in considerazione non garantisce l'ordinamento della permutazione in input, dunque gli algoritmi devono essere iterati piú volte, ogni volta sul risultato della iterazione precedente. In ogni caso alla fine della i-esima iterazione i maggiori i elementi avranno raggiunto la loro posizione finale, dunque sono necessarie al massimo $n-1$ iterazioni per ordinare la permutazione.

In questo capitolo si mostrerá il comportamento di una singola applica-

zione di questi algoritmi. Le proprietá trovate verranno poi sfruttate per lo studio di varie composizioni.
Infatti generalmente quando si parla di questi algoritmi essi vengono descritti in funzione di varie iterazioni che vengono applicate fino a che la permutazione non è ordinata, mentre qui esamineremo gli effetti di una singola iterazione.
Ad esempio prendendo l'algoritmo bubblesort (certamente il piú noto fra quelli che verranno presentati) si fará riferimento ad un operatore B definito appositamente: dove π è una permutazione di interi, $B(\pi)$ rappresenta il risultato di una sola iterazione di bubblesort su di essa.

ALGORITMI

Bubblesort

L'algoritmo di ordinamento bubblesort prevede di scorrere dal primo al penultimo elemento e confrontare ciascuno con il proprio successivo per scambiarli se non sono ordinati.
Si definisce l'operazione $Swap(\pi_i, \pi_j)$, che fissata una permutazione π la restituisce con l'i-esimo e il j-esimo elementi scambiati di posizione.
Il risultato di una singola iterazione di bubblesort su una permutazione $\pi = \pi_1\pi_2 \dots \pi_n$ è calcolato dall'operatore B e si indica con $B(\pi)$.
Si puó scrivere permutazione π come $\pi_L n \pi_R$, dove n è il massimo valore della permutazione, π_L è il suo prefisso e π_R il suo suffisso. Vale che $B(\pi) = B(\pi_L)\pi_r n$.

Algorithm 1 $B - \mathrm{bubblesort}$

```
for i = 1 to n − 1 do
    if π_i > π_{i+1} then
        Swap(π_i, π_{i+1})
    end if
end for
```

Stacksort

Stacksort è un algoritmo che utilizza una pila per ordinare una permutazione.

Si definisce una pila come un insieme di interi su cui si possono applicare le seguenti operazioni:

$\mathrm{Push}(\pi_i, S)$ aggiunge l'elemento π_i alla pila S

$\mathrm{Pop}(S)$ estrae dalla pila S l'ultimo intero inserito e lo aggiunge all'output

$\mathrm{Top}(S)$ permette di osservare il valore dell'elemento che verrebbe estratto da $\mathrm{Pop}(S)$ senza rimuoverlo.

Per ogni elemento π_i presente nell'input se la pila è vuota viene eseguito un push e si passa al prossimo. Se la pila non è vuota si esegue l'operazione pop fino a che l'elemento in cima alla pila non è maggiore di π_i, poi si esegue un push.
Finiti gli elementi nell'input, se necessario, si svuota completamente la pila nell'output.[8]
L'espressione $S(\pi)$ rappresenta il risultato ottenuto applicando un'iterazione di stacksort su una permutazione $\pi = \pi_1\pi_2 \dots \pi_n$.
In questo caso, sia $\pi = \pi_L n \pi_R$, con n valore massimo in π, e π_L, π_R rispettivamente suo prefisso e suffisso, vale che $S(\pi) = S(\pi_L)S(\pi_R)n$.

Algorithm 2 S - stacksort

```
S ← ∅
for i = 1 to n do
    while  S ≠ ∅ and π_i > Top(S) do
        Pop(S)
    end while
    Push(π_i, S)
end for
while  S ≠ ∅  do
    Pop(S)
end while
```

Queue sort

Queuesort utilizza una coda per ordinare una permutazione.
Si definisce una coda come un insieme su cui si possono applicare le seguenti operazioni:

$\mathrm{Enqueue}(\pi_i, Q)$ aggiunge l'elemento π_i alla coda Q

$\mathrm{Dequeue}(Q)$ estrae da Q l'elemento che è stato inserito per primo e lo aggiunge all'output

$\mathrm{Front}(Q)$ permette di osservare del prossimo valore da estrarre senza rimuoverlo dalla pila

$\mathrm{Back}(Q)$ permette di osservare l'ultimo valore inserito nella pila.

Per descrivere queuesort, è infine necessaria un'ultima operazione che non prevede di utilizzare la coda:

$\mathrm{Bypass}(\pi_i)$ il valore π_i viene aggiunto all'output senza passare dalla coda

Per ogni elemento π_i della permutazione π in input se la coda è vuota o il suo ultimo elemento è minore di π_i, si accoda π_i, altrimenti si tolgono elementi dalla coda ponendoli nell'output fino a che l'elemento in testa alla coda non è maggiore di π_i, poi si aggiunge π_i all'output. Si svuota la coda nell'output.

Algorithm 3 operatore Q - queue sort, singola iterazione

$Q \leftarrow \emptyset$
for $i = 1$ **to** n **do**
if $Q = \emptyset$ **or** $\mathrm{Back}(Q) < \pi_i$ **then**
Enqueue(π_i,Q)
else
while $\mathrm{Front}(Q) < \pi_i$ **do**
Dequeue(π_i)
end while
$\mathrm{Bypass}(\pi_i)$
end if
end for
while $Q \neq \emptyset$ **do**
Dequeue(π_i)
end while

OSSERVAZIONE *bubblesort* è un caso particolare sia di *queue sort* che di *stacksort*.
Se infatti si fissa a 1 la capacitá della pila o della coda dei rispettivi operatori il comportamento che questi assumono è quello di una cella che, scorrendo l'input, contiene sempre il massimo valore trovato, mentre gli altri vengono messi nell'output, ovvero analogo a quello di bubblesort.

LEFT-TO-RIGHT MASSIMI In una permutazione i *LTR massimi* sono gli elementi che risultano essere maggiori di ogni altro elemento che li precede. Ad esempio nella permutazione 142387596 i *LTR massimi* sono 1,4,8,9.

La nozione di *LTR massimi* risulta molto importante per lo studio di alcuni degli algoritmi di ordinamento presentati. In particolare durante l'esecuzione di *queuesort* i LTR massimi sono tutti e soli gli elementi che entrano nella coda, mentre durante l'esecuzione di *bubblesort* sono gli unici che vengono scambiati; in entrambi gli algoritmi l'ordine relativo degli altri elementi non viene alterato.

CONTENITORI POP Un caso di studio interessante è quello in cui i contenitori di stacksort e queue sort vengano sostituiti dalla loro versione POP, ovvero un contenitore con politiche di estrazione e inserimento analoghe ma quando viene eseguita un'estrazione il contenitore viene svuotato completamente. Si definiscono con questa variante, gli algoritmi **pop-stacksort** e **pop-queuesort**

CLASSI DI PATTERN DI PERMUTAZIONI

Siano α, β due sequenze di interi indichiamo con $\alpha \subseteq \beta$ che α è una sottosequenza di β, anche se non necessariamente una sottosequenza consecutiva[2].

STANDARDIZZAZIONE Un pattern classico è sempre rappresentato dalla **standardizzazione** di una permutazione.
Per una sequenza di numeri la sua standardizzazione[7] è un altra sequenza della stessa lunghezza in cui l'elemento minore della sequenza originale è stato sostituito da 1, il secondo minore con un 2, ecc.
Ad esempio, la standardizzazione di 5371 è 3241 .

Si dice che una permutazione τ contiene un pattern δ se esiste $\lambda \subseteq \tau$ la cui standardizzazione è uguale a δ, e si indica con $\delta \preceq \tau$.
Ad esempio 24153 contiene il pattern 312 perché 413 $\subseteq$ 24153 e la standardizzazione di 413 è 312.

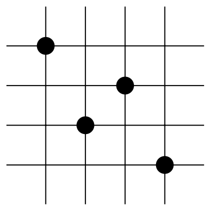
CLASSI DI PATTERN La relazione di sottopermutazione è una relazione d'ordine parziale che viene studiata con dei sottoinsiemi chiamati

pattern di classi. Ogni classe D puó essere caratterizzata dall'insieme minimo M che evita:

$$D = Av(M) = \{\beta : \mu \not\preceq \beta \forall \mu \in M\}$$

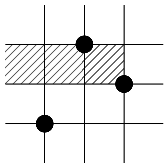
In questa sezione verranno da qui in poi introdotti brevemente altri tipi di pattern[3] attraverso degli esempi; le macro per la realizzazione delle griglie sono state prodotte dalla *Reykjavik University*[10].

PLOT, RAPPRESENTAZIONE GRAFICA DI PERMUTAZIONI Data una permutazione $\pi = \pi_1\pi_2 \dots \pi_n$ la sua rappresentazione grafica è data dall'insieme di punti (i, π_i). Questi punti possono essere rappresentati in un piano cartesiano, in particolare nella griglia sottostante, che rappresenta il pattern 4231, le righe verticali rappresentano le ascisse i (numerate a partire da 1 da quella piú a sinistra) e quelle orizzontali le ordinate, quindi i valori interi π_i (numerati a partire da 1 da quella piú in basso).

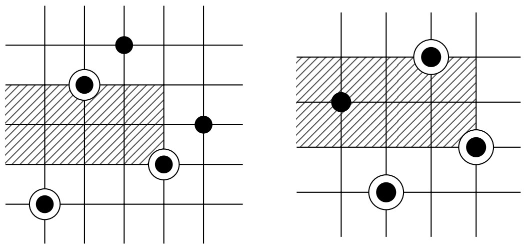


PATTERN BARRATI I pattern classici esprimono in quale relazione d'ordine devono essere gli elementi di una sequenza. In alcuni casi puó rivelarsi necessario dover esprimere informazioni ulteriori, come l'assenza di un elemento in una data posizione. Questa informazione puó essere fornita grazie all'utilizzo dei pattern barrati.
Un pattern barrato è un pattern in cui ogni numero puó essere barrato.
Un'istanza di un pattern barrato è rappresentata da un'istanza della parte non barrata del pattern che non si estenda ad un'occorrenza della parte barrata. Ad esempio una permutazione π contiene il pattern $23\bar{4}1$ se contiene degli elementi $a < b < c$ tali che $\pi = \dots b \dots c \dots a \dots$ e fra c e a non è presente un elemento maggiore di c.
Una permutazione quindi evita un pattern barrato se ad ogni occorrenza della parte non barrata del pattern corrisponde un'occorrenza della parte non barrata.

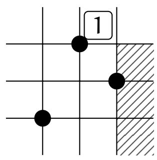
MESH PATTERN I pattern barrati possono essere rappresentati da plot come quello mostrato prima, dove l'area corrispondente all'elemento barrato viene oscurata. Si osservi il seguente pattern a titolo di esempio:



Una permutazione π di lunghezza n contiene questo pattern se ci sono 3 valori $a < b < c < n$ tali che $\pi = \dots a \dots c \dots b \dots$ e prima di b nessun valore è compreso tra b e c.
Ad esempio la permutazione 14523 (mostrata sotto a sinistra) contiene un'istanza di questo pattern mentre la permutazione 3142 (mostrata a destra) no. In entrambi i casi l'occorrenza della parte non barrata del pattern è evidenziata.



PATTERN DECORATI Un pattern decorato è un mesh-pattern in cui vengono aggiunte informazioni relative alla presenza di elementi in determinate aree. Senza dare la definizione formale descriviamo un esempio di tale pattern grazie al quale introduciamo anche la notazione che verrá utilizzata. Nel seguente pattern l'area bianca indicata con 1 indica che per avere un'istanza di questo pattern in una permutazione, all'interno dell'area deve essere presente almeno un altro elemento della permutazione:



Questo pattern è contenuto, ad esempio, nelle sequenze 1342, 12453 o 13524.

ALGORITMI DI ORDINAMENTO E PATTERN-AVOIDANCE

Per tutti gli algoritmi presentati all'inizio del capitolo sono giá note le classi di pattern che garantiscono l'ordinabilitá con una sola passata.

Operatore	**Permutazioni ordinabili con una sola passata**
Stacksort[8]	Av(231)
Queuesort[6]	Av(321)
Bubblesort[1]	Av(231,321)
Pop-stacksort[9]	Av(231,312)
Pop-queuesort[4]	Av(321,2413)

3

PROGRAMMI REALIZZATI

In questo capitolo presenteró il funzionamento di alcuni programmi che ho realizzato come supporto allo studio degli argomenti di questa tesi.

Tutto il software che verrá presentato é stato realizzato con il linguaggio Python, nel seguente ambiente: `3.11.6 (main) [GCC 13.2.1 20230801]`. Tutti gli script vengono lanciati da linea di comando e mostrano i risultati sulla console.

PERMUTASORT

Il primo programma che ho realizzato viene lanciato da linea di comando specificando come argomenti un intero n e un operatore di ordinamento X. Il suo scopo é quello di enumerare tutte le n-permutazioni ordinabili e non ordinabili con una sola passata di X, stampando anche i possibili risultati di X su n-permutazioni.

La classe `SelectorPermutations` é il core del programma, viene inizializzata passando un intero e un operatore al costruttore. Grazie alla funzione `itertools.permutations` vengono generate le n-permutazioni e , scorrendole tutte, vi si applica l'operatore X. Il risultato viene aggiunto alla lista `outcomes` e viene valutato: se é ordinato la permutazione da cui si é ottenuto viene aggiunta alla lista `sortable`, altrimenti a `unsortable`. La classe fornisce dei *getters* per le liste e non ha altri metodi, oltre al costruttore.
Dopo essere stata istanziata la classe viene usata come riferimento per i dati che ha giá calcolato nel costruttore, e non produce ulteriori risultati. Di seguito viene mostrato il listato della classe:

```
import sys
```

```python
from itertools import permutations

if __name__ == '__main__':
  from src.operators import *
  from src.utils import *
else:
  from .src.operators import *
  from .src.utils import *

class selectorPermutations:
  def __init__(self,num,op):
    if num <=0:
      print("ERROR: was expecting a positive integer but got " +str(num))
      exit()

    # initialization of lists
    self.__sortable = []
    self.__unsortable = []
    self.__outcomes = []

    # generation of permutations
    permutations_list = list(permutations(range(1,num+1)))

    for P in permutations_list:

      # applying the operator to the permutation
      op_P_ = op(P)

      # adding outcome to the list
      if op_P_ not in self.__outcomes:
        self.__outcomes.append(op_P_)

      # adding permutations to the right list
      if isIdentityPermutation(op_P_):
        self.__sortable.append(P)
      else:
        self.__unsortable.append(P)

  def getSortable(self):
    return self.__sortable

  def getUnsortable(self):
```

```
    return self.__unsortable

  def getOutcomes(self):
    return self.__outcomes
```

PATTFINDER

Questo programma ha lo scopo di selezionare tra tutte le permutazioni di una data dimensione quali contengono un dato pattern classico e quali no.

Il programma viene lanciato da linea di comando prendendo come argomenti un intero e una sequenza di interi consecutivi eventualmente non ordinati.
Il primo intero rappresenta la lunghezza delle permutazioni da scorrere e la sequenza rappresenta il pattern da ricercare.
Gli argomenti vengono passati al costruttore della classe `PatternAvoid`, l'intero n viene utilizzato per generare le n-permutazioni grazie alla funzione `permutations`. Per ogni permutazione viene controllato se contiene o no il pattern e viene inserita nella rispettiva lista: `containing` o `notcontaining`.

La classe presenta i getter per le liste prodotte e due ulteriori metodi:

`patternize` serve a *standardizzare* una sequenza, si fa riferimento alla definizione di standardizzazione (di una permutazione) presentata nel capitolo precedente

`contains` verifica che una sequenza contenga un pattern o meno, grazie alla funzione `itertools.combinations` si ottengono tutte le sottosequenze (anche non consecutive), le si standardizzano e le si confrontano con il pattern da ricercare.

```
import sys
from itertools import permutations,combinations

class PatternAvoid():
  def __init__(self,n, pattern):
    if n <= 1:
      print("ERROR: expecting a whole number greater than 1 but got: "
          + str(n))
```

```
            exit()

        self.__n = n

        # checking if the argument really is a classical pattern
        pattern_list = [int(p) for p in pattern]
        if len(pattern) != max(pattern_list):
            print("ERROR: expected a pattern but got: " + str(pattern))
            exit()
        for i in range(1, len(pattern)):
            if i not in pattern_list:
                print("ERROR: expected a pattern but got: " + str(pattern))
                print("\nA pattern of lenght n must have all the numbers from 1
                    to n, " + str(i) + " not present")
                exit()

        # generating permutations
        permutations_list = list(permutations(range(1,n+1)))

        # declaring list fields
        self.__notcontaining = []
        self.__containing = []

        for p in permutations_list:
            if not self.contains(p,pattern):
                self.__notcontaining.append(p)
            else:
                self.__containing.append(p)

    # get the standard image of the number sequence:
    def patternize(self,pi):
        output = []
        id_ = sorted(pi)
        for p in pi:
            output.append(id_.index(p)+1)
        return output

    # check if a seqence contains the given pattern
    def contains(self, seq, pattern):
        if len(seq) < len(pattern):
            return False
        subseq = combinations(seq, len(pattern))
```

```
    for s in subseq:
      if list(self.patternize(s)) == list(pattern):
        return True
    return False

  def getNotContaining(self):
    return self.__notcontaining

  def getContaining(self):
    return self.__containing
```

SCRIPT PER LA VERIFICA DEI PATTERN

Spesso si é utilizzato un ulteriore script, che si serve delle classi presentate prima, per verificare che i pattern trovati per un dato operatore siano sufficienti a definire tutto l'insieme di permutazioni non ordinabili.

Grazie a `selectorPermutations` vengono generate tutte le permutazioni non ordinabili (l'operatore e la dimensione delle permutazioni vengono passate come argomenti allo script) ed in seguito si scorre la lista di pattern che si vogliono veririficare. Per ogni pattern `PatternAvoid` genera le permutazioni che lo contengono e queste vengono rimosse dalla lista di permutazioni non ordinabili. Se alla fine la lista rimane vuota allora viene stampato a schermo che tutte le permutazioni non ordinabili contengono almeno uno dei pattern specificati, altrimenti elenca quelle rimaste.

Lo script viene lanciato da linea di comando con i seguenti argomenti (nell'ordine): l'operatore di ordinamento, la dimensione delle permutazioni, l'elenco di pattern da verificare.

Di seguito si presenta un esempio di utilizzo ed il codice dello script; l'utilizzo mostrato é in riferimento alla combinazione degli operatori stacksort e bubblesort, che verrá speigato mostrata nel dettaglio nel capitolo successivo.

```
$ python verify_patterns.py SB 5 2341 2431 3241

Generating permutations:
      found 50 unsortable permutations
```

```
Verifying 3 patterns:

Verifying pattern 2341:
      found 17 matches

Verifying pattern 2431:
      found 17 matches

Verifying pattern 3241:
      found 17 matches

These 10 permutations are not sorted by SB but not contain any
    specified pattern:
(5, 2, 4, 1, 3)
(5, 3, 1, 4, 2)
(5, 2, 3, 1, 4)
(5, 3, 4, 2, 1)
(1, 5, 3, 4, 2)
(5, 1, 3, 4, 2)
(4, 5, 2, 3, 1)
(4, 2, 3, 1, 5)
(5, 4, 2, 3, 1)
(5, 3, 4, 1, 2)

$ python verify_patterns.py SB 5 2341 2431 3241 4231

Generating permutations:
      found 50 unsortable permutations

Verifying 4 patterns:

Verifying pattern 2341:
      found 17 matches

Verifying pattern 2431:
      found 17 matches

Verifying pattern 3241:
      found 17 matches

Verifying pattern 4231:
```

```
      found 17 matches

All unsortable permutations contains some of the specified patterns
```

```
import sys
from permutasort.permutasort import *
from pattFinder.pattfinder import *

# allows to find out if the specified patterns cover all the
    permutations of lenght n
# that are not sortable by the specified sorting operator

if __name__ == '__main__':

  if len(sys.argv) < 4:
    print("ERROR; requested at least 3 arguments but got: " +
        str(sys.argv[1:]))
    exit()

  op = getOperator(sys.argv[1])
  n = int(sys.argv[2])
  av = sys.argv[3:]

  print("Generating permutations: ")
  unsortable = selectorPermutations(n,op).getUnsortable()
  result = unsortable
  print("\tfound " + str(len(unsortable)) + " unsortable
      permutations\n")

  print("Verifying "+str(len(av))+" patterns:\n")
  for p in av:
    print("Verifying pattern " + p+":")
    # from the list of unsortable premutations the ones containing
        the specified patterns get removed
    pattern = [int(char) for char in p]
    contains_p = PatternAvoid(n, pattern).getContaining()
    result = list(set(result).difference(set(contains_p)))
    print("\tfound
        "+str(len(set(contains_p).intersection(set(unsortable)))) +
        " matches\n")

  n_uns = len(result)
```

```
if(n_uns==0):
    print("All unsortable permutations contains some of the specified
        patterns\n")
else:
    print("These "+str(n_uns)+" permutations are not sorted by " +
        sys.argv[1]+" but not contain any specified pattern:")
    print(printlist(result))
```

4

COMPOSIZIONE DI OPERATORI DI ORDINAMENTO

É molto interessante studiare la combinazione dei vari operatori e le relazioni tra classi di pattern e le loro controimmagini.

controimmagini Sia X un operatore di ordinamento, la controimmagine di una certa permutazione p, secondo X, indicata con $X^{-1}(p)$ rappresenta l'insieme di tutte le permutazioni la cui immagine secondo l'operatore X è uguale a p.

$$X^{-1}(p) = \{\beta : p = X(\beta)\}$$

$A\nu(21)$ è l'insieme formato dalle sole permutazioni identitá, dato che contiene tutte le permutazioni in cui nessun elemento sia disordinato rispetto ad un altro, ovvero solo le permutazioni crescenti. Si indica con $X^{-1}(A\nu(21))$ l'insieme di tutte le permutazioni ordinabili da X.

Ad esempio, dato che è noto che l'operatore *bubblesort* ordina solo le permutazioni che non contengono pattern 231 e 321[7], vale che:

$$B^{-1}(A\nu(21)) = A\nu(231, 321)$$

composizione di operatori Dati due operatori di ordinamento X e Y, la loro composizione è indicata con $XY = X \circ Y$, quindi, ad esempio, la composizione di *Stacksort* e *Bubblesort* (in questo ordine) si indica con $SB(\pi) = (S \circ B)(\pi) = S(B(\pi))$.

algoritmi per il calcolo di controimmagini Sono giá stati prodotti alcuni algoritmi per calcolare le controimmagini di pattern secondo gli operararatori *bubblesort*[1], *stacksort*[7] e *queuesort*[9][5].
Questo ci permette, quando si combinano due operatori di ordinamento (di cui di quello applicato per secondo siano noti le classi di pattern

da evitare per l'ordinabilitá) di cercare per quali pattern l'operatore che viene applicato per primo produce permutazioni che siano ordinabili dal secondo.

UN CASO GIÁ NOTO: COMBINAZIONE DI *stacksort* CON *bubblesort* Si considera dunque la composione $SB = S \circ B$ e ci si chiede quali permutazioni possano essere ordinate da esso.

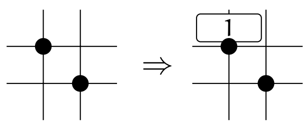
$$(SB)^{-1}Av(21) = B^{-1}S^{-1}(Av(21)) = B^{-1}(Av(231))$$

Dunque ricercando per quali permutazioni *bubblesort* evita il pattern 231 si trovano le condizioni per cui una permutazione risulta ordinabile da *SB*.
Utilizzando l'algoritmo per le controimmagini di bubblesort[1] si ottiene che:

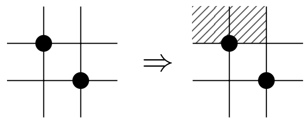
$$(SB)^{-1}(Av(21)) = Av(3241, 2341, 4231, 2431)$$

Per ottenere una controimmagine di bubblesort di un pattern classico, si applicano le seguenti regole:

- si cercano tutte le inversioni nell'immagine in esame: poiché bubblesort non produce nuove inversioni, tutte le inversioni nell'immagine dovranno giá essere presenti nella controimmagine;
- si considera una lista di **candidati**, composta da ogni pattern minimale che contiene almeno le stesse inversioni;
- per ogni candidato, si processa ogni sua inversione (b, a) nel modo seguente:
  - se (b, a) è contenuta anche nell'immagine si ha che b non è un LTR massimo o che lo è ma c'é un altro LTR massimo tra b e a, quindi si decora la coppia come segue, producendo un nuovo pattern:



  - se (b, a) è un'inversione del candidato che non è nell'immagine allora la coppia viene ordinata da una passata di bubblesort, si ha che b è l'ultimo LTR massimo prima di a, la coppia viene decorata nel modo seguente:



applicare ad un candidato le condizioni per tutte le sue inversioni (considerando che se una zona è oscurata da una condizione, non puó essere decorata da un altra) fornisce un mesh-pattern decorato che descrive il pattern contenuto nella controimmaigne.

Esempio pratico di calcolo di una controimmagine

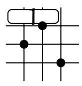
A titolo d'esempio verrá mostrato come si puó arrivare al risultato (giá noto[1]) riguardo alla combinazione $(S \circ B)$:

$$(SB)^{-1}(Av(21)) = Av(2341, 2431, 3241, 4231)$$

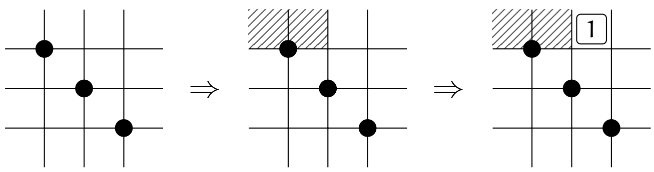
Prima di tutto si ricerca di quale classe di pattern nonché tramite quale operatore occorre calcolare le controimmagini:

$$(SB)^{-1}(Av(21)) = B^{-1}(S^{-1}(Av(21))) = B^{-1}(Av(231))$$

Dato che, come noto[8] $S^{-1}(Av(21)) = Av(231)$.
I pattern candidati ad essere controimmagini di 231 sono i pattern che contengono le coppie $(2, 1), (3, 1)$, ovvero i pattern 231,321.

231: si osserva che una condizione per l'inversione $(2, 1)$ è giá realizzata dall'elemento 3, applicando le regole descritte prima all'inversione $(3, 1)$ si ha il mesh-pattern  , ovvero i pattern 2341, 2431, 4231.

321: in questo caso si ha l'inversione $(3, 2)$ che deve essere invertita, quindi si inseriscono le aree oscurate, in seguito si osserva che, come prima, 3 realizza la condzione per l'inversione $(2, 1)$ e dunque basta aggiungere le decorazioni per l'inversione $(3, 1)$. Di seguito si mostrano i passaggi:



Si ottiene cosí il pattern 3241 che, unito agli altri fornisce esattamente il risultato atteso.

Si osservi che il software `permutasort`, presentato nel capitolo 3, puó aiutare ad arrivare alle stesse conclusioni. Infatti lanciando il programma sull'operatore SB e con $n = 4$ si ottengono i seguenti risultati:

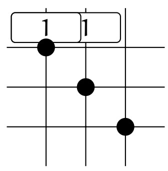
```
$ python permutasort.py 4 SB
...
The following 4 4-permutations are not sortable with the operator
SB:
(2, 3, 4, 1)
(2, 4, 3, 1)
(3, 2, 4, 1)
(4, 2, 3, 1)
...
```

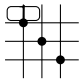
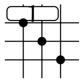
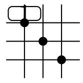
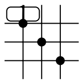
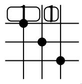
Questo tuttavia è utile come indicazione, ma non è sufficiente per essere sicuri del risultato: infatti non si puó escludere la presenza di pattern classici piú lunghi né la presenza di pattern barrati.
Per quanto quindi questo strumento si sia rivelato molto utile per avere un'idea di quali pattern ricercare, soprattutto durante le combinazioni piú complesse che presenteremo in seguito, non puó sostituire del tutto un approccio piú formale e teorico, come l'applicazione di uno degli algoritmi introdotti prima.

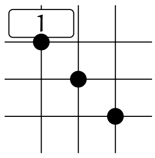
COMBINAZIONE DI QUEUESORT CON BUBBLESORT

$$(QB)^{-1}(Av(21)) = B^{-1}Q^{-1}(Av(21)) = B^{-1}(Av(321))$$

Si osserva che le inversioni in 321 sono $(3, 2), (3, 1), (2, 1)$ e l'unico pattern minimo che le contiene tutte è appunto 321, che quindi è l'unico candidato.
In questo caso a tutte le inversioni deve essere applicata la prima condizione, a parte a $(2, 1)$ perché la condzione è giá soddisfatta da 3. Il pattern decorsato che si ottiene é:



Ovvero l'unione di  e . Tuttavia questi due pattern possono essere semplificati nel primo: si nota infatti che , che si ottiene dalle condizioni per $(3,2)$ soddisfa anche le condizioni per $(3,1)$ e inoltre vale che  $\preceq$ .
Quindi le preimmagini del pattern 321 tramite bubblesort sono date da:



.

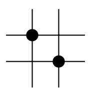
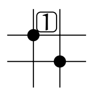
Questo mesh pattern corrisponde ai pattern classici $4321, 3421$.

$$QB^{-1}(Av(21)) = Av(4321, 3421)$$

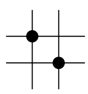
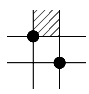
COMPOSIZIONI DI UN OPERATORE DI ORDINAMENTO CON STACKSORT

L'algoritmo per le controimmagini di *stacksort* è abbastanza simile a quello per *bubblesort*.
Anche in questo caso si osservano tutte le inversioni contenute nell'immagine in esame e se ne trae una lista di pattern minimali candidati ad essere controimmagini e li si esaminano ad uno ad uno.
Per ogni inversione (b, a) presente nel candidato:

se (b, a) è presente anche nell'immagine allora deve essere presente un elemento $c > b$ tra b e a che fa uscire b dalla pila prima che a vi entri

 $\Longrightarrow$ 

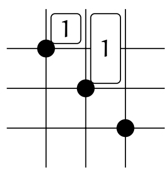
se invece (b, a) non è presente allora si puó escludere la presenza di tale elemento

 $\Longrightarrow$ 

Composizione di queuesort con stacksort

$$(QS)^{-1}(Av(21)) = S^{-1}Q^{-1}(Av(21)) = S^{-1}(Av(321))$$

Si ricercano dunque le controimmagini di 321 secondo *stacksort*.
321 è l'unico candidato, quindi sappiamo che tutte le controimmagini devono contenere il pattern 321.
L'elemento 3 deve entrare nella pila ed uscirne prima del 2, per fare ció deve esserci un elemento 3^+ maggiore di 3 tra 3 e 2. Similmente è necessario un elemento $2^+ > 2$ tra 2 e 1 per assicurare che 2 esca dalla pila prima che 1 vi entri.



Dunque le controimmagini che cerchiamo devono essere nella forma 33^+22^+1.
Se $2^+ < 3$ allora la controimmagine assume la forma del pattern 45231.
Altrimenti se $2^+ > 3$, si possono ottenere due diverse controimmagini:

$3^+ > 2^+$ genera il pattern 35241

$2^+ > 3^+$ genera il pattern 34251

$$(QS)^{-1}(Av(21)) = Av(34251, 35241, 45231)$$

Composizione di bubblesort con stacksort

$$(BS)^{-1}(Av(21)) = S^{-1}B^{-1}(Av(21)) = S^{-1}(Av(231, 321))$$

Giá dall'analisi della combinazione precedente è risultato che $S^{-1}(Av(321)) = Av(34251, 35241, 45231)$ quindi è necessario calcolare solo $S^{-1}(Av(231))$.
Quest'ultimo risultato è stato ampiamente studiato in letteratura, in quanto analogo al caso di una variante di *stacksort* che utilizza 2 pile. Il risultato che si ottiene è dunque che $S^{-1}(Av(231)) = Av(2341, 3\bar{5}241)$[7].
Si uniscono dunque i due risultati:

$$S^{-1}(Av(231)) = Av(2341, 3\bar{5}241), S^{-1}(Av(321)) = Av(34251, 35241, 45231)$$

Si osserva che $2341 \preceq 34251$, quindi 34251 non è minimo.
Inoltre i pattern $35241, 3\bar{5}241$, possono essere rappresentati dal pattern

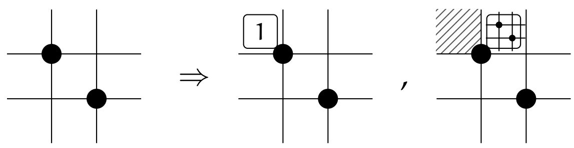
minimo 3241 che rende non minimo anche 35241.
Il risultato che si ottiene é:

$$(BS)^{-1}(Av(21)) = Av(2341, 3241, 45231)$$

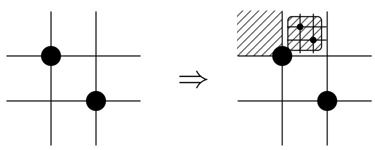
COMBINAZIONI DI UN OPERATORE DI ORDINAMENTO CON *queuesort*

Come per i due casi precedenti esiste un algoritmo per trovare controimmagini di un pattern applicando decorazioni al pattern di cui si ricerca la controimmagine.
Come per i casi precedenti si stabilisce una lista di pattern candidati, e li si esaminano ad uno ad uno, applicando le seguenti condizioni ad ogni inversione (b, a):

Se (b, a) è presente anche nell'immagine è perché non viene ordinata da una passata di queuesort, questo si verifica sicuramente quando b non è un LTR massimo (a non lo è mai, in quanto minore di b), se invece b risulta essere un LTR massimo allora deve essere presente un ulteriore inversione (d, c) tra b e a, cosí che d si accodi e c provochi l'estrazione di b prima che a possa bypassare. Queste condizioni si applicano decorando l'inversione in uno dei modi seguenti:



Se una inversione contenuta nel candidato non è presente nell'immagine significa che nessuna delle condizioni espresе prima viene soddisfatta, la decorazione che si applica rappresenta una negazione delle condizioni mostrate prima:

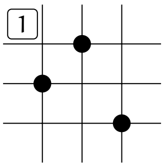
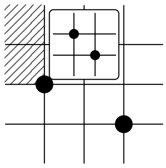
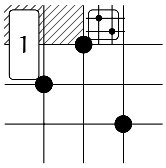
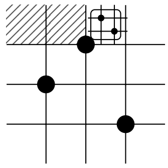


Composizione di stacksort con queuesort

$$SQ^{-1}(Av(21)) = Q^{-1}S^{-1}(Av(21)) = Q^{-1}(Av(231))$$

Si ricercano dunque le controimmaigini di 231 secondo queuesort. Gli stessi elementi che formano un pattern 231 nell'immagine possono formare nella controimmagine lo stesso pattern 231 o un pattern 321. Si esaminano i due casi separatamente.

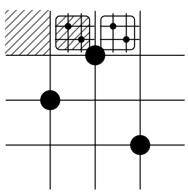
Si applicano alle inversioni di 231 le possibili condizioni espresse precedentemente, ottenendo i seguenti mesh-pattern:

(A)  , **(B)**  , **(C)**  , **(D)** 

Si osserva che i pattern (D) e (C) contengono un'istanza del pattern (B) dunque possono essere tralasciati.
Il pattern (A) corrisponde al pattern classico 4231, mentre il pattern (B) è simile ad un pattern 2431 con alcune aree oscuarate.

Esaminiamo questo caso in cui una permutazione eviti il pattern (B) piú approfonditamente: una permutazione evita questo pattern se in ogni occorrenza di un pattern 2431 l'elemento corrispondente al 2 del pattern non è un LTR massimo. Si prenda il primo LTR massimo alla sinistra dell'elemento 2 (tale elemento esiste dato che stiamo esaminando il caso di una permutazione che eviti (B)): questo entra in coda, 2 viene aggiunto all'output, e almeno uno tra l'elemento precedente e gli elementi della coppia 43 deve essere aggiunto all'output; ognuno di questi elementi è maggiore di 2 e dunque si va a formare un pattern 231 quando 1 bypassa. Se ne deduce che, ai fini di evitare la produzione di pattern 231, il fatto che una permutazione eviti il pattern (B) corrisponde all'evitare il pattern 2431, dato che la presenza di elementi nelle aree oscurate non previene la formazione di pattern 231.

Si cercano adesso le controimmagini di 231 che contengono 321, applicando le regole dell'algoritmo alle inversioni di 321.
Applicando le regole esposte, ed escludendo quelle che si contraddicono, si ottiene il seguente mesh-pattern:



che contiene un'occorenza del pattern 2431 trovato prima, quindi anche questo puó essere tralasciato in quanto giá incluso.

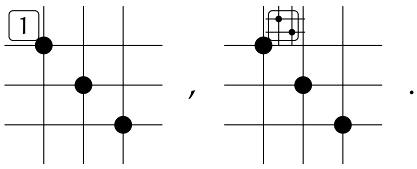
Si ottiene che:

$$SQ^{-1}(Av(21)) = Av(2431, 4231)$$

Composizione di bubblesort con queuesort

$$BQ^{-1}(Av(21)) = Q^{-1}(B^{-1}(Av(21))) = Q^{-1}(Av(231, 321))$$

L'insieme $Q^{-1}Av(231)$ è giá stato calcolato nella sezione precedente, quindi resta da cercare l'insieme $B^{-1}Av(321)$ ed eseguire l'intersezione tra i due.
321 è l'unico candidato per la ricerca delle controimmagini, ed inoltre l'elemento 3 soddisfa giá una delle condizioni per l'inversione $(2, 1)$. Applicando le condizioni alle inversioni $(3, 1), (3, 2)$, si ottengono i seguenti mesh-pattern:



Entrambi corrispondono a pattern classici, rispettivamente 4321, 35421. Si noti che il secondo contiene un'occorrenza del primo, che quindi è l'unico da considerare. Si unisce questo risultato a quelli del capitolo precedente.

$$BQ^{-1}(Av(21)) = Av(2431, 4231, 4321)$$

CONTENITORI POP

Lo studio di combinazioni di operatori che utilizzano contenitori POP si rivela piú difficile, in quando si tratta di casi non approfonditi in letteratura.
Sono tuttavia noti i pattern che rendono una permutazione non ordinabile, come presentato nel capitolo 2, ovvero:

$$S_{POP}^{-1}(Av(21)) = Av(231, 312)$$

$$Q_{POP}^{-1}(Av(21)) = Av(321, 2413)$$

Grazie a questi dati, è possibile individuare un caso di facile risoluzione: ovvero quello in cui un algoritmo che usa contenitori POP è concatenato ad un operatore regolare.

$$(X_{POP} \circ Y)(\pi) = X_{POP}(Y(\pi))) \Rightarrow (X_{POP} \circ Y)^{-1}(\pi) = Y^{-1}X_{POP}^{-1}(\pi) = Y^{-1}(Av(m))$$

Sapendo l'insieme di pattern m che rendono la permutazione non ordinabile dall'operatore POP basterá cercare le loro controimmagini secondo l'operatore regolare con i metodi usati finora, come verrá mostrato nelle sezioni seguenti.

POP QUEUESORT

L'algoritmo POP queuesort richiede una diversa analisi da POP stacksort. Esistono diverse versioni di *POP queuesort* ed in particolare ne esistono due ottimali[4]: *Min* e *Cons*.

min in questa versione l'operazione di POP viene eseguita solo se il primo elemento della coda è il successivo dell'ultimo elemento aggiunto all'output; se l'elemento in input è maggiore dell'ultimo elemento della coda (o se la coda è vuota) allora viene accodato mentre negli altri casi, se l'elemento dell'input è minore della testa della coda allora bypassa altrimenti si esegue un POP.

cons questa versione si basa sull'idea di avere sempre elementi consecutivi nella coda; è la versione che verrá adottata in questa tesi; da qui in avanti ogni riferimento a *POP-queuesort* sará riferito a *Cons*

Nell'algoritmo *Cons* si definisce una coda POP come un insieme Q, inizialmente vuoto, di interi su cui è possibile fare le seguenti operazioni:

ENQUEUE(π_i, Q): inserisce l'i-esimo elemento della permutazione π in Q, accodandolo alla coda;

POP(Q): estrae gli elementi contenuti in Q e li aggiunge all'output nello stesso ordine a cui sono stati aggiunti a Q;

BYPASS(π_i): posiziona l'i-esimo elemento di π nell'output.

Inoltre si utilizzano le seguenti funzioni:

BACK(Q): restituisce il valore dell'ultimo elemento aggiunto a Q

FRONT(Q): restituisce il valore dell'elemento in Q che vi è stato inserito per primo

Algorithm 4 Cons - POP Queuesort

```
Q ← ∅
for i = 1 to n do
    if Q = ∅ or π_i = Back(Q) + 1 then
        Enqueue(π_i)
    else
        if Front(Q) > π_i then
            Bypass(π_i)
        else
            Pop(Q)
            Enqueue(π_i)
        end if
    end if
end for
if Q ≠ ∅ then
    Pop(Q)
end if
```

La dimostrazione che *Cons* (cosí come *Min*) sia un algoritmo ottimale nella classe degli algoritmi *POP-queuesort* si ha dal fatto che esso ordina tutte e sole le sequenze dell'insieme Av(321, 2413), che sono esattamente tutte e sole le permutazioni ordinabili con una POP queue[4].

Composizione di POP-stacksort con stacksort

$$(S_{POP} \circ S)^{-1}(Av(21)) = S^{-1}(Av(231, 312))$$

Si è giá calcolato che $S^{-1}(Av(231)) = Av(2341, 3\bar{5}241)$, quindi manca da calcolare le controimmagini di 312.
I pattern candidati che possono generare 312 sono quelli che contengono le coppie $(3,1),(3,2)$, ovvero $321, 312$. Perché gli elementi di un pattern 321 nella controimmagine generino un pattern 312 tramite *stacksort* è necessario che 3 entri in pila e ne esca prima che 2 vi entri, deve quindi essere presente un valore 4 posizionato tra 3 e 2 che provochi l'uscita di 3, ovvero il pattern 3421.
Se gli elementi del pattern 312 nell'immagine formano lo stesso pattern anche nella controimmagine allora anche in questo caso bisogna assicurarsi che 3 venga estratto dalla pila prima di 1, si ha quindi il pattern

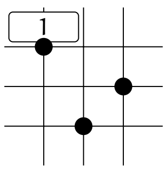
3412.
Si uniscono tutti i risultati:

$$(S_{POP}S)^{-1}(Av(21)) = Av(2341, 3412, 3421, 3\bar{5}241)$$

Composizione di POP-stacksort con bubblesort

$$(S_{POP} \circ B)^{-1}(Av(21)) = B^{-1}(Av(231, 312))$$

Anche in questo caso la controimmagine di 231 è stata calcolata: $B^{-1}(Av(231)) = (S \circ B)^{-1}(Av(21)) = Av(2341, 2431, 3241, 4231)$.
Si ricerca dunque le preimmaigini di 312 secondo bubblesort.
Gli elementi del pattern 312 nell'immagine possono solo formare un pattern 312 o 321 nella controimmagine.
Si puó osservare come gli stessi elementi di un pattern 321 non possono divenire un 312. La presenza dell'elemento maggiore del pattern all'inizio di essa evita l'ordinamento degli elementi successivi.
Perché un pattern 312 rimanga invariato è sufficiente la presenza di un elemento maggiore di tutto il pattern in posizione tale da evitare l'ordinamento di 3 con qualsiasi elemento, applicando le regole descritte prima per *bubblesort*:



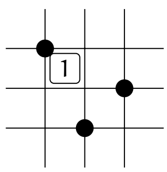
$= 4312, 3412$

$$(S_{POP}B)^{-1}(Av(21)) = Av(2341, 2431, 3241, 3412, 4231, 4312)$$

Composizione di POP-stacksort con queuesort

$$(S_{POP} \circ Q)^{-1}(Av(21)) = Q^{-1}(Av(231, 312))$$
$$Q^{-1}(Av(231) = Av(4231, 2431)$$

Possibili controimmagini di $312 : 312, 321$.
Se la controimmagine contiene 312 bisogna far si che 3 effettui un bypass o che esca prima di 1, in ogni caso queste condizioni sono soddisfatte dal seguente mesh-pattern, 4312:



$$(S_{POP}Q)^{-1}(Av(21)) = Av(2431, 4231, 4312)$$

Composizione di POP-queuesort con stacksort

$$(Q_{POP}S)^{-1}(Av(21)) = S^{-1}(Q_{POP}^{-1}(Av(21))) = S^{-1}(Av(321, 2413))$$

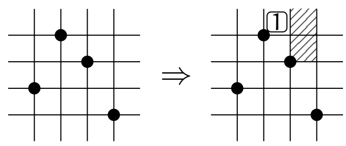
Come in molti altri casi la controimmagine di 321 è stata calcolata:

$$S^{-1}(Av(321)) = (QS)^{-1}(Av(21)) = Av(34251, 35241, 45231)$$

Si ricerca dunque le controimmagini di 2413 secondo stacksort, applicando lo stesso algoritmo dei casi precedenti.
Le inversioni di 2413 sono $(2, 1), (4, 1), (4, 3)$, che definiscono l'insieme di candidati 2413, 2431, 4213, 4231.

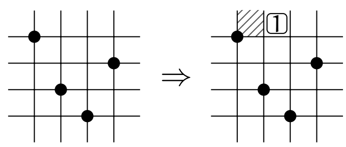
2431 I primi due elementi del pattern sono ordinati fra loro, dunque per lasciare il pattern invariato è sufficiente evitare che le coppie 41 e 43 vengano ordinate. Come giá visto questo si ottiene posizionando in mezzo ai due elementi della coppia un elemento maggiore di entrambi. Si ottiene cosí il pattern 24513.

2431 Le inversioni di questa controimmagine sono quelle giá individute nell'immagine piú $(3, 1)$. Quindi applicando le condizioni per le controimmagini si ottiene il seguente mesh-pattern:



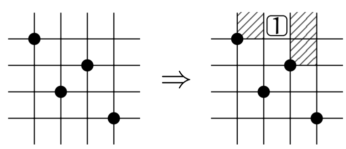
La controimmagine di 2413 che contiene 2431 è quindi 24531.

4213



La controimmagine di 2413 che contiene 4213 è 42513.

4231



La controimmagine di 4213 è 42531.

Unendo tutti i risultati si ottiene:

$(Q_{POP} \circ S)^{-1}(Av(21)) = Av(24513, 24531, 34251, 35241, 42513, 42531, 45231)$

Composizione di POP-queuesort con bubblesort

$$(Q_{POP}B)^{-1}(Av(21)) = B^{-1}Q_{POP}^{-1}(Av(21)) = B^{-1}(Av(321, 2413))$$

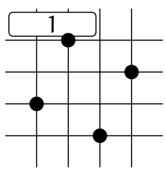
La classe di pattern $B^{-1}(Av(321))$ è giá stata calcolata:

$$B^{-1}(Av(321)) = QB^{-1}(Av(21)) = Av(3421, 4321)$$

Si ricerca adesso la classe di pattern corrispondente a $B^{-1}(Av(2413))$. La lista dei candidati è la seguente: 2413, 2431, 4213, 4231, 4321.

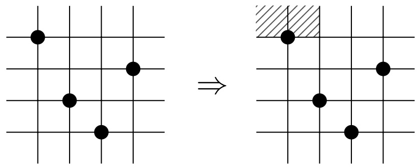
OSSERVAZIONE: i pattern 2431, 4231, 4321 contengono l'inversione $(3, 1)$ che non presente nel pattern dell'immagine. Questa specifica inversione è in posizione tale che nessuno dei due elementi sia un LTR massimo, e dunque l'inversione non possa essere ordinata. In questi 3 pattern infatti, l'elemento 4 si trova all'interno dell'area oscurata che si dovrebbe aggiungere durante l'applicazione dell'algortimo per le preimmagini di bubblesort all'inversione $(3, 1)$. Ne consegue che i pattern 2431, 4231, 4321, pur essendo nella lista di candidati non possono generare il pattern 2413.
Si ricercano ora le preimmagini date dagli altri candidati:

2413: Si osserva che 4 soddisfa giá la condizione per l'inversione $(2, 1)$, dunque si prende l'intersezione tra le condizioni per le inversioni $(4, 3), (4, 1)$ ottenendo il seguente pattern decorato:

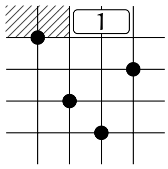


Da questo si ottengono i pattern classici 52413, 25413, 24513.

4213: L'inversione $(4, 2)$ deve essere ordinata, in quanto non presente in 2413.



Si osserva poi che 4 soddsfa giá le condizioni per l'inversione $(2,1)$, mentre applicando le condizioni per le altre inversioni, tenendo conto dell'area oscurata si ottiene:



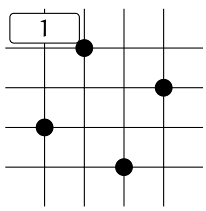
Corrispondente ai pattern classici 42513, 42153.
Si uniscono i risultati ottenuti.

$$(Q_{POP}B)^{-1}(Av(21)) = Av(3421, 4321, 24513, 25413, 42153, 42513, 52413)$$

Composizione di POP-queuesort con queuesort

$$(Q_{POP}Q)^{-1}(Av(21)) = Q^{-1}(321, 2413)$$

La controimmagine $Q^{-1}(Av(321))$ è giá stata trovato nel caso $B \circ Q$, e vale che $Q^{-1}(Av(321)) = Av(4321)$.
Se si applica le regole per la ricerca di preimamgini di queuesort a 2413 si ottiene:



Quindi si ottiene che $(Q_{POP} \circ Q)^{-1}(Av(21)) = Av(4321, 25413, 52413)$.

PRESENTAZIONE DELL'ALGORITMO PER LA RICERCA DI CONTROIMMAGINI SECONDO L'OPERATORE POPSTACKSORT

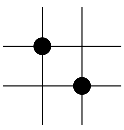
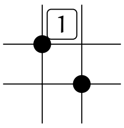
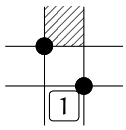
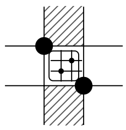
Si ricercano le condizioni che un candidato deve soddisfare per produrre una voluta immagine. Come negli altri casi, si esaminano le inversioni nel candidato, per ogni inversione (b, a) che non viene ordinata da una passata di POPstacksort si verifica una delle sequenti condizioni:

A) un elemento $c > b$ è presente tra b e a, questo provoca il POP prima che l'algoritmo arrivi ad esaminare a;

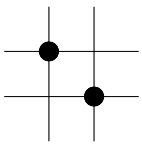
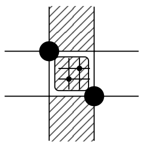
B) un elemento $a^- < a$ (e nessun elemento $c > b$) è presente tra b e a, quindi a^- viene inserito in coda ed è a a provocare il POP;

C) tra i due elementi ma non c'é nessun elemento minore di a o maggiore b ma c'é una coppia ordinata, ovvero esistono $b^-, a^+ : a < a^+ < b^- < b$ e $b \ldots a^+ \ldots b^- \ldots a$ è una sottosequenza nel candidato.

Quindi per trovare una controimmagine secondo POPstacksort occorre applicare le seguenti decorazioni, per ogni inversione che debba restare tale, al mesh-pattern del candidato:

 $\Rightarrow$ A)  , B)  , C) 

Se invece ci si vuole assicurare che una inversione venga ordinata, la decorazione da apllicare consiste semplicemente nella "negazione" di tutte quelle espresse finora:

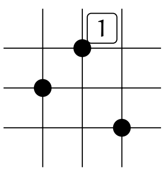
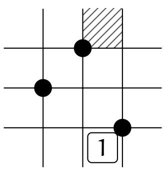
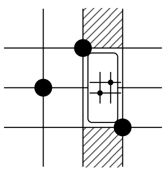
 $\Rightarrow$ 

Composizione di stacksort con POP-stacksort

$$(SS_{POP})^{-1}(Av(21)) = S^{-1}_{POP}(S^{-1}(Av(21))) = S^{-1}_{POP}(Av(231))$$

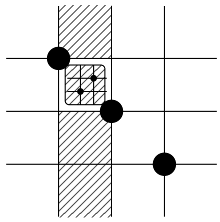
Si applicano le condizioni per le controimmagini ai due candidati: 231 e 321.

Nel caso di 231, le inversioni sono $(2,1),(3,1)$. Si osserva che 3 giá soddisdfa la condzione A per la coppia $(2,1)$, quindi resta da applicare le condizioni alla coppia $(3,1)$:

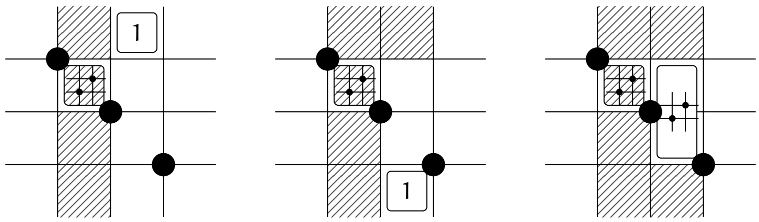
 ,  , 

Dalla condizione A si ottiene il pattern 2341, dalla B il 3412 dalla condizione C si ottengono, considerando la posizione relativa degli elementi nella coppia aggiuntiva rispetto all'elemento 2, i pattern 25341, 35241, 45231.

Se gli elementi del pattern 231 nell'immagine invece formano un pattern 321 nella controimmagine, allora si applica la condizione per ordinare una coppia alla coppia $(3, 2)$:



Si applicano adesso le condizioni per la coppia $(3, 1)$ che ci si aspetta resti disordinata:



Si osservi come applicando le condzioni A e B per la coppia $(3, 1)$ queste siano giá soddisfatte per la coppia $(2, 1)$. Applicando la condizione C la coppia puó essere costruita utilizzando l'elemento 2 come membro minore all'interno della coppia, e questo soddisfa anche la condizione A per la coppia $(2, 1)$. Si ottengono dunque i pattern 3241, 4312, 4231.

Si osserva che, tra i risultati ottenuti prima, alcuni risultano non necessari: $2341 \preceq 25341, 3241 \preceq 35241, 4231 \preceq 45231$. Si semplificano e si ottiene il seguente risultato:

$$(SS_{POP})^{-1}(A\nu(21)) = A\nu(2341, 3241, 3412, 4231, 4312)$$

Composizione di queuesort con POP-stacksort

$$QS_{POP}{}^{-1}(A\nu(21)) = S_{POP}^{-1}(Q^{-1}(A\nu(21))) = S_{POP}^{-1}(A\nu(321))$$

Si applicano anche stavolta le condizioni per la ricerca di controimmagini al pattern 321. Essendo 321 una sequenza decrementale, non ci sono altri candidati; le sue inversioni sono $(3,2),(3,1),(2,1)$ ed è necessario applicare tutte le combinazioni possibili di condizioni A,B,C a queste coppie.

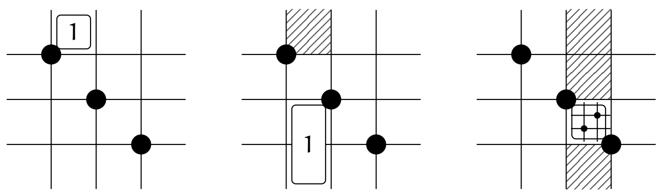
Una volta applicate le diverse condizioni alla coppia $(3,2)$ è evidente che queste soddisfano anche le condizioni per la coppia $(3,1)$:

applicando la condizione A e ponendo quindi un elemento maggiore di 3 tra 3 e 2, allora lo stesso elemento rappresenta un valore maggiore di 3 posizionato tra 3 e 1 (condzione A per la coppia $(3,1)$)

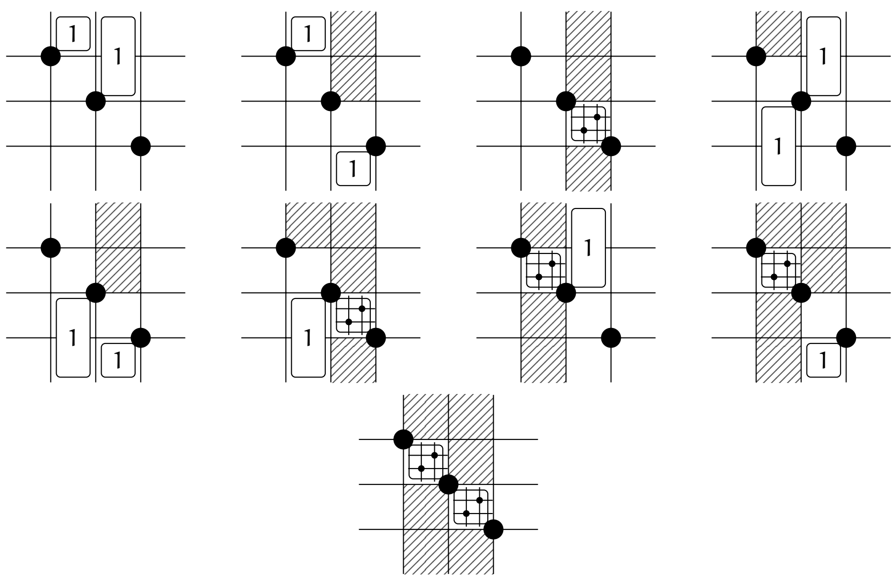
se si applica la condizione B, ponendo un valore $2^- < 2$ tra 2 e 3, allora o si ha che $2^- < 1$, che soddisfa la condizione B per la coppia $(3,1)$, o che $1 < 2^- < 2$, e allora la coppia $(2^-,2)$ è una coppia ordinata tra 3 e 1 (condizione C per ls coppia $(3,1)$)

nel caso, infine, in cui si applichi la condizione C e si ponga quindi una coppia ordinata tra 3 e 2 allora la stessa coppia soddisfa la condizione C per la coppia $(3,1)$.

In sintesi, i seguenti pattern soddisfano le condzioni da imporre alle coppie $(3,2),(3,1)$:



Una volta applicate le condizioni anche alla coppia $(2,1)$ si ottengono i seguenti 9 mesh-pattern, da cui si ottengono 22 pattern classici:



Scrivendo come pattern classici quelli che si ottengono da questi mesh-patterns ed eliminando quelli ridondanti, si ottiene il seguente risultato:

$$\begin{aligned}QS_{POP}(Av(21)) = Av(&34251, 35241, 41352, 42351, 45231,\\ &45312, 51342, 51423, 52341, 52413, 53412)\end{aligned}$$

Composizione di bubblesort con POP-stacksort

$$(B \circ S_{POP})^{-1}(Av(21)) = S_{POP}^{-1}(B^{-1}(Av(21))) = S_{POP}^{-1}(Av(231, 321))$$

Il l'insieme minimo di pattern per l'ordinabilitá di permutazione con la composzione bubblesort, POPstacksort è l'unione degli insiemi dei due risultati precedenti, stacksort e queuesort.
Si nota, tuttavia che ogni pattern contenuto nell'insieme ottenuto dalla combinazione con queuesort contiene **almeno** un pattern tra quelli ottenuti da stacksort. Nello specifico, a titolo esemplificativo:

$$3241 \preceq 34251, 35241, 41352, 42351$$

$$3412 \preceq 53412$$

$$4231 \preceq 45231, 51342, 52341, 52413$$

$$4312 \preceq 45312, 51423$$

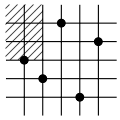
Dunque l'insieme minimo di pattern che caratterizza le permutazioni ordinabili da $B \circ S_{POP}$ è lo stesso di $S \circ S_{POP}$.

$$(BS_{POP})^{-1}(Av(21)) = Av(2341, 3241, 3412, 4231, 4312)$$

NOTA: Non si ha a disposizione un algoritmo per la ricerca di controimmagini di pattern tramite *Cons*. Per questo motivo le classi di pattern ordinabili dalle combinazioni mostrate di seguito, ovvero quelle in cui si compone POP-queuesort ad un altro algoritmo di ordinamento, verrano ricercate con un metodo diverso da quello adottato finora: si presenteranno delle congetture su quali classi di pattern che si ritengono possano rispondere alle nostre condizioni, e si verificherá che l'insieme di permutazioni ordinabili e l'insieme di permutazioni contenute nella classe di pattern coincidano.

Composizione di stacksort con POP-queuesort

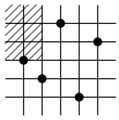
Grazie a vari risultati ottenuti applicando i software mostrati nel capitolo 3, è stato possibile formulare la seguente congettura:

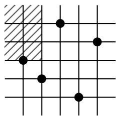
$$SQ_{POP}^{-1}Av(21) = Av(2431, 4231, 23514,$$  $$)$$

Questo equivale a cercare le controimmagini del pattern 231 secondo *Cons*, dato che:

$$SQ_{POP}^{-1}Av(21) = Q_{POP}^{-1}(S^{-1}(Av(21))) = Q_{POP}^{-1}(Av(231))$$

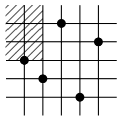
Per dimostrare la congettura occorre dimostrare la doppia inclusione:

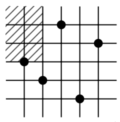
(A) $Cons^{-1}Av(231) \subseteq Av(2431, 4231, 23514,$  $)$

(B) $Av(2431, 4231, 23514,$  $) \subseteq Cons^{-1}Av(231)$

NOTA: Per tutta la dimostrazione i valori che formano i pattern saranno indicati da a, b, c, d ed e (quando presente), n indicherá la lunghezza della permutazione π e si intende $1 \leqslant a < b < c < d < e \leqslant n$.

OSSERVAZIONE: Sia π una permutazione in input all'algoritmo *Cons*, tutti gli elementi di π che non sono LTR massimi effettuano un bypass. Un LTR massimo puó accodarsi se e solo se è il consecutivo dell'elemento in fondo alla coda e provoca un *pop* in tutti gli altri casi.

(A) $Cons^{-1}Av(231) \subseteq Av(2431, 4231, 23514,$  $)$

Dimostrazione. Dobbiamo dimostrare che tutte le permutazioni la cui immagine evita il pattern 231 evitano i pattern 2431, 4231, 23514, , ovvero che l'immagine di ogni permutazione che contiene almeno uno di questi pattern contiene un pattern 231.

2431

$$\pi = \ldots b \ldots d \ldots c \ldots a \ldots$$

Le non inversioni (b, d) e (b, c) saranno presenti anche nell'immagine e inoltre c, a, non essendo LTR massimi, effettuano un bypass. Si osserva che se b effettua un bypass si ha necessariamente un pattern 231 nella sottosequnze bca. Se b entra in coda allora d provoca necessariamentee un pop (non puó essere accodato alla stessa coda che contiene b, perché la coda è consecutiva e c, che è un valore intermedio tra b e d, è piú avanti nell'input) e l'immagine contiene la sottoseqenza bca.

4231

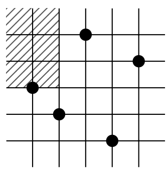
$$\pi = \ldots d \ldots b \ldots c \ldots a \ldots$$

Nessuno degli elementi che formano la sottosequenza bca contenuta in π è un LTR massimo, quindi bypassano tutti e la sequanza è contenuta anche in $Cons(\pi)$.

23514

$$\pi = \ldots b \ldots c \ldots e \ldots a \ldots d \ldots$$

Le non inversioni sono presenti anche nell'immagine, in particolare la non inversione (b, c), inoltre a non è un LTR massimo quindi effetua un bypass. Si vuole mostrare che la non inversione (b, c) viene sempre inserita nell'output prima di a. Se b, c effettuano un bypass si ha sicuramente bca nell'immagine. Se entrambi entrano in coda almeno un pop viene provocato da e prima che si arrivi ad a. Anche se b effettua un bypass e c entra in coda quest'ultimo viene sicuramente aggiunto all'output prima di a, eventualmente da e.



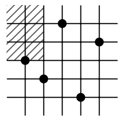
Una permutazione che contiene occorrenza di questo pattern ha la forma:

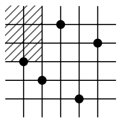
$$\pi = \ldots c \ldots b \ldots e \ldots a \ldots d \ldots$$

dove nessun valore precedente a b è maggiore di c.

Si osserva che b, a effettuano un bypass. c è un LTR massimo, quindi sia che provochi un pop o meno, verrá inserito in coda, dopodiché si

verifica il bypass di b, e c viene sicuramente estratto dalla coda prima di arrivare ad a (eventualmente è e a provocare il pop). Si puó escludere che c esca dalla coda prima che b effettui un bypass perché prima di b nessun elemento è maggiore di c.
Si forma cosí la sottosequenza bca. □

(B) $Av(2431, 4231, 23514,$  $) \subseteq Cons^{-1}Av(231)$

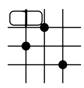
Dimostrazione. Dobbiamo dimostrare che tutte le permutazioni che evitano i pattern $2431, 4231, 23514,$  sono contenute anche nell'insieme $Cons^{-1}Av(231)$, ovvero si dimostrerá che ogni permutazione π tale che $231 \preceq Cons(\pi)$ contiene uno dei pattern elencati.
Si osserva che se 3 elementi nell'immagine $Cons(\pi)$ formano un pattern 231, quegli stessi elementi devono formare un pattern 231 o 321 nella preimmagine, dato che *Cons* non produce nuove inversioni. Si analizzano questi due casi separatamente.

$231 \preceq \pi$:

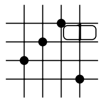
$$\pi = \dots b \dots c \dots a \dots$$

$$Cons(\pi) = \dots b \dots c \dots a \dots$$

La non inversione (b, c) resta anche nell'imamgine. Si vuole definire l'ordine relativo degli altri elementi di π rispetto a a, b, c perché le inversioni $(b, a), (c, a)$ non vengano rimosse da *Cons*.

Se la forma di π è tale che b entra in coda, per mentenere la sottosequenza bca è necessario che c non sia un LTR massimo. Questa condizione è soddisfatta se π contiene un'occorrenza del mesh-pattern , ovvero se contiene uno dei pattern classici 4231 o 2431.

Se la forma di π è invece tale per cui c entri in coda, devono essere presenti degli elementi in π che provocano un pop prima che a possa bypassare (si noti come queste considerazioni si applicano sia nel caso in cui b sia in coda con c al momento del pop, sia in quelli in cui b è giá stato inserito nell'output). Per assicurare il pop è sono necessari due nuovi elementi: e, d dove e deve essere posizionato tra c e a mentre d

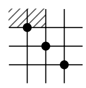
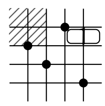
dopo di e. L'elemento e assicura il pop se è maggiore della testa della coda (se è minore tutti gli elementi del pattern effettuano un bypass) ma non puó accodarsi, in quanto d non puó ancora essere in coda e quindi e non puó essere consecutivo alla coda. Queste condizioni sono soddisfatte dal mesh-pattern , quindi si deve avere che π contenga almeno uno tra i pattern classici 23541,23514. Il primo dei due pattern classici viene ignorato, in quanto contiene uno dei pattern giá trovati prima:$2431 \preceq 23541$.

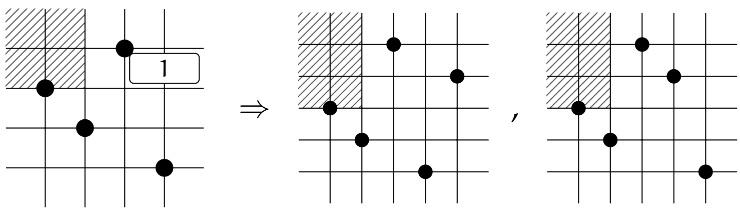
$321 \preceq \pi$:

$$\pi = \ldots c \ldots b \ldots a \ldots$$

$$\mathrm{Cons}(\pi) = \ldots b \ldots c \ldots a \ldots$$

Ci chiediamo quale deve essere l'ordine relativo degli altri elementi di π rispetto a a, b, c per ottenere che l'inversione $(3, 2)$ venga ordinata mentre le altre restino invariate.

Per ordinare (b, c), c deve essere un LTR massimo, in modo che possa entrare nella coda, e non deve verificarsi nessun pop tra c e b. Queste condizioni sono rappresentate dalle aree oscurate nel mesh pattern . Per forzare un pop tra c e a è necessario introdurre due elementi d, e come nel caso precedente: si ottiene cosí il mesh-pattern , la cui parte decorata permette rappresenta l'unione delle possibili posizioni del valore d nei due successivi mesh-pattern mostrati:



Degli ultimi due mesh pattern mostrati, si ignora il secondo in quanto contiene un'occorrenza del pattern 2431, giá trovato in precedenza, nella sottosequenza $beda$. $\square$

Composizione di queuesort con POP-queuesort

Similmente a come fatto per la sezione precedente, si vuole adesso dimostrare il seguente risultato:

$$QQ_{POP}^{-1}Av(21) = Av(4321, 35214, 35241)$$

Questo equivale a cercare le controimmagini del pattern 321 secondo *Cons*, dato che:

$$QQ_{POP}^{-1}Av(21) = Q_{POP}^{-1}(Q^{-1}(Av(21))) = Q_{POP}^{-1}(Av(321))$$

Per dimostrare la congettura occorre dimostrare la doppia inclusione:

(A) $Cons^{-1}Av(321) \subseteq Av(4321, 35214, 35241)$

(B) $Av(4321, 35214, 35241) \subseteq Cons^{-1}Av(321)$

NOTA: Si adotta la stessa dicitura, per gli elementi che formano i pattern, adottata nella precedente dimsotrazione

$$\textbf{(A)}\ Cons^{-1}Av(321) \subseteq Av(4321, 35214, 35241)$$

Dimostrazione. Dobbiamo dimostrare che tutte le permutazioni la cui immagine evita il pattern 321 evitano i pattern 4321, 35214, 35241.
Si procede a dimostrare che l'immagine di ogni permutazione che contiene almeno uno di questi pattern contiene un pattern 321.

4321:

$$\pi = \dots d \dots c \dots b \dots a \dots$$

Nessun memebro della sottosequenza cba è un LTR massimo, quindi effettuano tutti un bypass e vengono inseriti nell'output nello stesso ordine, formando un pattern 321.

35214:

$$\pi = \dots c \dots e \dots b \dots a \dots d \dots$$

Gli elementi b, a non sono LTR massimi e quindi effettuano un bypass. Si vuole dimostrare che l'elemento c viene sempre inserito nell'output prima dell'inversione (b, a). Questo è evidente quando c effettua un bypass. Se invece c entra in coda, anche se in seguito ad un pop, la testa della coda è sicuramente minore di e, che quindi deve necessariamente causare un pop (non puó accodarsi dato che c è in coda e d è ancora nell'input), ponendo c nell'output prima di b.

35241: In questo caso valgono le stesse osservazioni del caso precedente. □

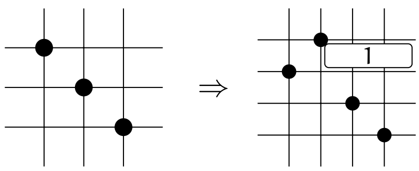
$$\textbf{(A)}\ Av(4321, 35214, 35241) \subseteq Cons^{-1}Av(321)$$

Dimostrazione. Dobbiamo dimostrare che se una permutazione evita i pattern 4321, 35214, 35241, la sua immagine evita il pattern 321. Si procede a dimostrare che per avere un pattern 321 nell'immagine $Cons(\pi)$ è necessario che π contenga almeno uno tra i pattern 4321, 35214, 35241.

Poiché *Cons* non produce nuove inversioni gli elementi che c, b, a che formano il pattern 321 nell'immagine devono essere nello stesso ordine relativo anche nella controimmagine.

Se nessuno degli elementi della sottosequenza cba è un LTR massimo allora tutti e 3 gli elementi effettuano un bypass e il pattern 321 è preservato. Questo si verifica quando cba sono i 3 elementi minori di un'occorenza di un pattern 4321.

Se c invece entra in coda, anche se in seguito ad un pop, deve avvenire un pop prima che b possa effettuare un bypass. Questo si verifica quando un elemento maggiore di e è posto tra c e b ed un ulteriore elemento d è successivo ad e (come mostrato anche nel caso precedente).

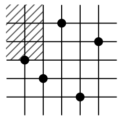


Una permutazione contiene un'occorrenza del pattern decorato cosí ottenuto se contiene uno dei seguenti pattern classici: 35421, 35241, 35214. Di questi il primo viene escluso perché contiene a sua volta un'occorrenza del pattern 4321, giá trovato prima. □

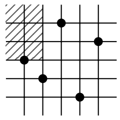
Composizione di bubblesort con POP-queuesort

$$(BQ_{POP})^{-1}(Av(21)) = Q_{POP}^{-1}(B^{-1}(Av(21))) = Q_{POP}^{-1}(Av(231, 321))$$

La classe di pattern ordinabile dalla composizione di bubblesort con Cons è data dall'unione dei risultati ottenuti per stacksort e queuesort.

$$(\mathrm{BQ_{POP}})^{-1}(Av(21)) = Av(2431, 4231, 4321, 23514, 35214, 35241,$$  $$)$$

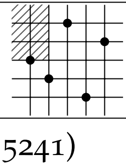
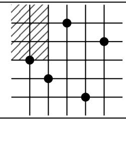
Si osserva che l'insieme non è minimo, in quanto alcuni pattern contengono istanze di altri: $2413 \preceq 35214, 4231 \preceq 35241$. Eliminati i pattern superflui, si ottiene:

$$(\mathrm{BQ_{POP}})^{-1}(Av(21)) = Av(2431, 4231, 4321, 23514,$$  $$)$$

5

CONCLUSIONI

[scrivere introduzione per le tabelle]

Composizione	**Permutazioni ordinabili con una sola passata**
$S \circ B$	Av(2341, 2431, 3241, 4231)
$Q \circ B$	Av(3421, 4321)
$Q \circ S$	Av(34251, 35241, 45231)
$B \circ S$	Av(2341, 3241, 45231)
$S \circ Q$	Av(2431, 4231)
$B \circ Q$	Av(2431, 4231, 4321)
$S_{\text{POP}} \circ S$	Av(2341, 3412, 3421, $3\bar{5}241$)
$S_{\text{POP}} \circ Q$	Av(2431, 4231, 4312)
$S_{\text{POP}} \circ B$	Av(2341, 2431, 3241, 3412, 4231, 4312)
$Q_{\text{POP}} \circ S$	Av(24513, 24531, 34251, 35241, 42513, 42531, 45231)
$Q_{\text{POP}} \circ Q$	Av(4321, 25413, 52413)
$Q_{\text{POP}} \circ B$	Av(3421, 4321, 24513, 25413, 42153, 42513, 52413)
$S \circ S_{\text{POP}}$	Av(2341, 3241, 3412, 4231, 4312)
$Q \circ S_{\text{POP}}$	Av(34251, 35241, 41352, 42351, 45231, 45312, 51342, 51423, 52341, 52413, 53412)
$B \circ S_{\text{POP}}$	Av(2341, 3241, 3412, 4231, 4312)
$S \circ Q_{\text{POP}}$	Av(2431, 4231, 23514, )
$Q \circ Q_{\text{POP}}$	Av(4321, 35214, 35241)
$B \circ Q_{\text{POP}}$	Av(2431, 4231, 4321, 23514, )

	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$	$n=9$	$n=10$
$S \circ B$	1	2	6	20	70	252	924	3432	12870	48620
$Q \circ B$	1	2	6	22	90	394	1806	8558	41586	206098
$Q \circ S$	1	2	6	24	117	652	3988	26112	180126	1295090
$B \circ S$	1	2	6	22	89	380	1678	7584	34875	162560
$S \circ Q$	1	2	6	22	90	394	1806	8558	41586	206098
$B \circ Q$	1	2	6	21	79	311	1265	5275	22431	96900
$S_{POP} \circ S$	1	2	6	20	72	272	1064	4272	17504	72896
$S_{POP} \circ Q$	1	2	6	21	79	311	1265	5275	22431	96900
$S_{POP} \circ B$	1	2	6	18	54	162	486	1458	4374	13122
$Q_{POP} \circ S$	1	2	6	24	113	578	3094	17056	96093	550886
$Q_{POP} \circ Q$	1	2	6	23	101	480	2400	12434	66142	359112
$Q_{POP} \circ B$	1	2	6	22	86	342	1366	5462	21846	87382
$S \circ S_{POP}$	1	2	6	19	63	218	778	2843	10587	40033
$Q \circ S_{POP}$	1	2	6	24	109	522	2574	12964	66426	345300
$B \circ S_{POP}$	1	2	6	19	60	189	595	1873	5896	18560
$S \circ Q_{POP}$	1	2	6	22	88	368	1580	6904	30556	136582
$Q \circ Q_{POP}$	1	2	6	23	101	480	2400	12434	66142	359112
$B \circ Q_{POP}$	1	2	6	21	76	276	998	3589	12845	45803

BIBLIOGRAFIA

[1] Michael H Albert, Mike D Atkinson, Mathilde Bouvel, Anders Claesson, and Mark Dukes. On the inverse image of pattern classes under bubble sort. *arXiv preprint arXiv:1008.5299*, 2010. (Cited on pages 12, 21, 22, and 23.)

[2] Mathilde Bouvel, Lapo Cioni, and Luca Ferrari. Preimages under the bubblesort operator. *arXiv preprint arXiv:2204.12936*, 2022. (Cited on page 9.)

[3] Petter Brändén and Anders Claesson. Mesh patterns and the expansion of permutation statistics as sums of permutation patterns. *arXiv preprint arXiv:1102.4226*, 2011. (Cited on page 10.)

[4] Lapo Cioni et al. Sorting with a popqueue. (Cited on pages 12, 30, and 31.)

[5] Lapo Cioni and Luca Ferrari. Characterization and enumeration of preimages under the queuesort algorithm. In *Extended Abstracts EuroComb 2021: European Conference on Combinatorics, Graph Theory and Applications*, pages 234–240. Springer, 2021. (Cited on page 21.)

[6] Lapo Cioni and Luca Ferrari. Preimages under the queuesort algorithm. *Discrete Mathematics*, 344(11):112561, 2021. (Cited on page 12.)

[7] Anders Claesson and Henning Ulfarsson. Sorting and preimages of pattern classes. *Discrete Mathematics & Theoretical Computer Science*, (Proceedings), 2012. (Cited on pages 9, 21, and 26.)

[8] CONG HAN LIM. Brief introduction on stack sorting. (Cited on pages 7, 12, and 23.)

[9] Hjalti Magnússon. Sorting operators and their preimages. *Computer Science*, 2013. (Cited on pages 12 and 21.)

[10] HENNING Ú'LFARSSON. Ultra-quick tutorial for generalized pattern macros. *http://staff.ru.is/henningu/notes/patternmacros/patternmacros.tex*. (Cited on page 10.)