Facoltà di Scienze Matematiche Fisiche Naturali

Laboratorio di Algoritmi e Strutture Dati

Prof. Aniello Murano

Stack e Code

Corso di Laurea	Informatica	Lezione numero: 5
Codice insegnamento	13917	
Email docente	murano@na.infn.it	Parole chiave: LIFO, FIFO
Anno accademico	2007/2008	

next

Facoltà di Scienze Matematiche Fisiche Naturali

Stack(pile) e Code

- Stack e code sono insiemi dinamici in cui l'elemento rimosso dall'insieme con l'operazione di cancellazione è sempre predeterminato.
- In uno stack, l'elemento cancellato è quello più recentemente inserito. Gli stack rispettano la politica LIFO (***last-in, first-out)***.
- In una coda, l'elemento cancellato è sempre quello che è rimasto più a lungo nell'insieme. Le code rispettano la politica FIFO (***first-in, first out)***
- In uno stack un nuovo elemento è sempre posto in testa agli altri, mentre nella coda esso è posto dopo tutti gli altri.
- Ci sono molti modi per implementare stack e code su un computer. Cominciamo con una implementazione di stack utilizzando array.

back next

Stack

Le operazioni fondamentali su Stack sono:

- **Push(S,x)** che serve a inserire l'elemento **x** al top dello stack
- **Pop(S)** che serve a cancellare il top dello stack **S**

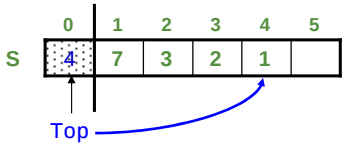
Un esempio di stack può essere dato da una pila di piatti posta su un tavolo.

- Si noti che l'ordine in cui i piatti sono tolti dallo stack è inversa all'ordine in cui essi sono stati inseriti nello stack, visto che solo il top dello stack è accessibile.
- Un'altra operazione importante sugli stack è **Empty-Stack**, necessaria per controllare se uno stack è vuoto

back next

Implementazione di Stack con Array

- Per implementare uno stack di **n** elementi, si può utilizzare un array S di dimensione n.
- L'array S utilizzerà l'attributo TOP che indicherà l'indice dell'elemento più recente immesso nello stack. In una implementazione si potrebbe pensare di memorizzare questo indice nel primo elemento dell'array che dunque avrà come dimensione non più n ma **n+1**.
- Esempio:



- Un'operazione di Pop sullo stack dell'esempio precedente, restituirà l'elemento 1 e il TOP diventerà 3.
- Quando TOP=0, lo stack è vuoto, cioè non contiene elementi.

back next

Implementazione di Push e Pop

- Utilizzando un array **S[MAX+1]** per l'implementazione di uno stack, nel modo descritto precedentemente, le operazioni di **Push** e **Pop** possono essere realizzate semplicemente nel modo seguente:

Push

```
void push(int S[], int valore)
{
        S[0] = S[0]+ 1;
        S[S[0]]= valore;
}
```

Pop

```
int pop(int S[])
{
        S[0] = S[0]-1;
        return S[S[0]+1];
}
```

back next

Implementazione di Empty_stack e Full_stack

- Utilizzando un array **S[MAX+1]** per l'implementazione di uno stack di **MAX** elementi, empty_stack può essere realizzata controllando se Top=0. Inoltre, si può utilizzare una procedura full_stack per controllare se lo stack è pieno, cioè se Top=MAX.

empty_stack

```
int emtpy_stack(int S[])
{
        return S[0]==0;
}
```

full_Stack

```
int full_stack(int S[])
{
        return S[0]==MAX;
}
```

back next

Implementazione efficiente di Push e Pop

- Facoltativo: Utilizzando empty_stack e full_stack è possibile realizzare una versione più efficiente di Pop e Push con controllo di errore, nel modo seguente:

Push_check

```
void push_c(int S[], int val, int *err)
{
    if (full_stack(S))
        *err=1;
    else
    {
        S[0] = S[0]+ 1;
        S[S[0]]= val;
        *err=0;
    }
}
```

Pop_check

```
int Pop_c(int S[],int *err)
{
    int val=0;
    if (empty_stack(S))
        *err=1;
    else
    {
        S[0] = S[0]-1;
        val=S[S[0]+1];
        *err=0;
    }
    return val;
}
```

back next

Costruzione di uno Stack

- La seguente procedura permette di costruire uno stack **S** di taglia **num_elementi**, sapendo che **S** può contenere al più **MAX** valori

```
void new_stack(int S[])
{       int num_elementi,valore;
        inizializza_stack(S);
        printf("\n Quanti elementi (max %d elementi): ", MAX);
        scanf("%d",&num_elementi);
        while (num_elementi >MAX)       {
                printf("\n max %d elementi: ", MAX);
                scanf("%d",&num_elementi);
                }
        while(num_elementi) {
                printf("\n Inserire un valore: ");
                scanf("%d",&valore);
                push(S, valore);
                --num_elementi;
                }
}
```

- Cosa succede se si inserisce **0** per num_elementi?

back next

Stampa di uno Stack

- La stampa di uno stack **S** può essere fatta nel modo seguente:

Stampa di uno Stack

```
void stampa_stack ( int S[])
{
    int valore;
    if (!empty_stack(S))
    {
        valore= pop(S);
        printf(" %d ",valore);
        stampa_stack(S);
        push(S,valore);
    }
}
```

back next

Gestione di uno Stack (1)

- Il seguente programma gestisce uno stack **S** di **MAX** valori. Si noti come i controlli siano indipendenti da MAX. Questo è utile quando MAX e S[MAX+1] sono dati esternamente al programma (come solitamente avviene)

```
#define MAX 20
main()
{
    int S[MAX+1],scelta,valore;
    do
    {
        printf("\n scelta: 0-Crea, 1-Stampa, 2-Pop, 3-Push, 4-uscita : ");
        scanf("%d ",&scelta);
        switch (scelta)
                { ..... } /* vedi prossima slide */
    }
    while(scelta==0||scelta==1 ||scelta==2||scelta==3);
}  /* fine main() */
```

back next

Gestione di uno Stack (2): Implement. Switch

```
switch (scelta)
   {
        case 0:
                new_stack(S);    break;
        case 1:
                stampa_stack(S);          break;
        case 2:
                if (!empty_stack(S))
                        printf("\n Top dello Stack %d", pop(S));
                else
                        printf("\n spiacente, stack vuoto");
                break;
        case 3:
                if (!full_stack(S)) {
                        printf("\n valore da inserire nello stack: ");
                        scanf("%d",&valore);
                        push(S,valore);
                        }
                else
                        printf("\n spiacente, stack pieno");
   } /* fine switch */
```

back next

Esercizio su Stack

- Si consideri uno Stack **S**, implementato con array **S[MAX+1]**.
- Si implementi la funzione **ricorsiva**

void togli_da_Stack(int S[], int el)

che elimini dallo stack S tutti gli elementi uguali ad **el** lasciando invariato l'ordine degli altri elementi.
- Non utilizzare altre strutture dati di appoggio.
- Non utilizzate accessi diretti all'array, ma servitevi solo delle funzioni implementate per la gestione degli stack.
- Si ricordi che lo stack è una struttura dati che permette l'accesso ai suoi dati solo dal top.

back next

Code

A differenza dello stack, una coda usa due attributi:

- Inizio coda (Head)
- Fine coda (Tail)

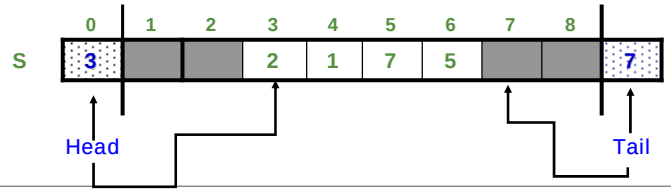
In pratica, usando come esempio di coda una fila ad uno sportello. L'inizio della coda è rappresentato dalla prima persona della fila (quella la prossima ad essere servita), mentre la fine della coda è rappresentata dall'ultima persona che si è aggiunta alla coda (cioè, l'ultima persona tra quelle attualmente in fila ad essere servita)

Un inserimento nella coda sarà fatto sempre alla sua fine mentre una cancellazione alla sua testa

back next

Implementazione di Code

- Come per gli stack, anche per le code possiamo avere diversi modi di rappresentazione su un computer.
- Iniziamo con l'utilizzo di array.
- Per memorizzare una coda con massimo **n** elementi, possiamo utilizzare uno stack di dimensione **n** più due variabili che memorizzano costantemente l'indice della testa e della coda.
- Come per gli stack, si può anche scegliere di memorizzare l'indice della testa e della coda nel vettore stesso.
- Per esempio:



back next

Implementazione di Empty_queue e Full_queue

- Utilizzando un array **Q[MAX+2]** per l'implementazione di una coda di **MAX** elementi, **empty_queue** può essere realizzata controllando se Head=0 (in questo caso Teal sarà uguale a 1).
- Inoltre, si può utilizzare una procedura **full_queue** per controllare se la coda è piena, cioè se Head=Tail.

empty_queue

```
int emtpy_stack(int Q[])
{
        return Q[0]==0;
}
```

full_queue

```
int full_queue( int Q[])
{
    return Q[0]==Q[MAX+1];
}
```

- Dunque, per creare una coda vuota basterà avere Head= 0 e Teal=1

back next

Implementazione di Dequeue e Enqueue

Usando l'array Q precedente per l'implementazione di una coda, le operazioni di Cancellazione (Dequeue) e Inserimento (Enqueue) possono essere realizzate come segue:

Enqueue

```
void enqueue(int Q[], int valore)
{
   Q[Q[MAX+1]] = valore;
   if (Q[0] == 0)
         Q[0]=1;
   Q[MAX+1] = (Q[MAX+1] % MAX) + 1;
}
```

Dequeue

```
int dequeue(int Q[])
{
   int valore=Q[Q[0]];
   Q[0] = (Q[0] % MAX) + 1;
   if (Q[0] == Q[MAX+1])  {
         Q[0]=0;
         Q[MAX+1]=1;      }
return valore;
}
```

back next

Stampa di una Coda

- **Per la stampa di una coda, non possiamo usare esattamente la stessa procedura vista per gli stack. Infatti questa produrrebbe una inversione della coda. Per risolvere questo problema, provvediamo a invertire ulteriormente la coda. Dunque la funzione stampa prima chiama stampa_queue e poi rewerse**

stampa_queue

```
void stampa_queue(int Q[])
{
   int val;
   if (!empty_queue(Q))
   {
       val=dequeue(Q);
            printf(" %d \|",val);
            stampa_queue(Q);
            enqueue(Q,val);
       }
    }
```

rewerse

```
void rewerse (int Q[])
{
    int val;
    if (!empty_queue(Q))
    {
        val=dequeue(Q);
        rewerse(Q);
        enqueue(Q,val);
    }
}
```

back next

Costruzione di una Coda

- La seguente procedura permette di costruire una coda **Q** di taglia **num_elementi**, sapendo che **Q** può contenere al più **MAX** valori

```
void new_queue(int Q[])
{
  int num_elementi,valore;
  inizializza_queue(Q); /* set s[0]=0 e s[MAX+1]=1 */
  printf("\n Quanti elementi (max %d elementi): ", MAX);
  scanf("%d",&num_elementi);
  while (num_elementi >MAX) {
      printf("\n max %d elementi: ", MAX);
              scanf("%d",&num_elementi);
      }
  while(num_elementi) {
      printf("\n Inserire un valore: ");
      scanf("%d",&valore);
      enqueue(Q, valore);
      num_elementi--;
      }
}
```

back next

Gestione di una Coda (1)

- Il seguente programma gestisce una coda **Q** di **MAX** valori. Si noti come i controlli siano indipendenti da MAX. Questo è utile per le stesse motivazioni date per gli stack

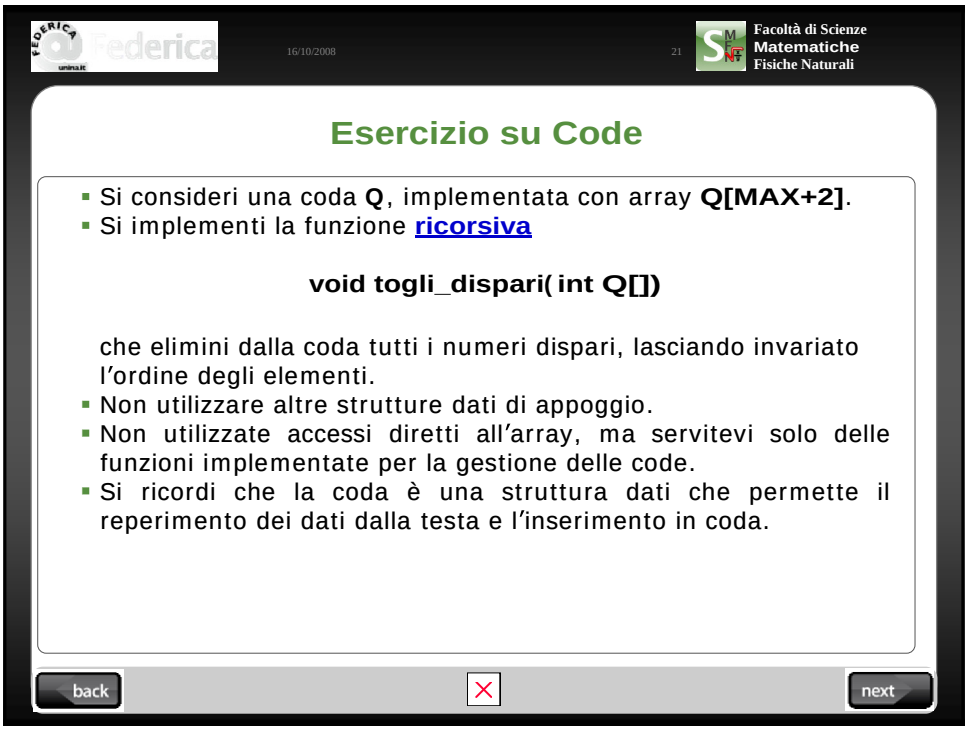
```
#define MAX 20
 main()
 {
    int Q[MAX+1],scelta,valore; /* usare typedef  per avere il main
                                   indipendente dalla scelta int
  Q[MAX+1] */
    do
    {
       printf("\n scelta: 0-Crea, 1-Stampa, 2-Deq, 3-Enq,, 4-uscita
  : ");          scanf("%d ",&scelta);
       switch (scelta)
                        {   .....   } /* vedi prossima diapositiva */
    }
    while(scelta==0||scelta==1||scelta==2||scelta==3);
 } /* fine main() */
```

back next

Gestione di una Coda(2): Implement. switch

```
switch (scelta)
    {
        case 0:
            new_queue(Q);           break;
        case 1:
            stampa_queue(Q);        break;
        case 2:
            if (!empty_queue(Q))
                printf("\n Head della coda %d", dequeue(S));
            else
                printf("\n spiacente, coda vuoto");
            break;
        case 3:
            if (!full_queue(Q)))        {
                printf("\n Valore da inserire nello stack: ");
                scanf("%d",&valore);
                enqueue(Q,valore);
                    }
            else
                printf("\n spiacente, coda piena");
    } /* fine switch */
```

back next


Federica
16/10/2008
21
Facoltà di Scienze
Matematiche
Fisiche Naturali
Esercizio su Code
Si consideri una coda Q, implementata con array Q[MAX+2].
Si implementi la funzione ricorsiva
void togli_dispari(int Q[])
che elimini dalla coda tutti i numeri dispari, lasciando invariato l'ordine degli elementi.
Non utilizzare altre strutture dati di appoggio.
Non utilizzate accessi diretti all'array, ma servitevi solo delle funzioni implementate per la gestione delle code.
Si ricordi che la coda è una struttura dati che permette il reperimento dei dati dalla testa e l'inserimento in coda.
back
next