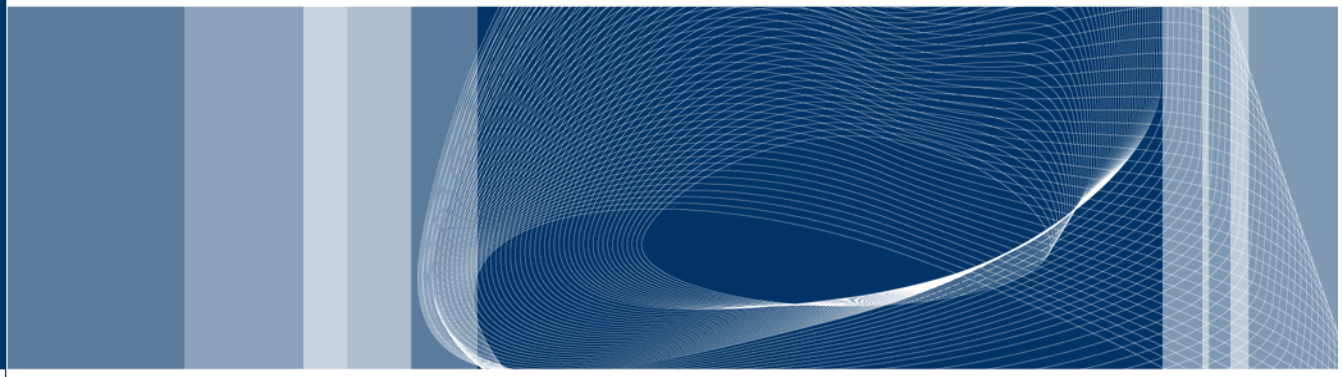
POLITECNICO DI MILANO

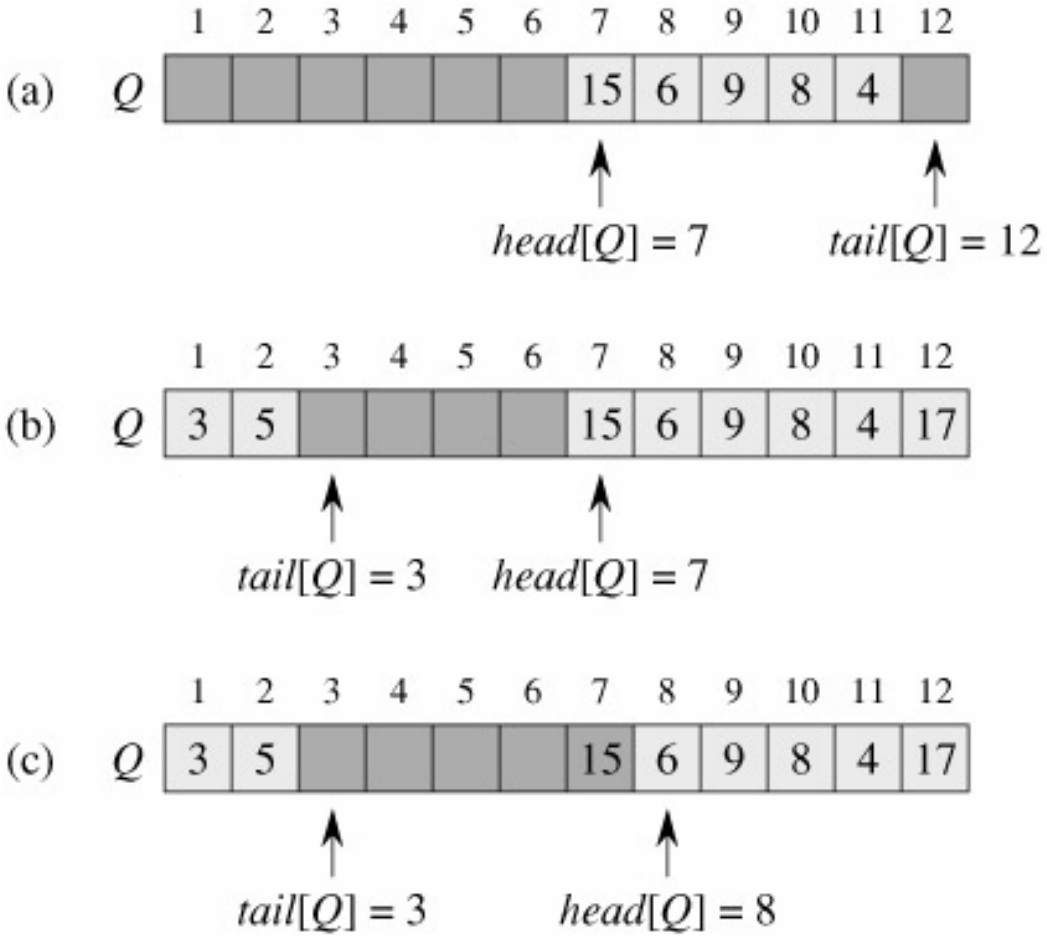


Algoritmi e Strutture Dati

Laboratorio 03/11/2008

Daniele Loiacono

Implementazione delle Code con array circolari

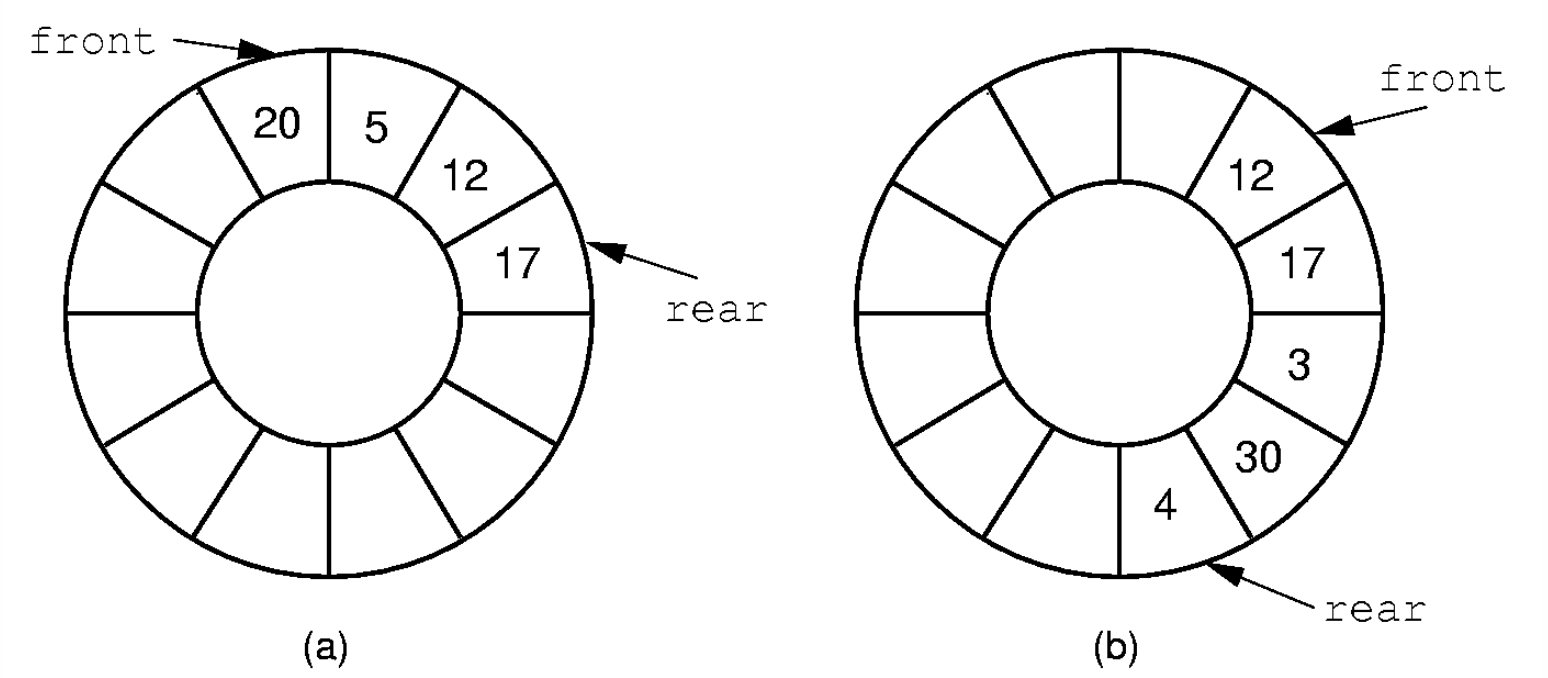


a) Coda iniziale

b) Dopo aver svolto ENQUEUE(Q, 17), ENQUEUE(Q, 3), e ENQUEUE(Q, 5)

c) Dopo aver svolto DEQUEUE(*Q*) che ritorna il valore 15

Implementazione delle Code con array circolari



Implementazione delle Code con array circolari


```
ENQUEUE(Q, x)
1  Q[tail[Q]] ← x
2  if tail[Q] = length[Q]
3     then tail[Q] ← 1
4  else
5     tail[Q] ← tail[Q]+1
```

```
DEQUEUE(Q)
1  x ← Q[head[Q]]
2  if head[Q] = length[Q]
3     then head[Q] ← 1
4  else
5     head[Q] ← head[Q]+1
6  return x
```

Esercizio 1

- Scrivere il template di una classe che implementi con un **array circolare** una coda derivata dalla seguente classe virtuale astratta

```
template <class Elem> class CodaBase {
public:
   virtual bool enque(Elem) = 0;
   virtual bool deque() = 0;
   virtual Elem head() = 0;
   virtual bool isempty() = 0;
};
```

Esercizio 1

- Verificare la classe implementata usando il seguente frammento di codice:

```
CodaCirc<int> c(10);

for(i=0;i<10;i++) c.enqueue(rand());

for(i=0;i<10;i++) {
  cout << c.head();
  c.dequeue();
}
```

C++ Standard Template Library

- Il C++ mette a disposizione diverse algoritmi e strutture dati che possono essere utilizzati dai programmatori
- Questi algoritmi e strutture dati sono raccolti in una libreria standard del C++ chiamata Standard Template Library (STL)

STL: vector

- La classe **vector** è un template che consente di memorizzare un insieme di elementi contigui (come avviene negli array) e di accedervi a tempo costante
- Dichiarazione:

```
vector<tipo> the_vector;
```

- Funzioni principali
  - `the_vector[i]`: accesso all'elemento i-esimo
  - `void clear()`: elimina tutti gli elementi del vettore
  - `bool empty()`: ritorna true se il vettore è vuoto
  - `void push_back( const tipo& val )`: aggiunge un elemento in fondo
  - `size_type size()`: restituisce la dimensione del vettore

STL: vector (2)

```
#include <cstdlib>
#include <iostream>
#include <vector>
using namespace std;
int main(int argc, char *argv[])
{
    vector<int> v;
    int i;
    for (i=0; i<10; i++)
        v.push_back((int) rand() % 10);
    for (i=0; i<v.size(); i++)
        cout << "Elem " << i << ": " << v[i] << endl;
}
```

STL: iterator

- Gli iteratori vengono usati per accedere agli elementi di un vector (prendono il posto dei puntatori utilizzati negli array)

```
vector<tipo>::iterator the_iterator;
```

- Funzioni principali
  - `the_iterator++`: si sposta all'elemento successivo nel vettore
  - `the_iterator--`: si sposta all'elemento precedente nel vettore
  - `*the_iterator`: valore dell'elemento "puntato" dall'iteratore
  - `iterator begin()`: restituisce un iterator al primo elemento del vector
  - `iterator end()`: restituisce un iterator alla **posizione successiva** all'ultimo elemento del vector
  - `iterator erase( iterator loc )`: elimina l'elemento alla posizione identificata dall'iterator loc e restituisce un iterator alla posizione successiva all'elemento appena cancellato

STL: iterator (2)

```
#include <cstdlib>
#include <iostream>
#include <vector>
using namespace std;
int main(int argc, char *argv[])
{
    vector<int> v;
    vector<int>::iterator i,j;
    int sum=0;
    for (i=v.begin(); i<v.end(); i++)
    {
        sum += *i;
        j=i;
        v.erase(j);
    }
}
```

Esercizio 2

- Scrivere il template di una classe che implementi con un **vector** una coda derivata dalla seguente classe virtuale astratta

```
template <class Elem> class CodaBase {
public:
   virtual bool enque(Elem) = 0;
   virtual bool deque() = 0;
   virtual Elem head() = 0;
   virtual bool isempty() = 0;
};
```

Esercizio 2

- Verificare la classe implementata usando il seguente frammento di codice:

```
CodaVect<int> c;

for(i=0;i<10;i++) c.enqueue(rand());

for(i=0;i<10;i++) {
  cout << c.head();
  c.dequeue();
}
```