lezione 4

Pile e Code

Collezioni

Definiamo una collezione come un gruppo di oggetti, detti gli elementi della collezione.

Durante il corso definiremo delle tecniche di base per gestire in modo efficiente collezioni di oggetti.

Utilizzeremo le nozioni di:
-Tipo di dato
-Struttura dati

Tipi di dato e Strutture dati

Dato: valore che un elemento può assumere

Tipo di dato: può essere definito come una coppia (valori, operazioni). Nel caso delle collezioni:
-valori: sono i valori degli elementi della collezione
-operazioni: sono le operazioni di interesse sulla collezione.
Un tipo di dato specifica *cosa* un'operazione deve fare ma non *come* l'operazione stessa può essere realizzata, e soprattutto non dice come gli oggetti della collezione possono essere organizzati in modo che le operazioni siano efficienti e la collezione stessa occupi poco spazio di memoria

Struttura dati: è una particolare organizzazione delle informazioni che permette di supportare le operazioni di un tipo di dato. Una struttura dati specifica *come organizzare i dati* di un particolare tipo di dato e *come realizzare le operazioni* definite.

Tecniche di rappresentazione

Vediamo due tecniche fondamentali usate per rappresentare collezioni di elementi:

-una tecnica basata su strutture indicizzate (array)
-una tecnica basata su strutture collegate (puntatori)

La scelta di una tecnica piuttosto di un'altra può avere un impatto cruciale sulle prestazioni delle operazioni fondamentali, cioè ricerca, inserimento e cancellazione

Strutture indicizzate (array)

Un array è una struttura indicizzata costituita da una collezione di celle numerate che possono contenere elementi di tipo prestabilito. Ipotesi:

- in un array di dimensione h (cioè contenente h celle) gli indici variano tra 0…h-1 (in java) oppure tra 1…h (in pseudocodice).
- E`possibile accedere in lettura e scrittura ad una qualsiasi cella in tempo costante

Proprietà basilari degli array:
1.(Forte) Gli indici delle celle di un array sono numeri consecutivi
2.(Debole) Non è possibile aggiungere nuove celle ad un array [ridimensionamento solo tramite riallocazione]

Le proprietà hanno conseguenze sull'efficienza delle operazioni!

Strutture collegate: record e puntatori

Per le strutture collegate i costituenti di base sono i record o nodi, che sono numerati e contengono ognuno un elemento della collezione.
I numeri associati ai record sono tipicamente i loro indirizzi di memoria, che non sono necessariamente consecutivi (visto che record vengono creati e distrutti dinamicamente in memoria).
Per mantenere i record di una collezione in relazione tra loro ognuno di essi contiene uno o più indirizzi di altri record della collezione.
Se il record A contiene l'indirizzo di un record B, diremo che esiste un *collegamento* tra A e B tramite un puntatore. I collegamenti tra record permettono ad un programma di esplorare una struttura collegata "saltando" da un record all'altro. *E`quindi importante che ci sia un record da cui si possono raggiungere tutti gli altri record.*

Proprietà delle strutture collegate

1. (Forte) E`sempre possibile aggiungere o togliere record ad una struttura collegata
2. (Debole) Gli indirizzi dei record di una struttura collegata non sono necessariamente consecutivi

Anche in questo caso le proprietà hanno conseguenze sull'efficienza delle operazioni

Tipo di dato: Contenitore

I contenitori (container) sono tipi di dato che permettono di memorizzare elementi e di poterli poi reperire. I contenitori permettono quindi di gestire collezioni di oggetti.
Le operazioni definite per i contenitori sono:

-put(C,x) inserisce un nuovo elemento x nel container C
-get(C) reperisce il prossimo elemento dal container C

I container differiscono per il modo con cui vengono reperiti gli elementi. Il metodo di reperimento si basa sull'ordine di inserimento e/o sulla posizione.

Contenitori

I contenitori più conosciuti sono:

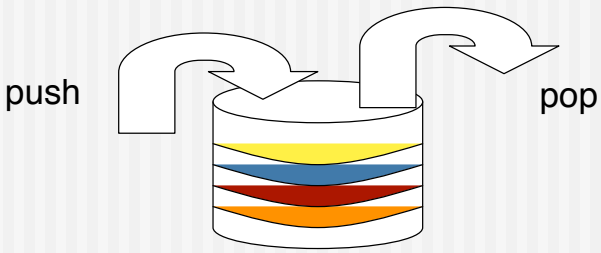
Pile (Stack): supportano il reperimento in ordine “last in first out” (LIFO). Le pile sono semplici da implementare e molto efficienti. Le operazioni di put e get per le pile si chiamano push e pop.

Code (Queue): supportano il reperimento di elementi in ordine “first in first out” (FIFO). Le operazioni put e get per le code si chiamano enqueue e dequeue .

Questi container possono essere implementati usando sia array che liste concatenate.
Usando gli array si pone un limite alla dimensione massima dei container. Per contro, le operazioni put e get possono essere implementate in tempo costante.

Tipo di dato Pila o Stack

Una pila può essere visualizzata in questo modo:



Esempio: una pila di piatti, un tubetto di pastiglie…

Tipo di dato Pila o Stack

Definiamo per il tipo di dato Stack le seguenti operazioni:

push(S,x) inserisce x in S in cima allo stack

pop(S) cancella e restituisce l'elemento in cima allo stack

---

top(S) restituisce l'elemento in cima allo stack

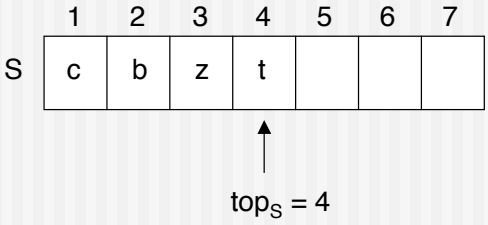
isEmpty(S) ritorna true sse lo stack è vuoto

size(S) ritorna il numero di elementi in S

clear(S) svuota lo stack

Pila rappresentata con array

Rappresentiamo uno stack S con un array e manteniamo un attributo top_S che ad ogni istante punta alla cima della pila, ovvero all'indice dell'ultimo elemento inserito.



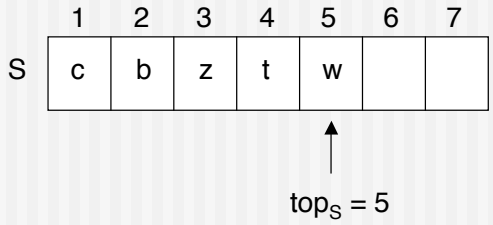
Pila rappresentata con array

Vediamo come inserire un nuovo elemento nello stack:

push(S,x)
1. $top_S \leftarrow top_S + 1$
2. $S[top_S] \leftarrow x$

La precondizione per poter eseguire questa operazione è che lo stack non sia pieno!

Es: push(S,"w");

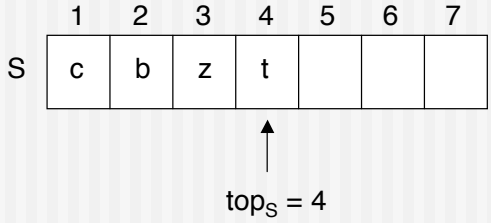


Pila rappresentata con array

Vediamo come cancellare un elemento dallo stack:
pop(S)
1. aux $\leftarrow top_S$
2. $top_S \leftarrow top_S$ -1
3. return S[aux]

La precondizione per poter eseguire questa operazione è che lo stack non sia vuoto!

Es: pop(S) ritorna "w", ovvero l'ultimo elemento inserito;



Pila rappresentata con array

Vediamo le altre operazioni:

isEmpty(S)
1. if top_S= 0
2. then return true
3. else return false

top(S)
1. return S[top_S]

La precondizione per poter eseguire questa operazione è che lo stack non sia vuoto!

La classe Object

Object è la classe generica (superclasse) per i riferimenti ad oggetti di qualsiasi altra classe. Tutte le classi estendono direttamente o indirettamente la classe Object.

```
public class Data{ int giorno; int mese; int anno;}
Object oggi = new Data();
Data domani = new Data();
oggi = “pippo pluto paperino”;
domani = “pippo pluto paperino”;   // TYPE ERROR !!
```

Tutte le classi quindi ereditano i metodi della classe Object, es:
public boolean equals(Object obj) : confronta il valore del riferimento dell’oggetto ricevente con il valore del riferimento del parametro. Se sono uguali ritorna true, altrimenti false. Le sottoclassi fanno override di questo metodo.
public String toString() : ritorna una stringa che rappresenta l’oggetto ricevente. Di solito si fa override nelle sottoclassi.

Pile in Java

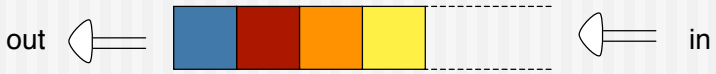
```
public class StackArray  {
    private static final int MAX=100;  // dimensione massima dello stack
    private Object[] S;                 // lo stack
    private int top;                    // puntatore al top dello stack

    // post: costruisce uno stack vuoto
    public StackArray() {
        S = new Object[MAX];
        top = -1;
    }

    // post: ritorna il numero di elementi nello stack
    public int size() { return top +1; }

    // post: ritorna lo stack vuoto
    public void clear() { top = -1; }

    // post: ritorna true sse lo stack e' vuoto
    public boolean isEmpty() { return (top == -1); }

    // pre: stack non vuoto!
    // post: ritorna l'oggetto in cima allo stack
    public Object top() { return S[top]; }
```

Pile in Java

```
    // post: inserisce ob in cima allo stack se lo stack non e' pieno
    public void push(Object ob) {
        if (top == MAX -1)
           return;
        S[++top] = ob;
    }
    // pre: stack non vuoto!
    // post: ritorna e rimuove l'elemento in cima allo stack
    public Object pop() {return S[top--];}
}
```

Tipo di dato Coda o Queue

Una coda può essere visualizzata in questo modo:



Esempio: la fila ad uno sportello!

Tipo di dato Coda o Queue

Definiamo per il tipo di dato Queue le seguenti operazioni:

enqueue(Q,x) inserisce x in Q come elemento in coda (ultimo)

dequeue(Q) cancella e restituisce il primo elemento in coda

---

front(Q) restituisce il primo elemento in coda

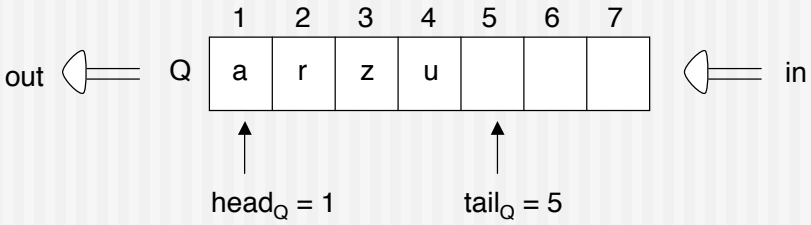
isEmpty(Q) ritorna true sse la coda è vuota

size(Q) ritorna il numero di elementi in Q

clear(Q) svuota la coda

Coda rappresentata con array

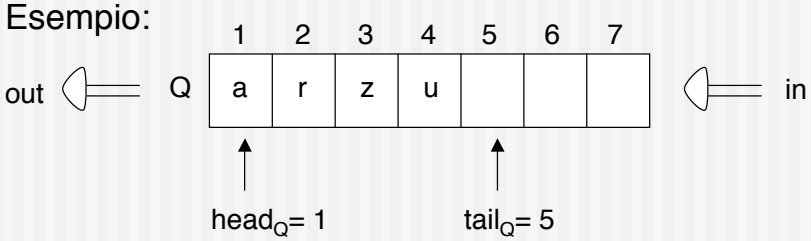
Rappresentiamo una coda Q con un array di dimensione N e manteniamo due attributi:
$head_Q$ è l'indice della posizione da cui estrarre un elemento
$tail_Q$ è l'indice della posizione in cui inserire un nuovo elemento



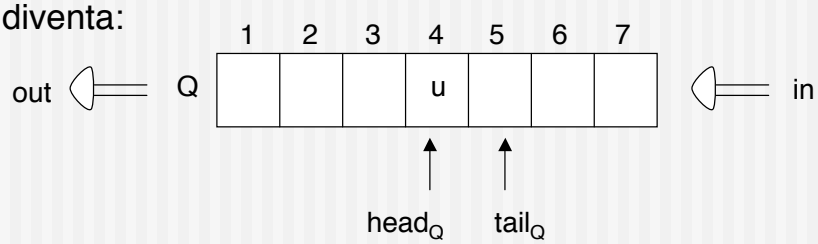
Se l'array viene gestito normalmente, cioè mantenendo $head_Q < tail_Q$, ci sono dei problemi…

Coda rappresentata con array

Esempio:



Se rimuoviamo uno alla volta i primi tre elementi in coda l'array diventa:



Risultano disponibili solo tre posizioni, anche se in coda c'e' un unico elemento!

Coda rappresentata con array

Prima soluzione: ad ogni rimozione si compatta l'array nelle posizioni iniziali [COSTOSO!]

Seconda soluzione: si gestisce l'array in modo circolare, ovvero

- Inizialmente $head_Q = tail_Q = 1$
- In ogni istante, gli elementi della coda si trovano nel segmento $head_Q$, $head_Q+1$,… $tail_Q-1$
  però non necessariamente $head_Q <= tail_Q$
- Infatti, dopo aver inserito in posizione N, se c'è ancora spazio in coda, si inseriscono ulteriori elementi a partire dalla prima posizione
- In questo modo si riesce a garantire che ad ogni istante la coda abbia capacità massima di N elementi

Coda rappresentata con array

Vediamo l'inserimento di un nuovo elemento in coda:

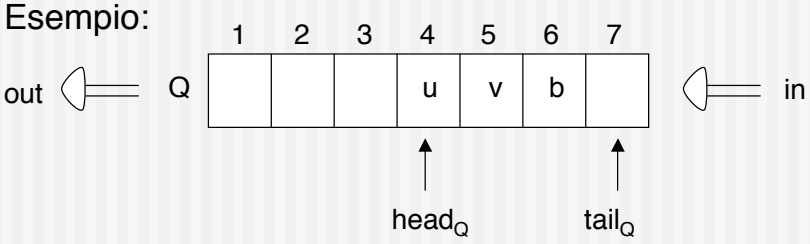
enqueue(Q,x)

1. $Q[tail_Q] \leftarrow x$
2. If $tail_Q = length(Q)$
3. then $tail_Q \leftarrow 1$
4. else $tail_Q \leftarrow tail_Q + 1$

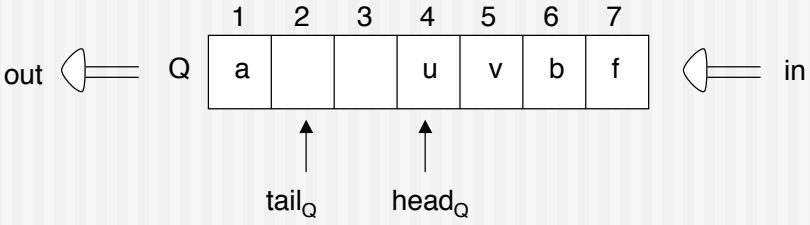
La precondizione per poter eseguire questa operazione è che la coda non sia piena!

Coda rappresentata con array

Esempio:



enqueue(S,"f")
enqueue(S,"a");



Coda rappresentata con array

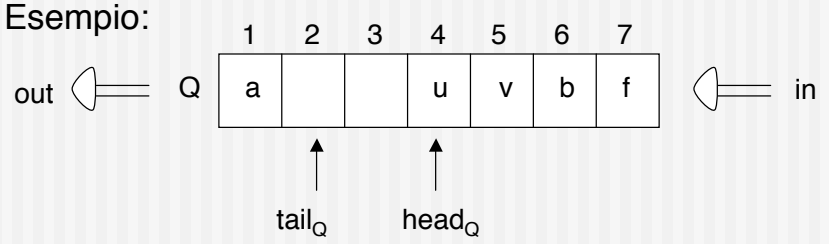
Vediamo la rimozione di un elemento dalla coda:

dequeue(Q)
1. X ← Q[head$_Q$]
2. If head$_Q$ = length(Q)
3. then head$_Q$ ← 1
4. else head$_Q$ ← head$_Q$ + 1
5. return X

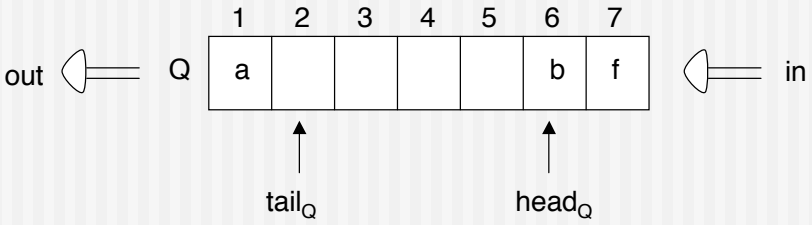
La precondizione per poter eseguire questa operazione è che la coda non sia vuota!

Coda rappresentata con array

Esempio:



dequeue(S)
dequeue(S)



Coda rappresentata con array

Vediamo le altre operazioni:

isEmpty(Q)
1. if head$_Q$ = tail$_Q$
2. then return true
3. else return false

front(Q)
1. return Q[head$_Q$]

La precondizione per poter eseguire questa operazione è che la coda non sia vuota!

Code in Java

```
public class QueueArray {
    private static final int MAX=100;   // dimensione massima della coda
    private Object[] Q;                 // la coda
    private int head;                   // puntatore al primo elemento in coda
    private int tail;                   // puntatore all'ultimo el. della coda

    // post: costruisce una coda vuota
    public QueueArray() {
        Q = new Object[MAX];
        head = 0;tail = 0;
    }

    // post: ritorna il numero di elementi nella coda
    public int size() { return tail - head; }

    // post: ritorna true sse la coda e' vuota
    public boolean isEmpty() { return (head  == tail); }

   // post: svuota la coda
    public void clear() { head = tail = 0;}
```

Code in Java

```
    // pre: coda non vuota
    // post: ritorna il valore del primo elemento della coda
    public Object front() { return Q[head % MAX]; }

     // pre: value non nullo
    // post: inserisce value in coda se la coda non e' piena
    public void enqueue(Object ob) {
        if (tail - head == MAX)
           return;
        Q[tail % MAX] = ob;
        tail = tail + 1;
        if (head > MAX) {
            tail = tail - MAX;
            head = head - MAX;
        }
    }

    // pre: coda non vuota
    // post: ritorna e rimuove l'elemento il primo elemento in coda
    public Object dequeue() {
        Object temp = Q[head % MAX];
        head = (head + 1);
        return temp;
    }
```