Si implementi il tipo di dato astratto `BitQueue`, che memorizza l'informazione contenuta in una coda first-in first-out (FIFO) di booleani all'interno di singoli bit di una variabile intera senza segno della stessa lunghezza, al fine di minimizzare l'uso della memoria. Si assuma che gli `unsigned short` siano su 16 bit. Le operazioni sul tipo `BitQueue` debbono essere il più veloci possibile, quindi viene espressamente richiesto di utilizzare le operazioni bit-a-bit.

E' VIETATO L'USO DI VETTORI IN CUI LA SINGOLA COMPONENTE RAPPRESENTI IL SINGOLO BIT, QUALI, AD ESEMPIO, I VETTORI DI BOOL.

Si implementi il tipo di dato astratto `BitQueue`, fornendo le seguenti funzionalità:

--- Metodi invocati nella PRIMA PARTE di main.cpp: ---

- ✓ **`BitQueue q(max_len);`**
  Costruttore che crea una `BitQueue q` di capienza massima `max_len` bit. All'inizio la coda è vuota.
- ✓ **`cout<<q;`**
  Operatore di uscita per il tipo `BitQueue`. L'uscita ha la seguente forma:

  ```
  12[1,0,0,0,1,1]
  ```

  dove il primo numero rappresenta la capienza massima in cifre decimali, e poi viene riportata la sequenza di valori binari 0 o 1 tra parentesi quadre, secondo la consueta corrispondenza `true = 1`, `false = 0`. Gli estremi della coda sono delimitati da parentesi quadre ***senza*** alcuno spazio tra le stesse e il più vicino elemento della coda. L'elemento più a destra rappresenta la testa della coda, cioè l'ultimo elemento a essere stato inserito e pertanto l'ultimo che ne uscirà. L'operatore ***non*** aggiunge il *new line* alla fine.
- ✓ **`q[i];`**
  Operatore parentesi quadra che restituisce il valore del bit di indice `i` della coda, considerando il primo bit ad essere stato inserito come avente indice 0. L'operatore restituisce un booleano. Non importa che tale ritorno sia un valore sinistro, quindi può essere restituito per valore. Se l'indice non è valido, l'operatore deve restituire false.
- ✓ **`q.enqueue(bit);`**
  Operazione che inserisce un bit di valore booleano `bit` alla coda. La funzione restituisce `false` se la coda ha già raggiunto la capienza massima e quindi l'operazione non può essere eseguita, `true` altrimenti. Il tempo di esecuzione non deve dipendere dalla lunghezza della coda.
- ✓ **`q.dequeue(bit);`**
  Operazione che estrae un bit dalla coda. Il valore del bit è restituito mediante il booleano `bit`. La funzione restituisce `false` se la coda è già vuota e quindi l'operazione non può essere eseguita, `true` altrimenti. Il tempo di esecuzione non deve dipendere dalla lunghezza della coda.

--- Metodi invocati nella SECONDA PARTE di main.cpp: ---

- ✓ **`~BitQueue();`**

  Qualora necessario, implementare il distruttore.

- ✓ **`BitQueue q(q2);`**

  Costruttore di copia per oggetti `BitQueue`.

- ✓ **`q1|q2;`**

  Operatore di OR bit a bit tra `BitQueue` che restituisce una ***nuova*** `BitQueue` di capienza massima tra quelle di `q1` e `q2` e contenente il risultato dell'OR bit a bit tra `q1` e `q2`. Qualora le lunghezze delle due `BitQueue` fossero diverse, la `BitQueue` risultante ha lunghezza pari alla massima delle due e (***solo***) per l'operazione di OR la più breve delle due `BitQueue` deve essere considerata estesa mediante l'aggiunta in coda di valori **0.**

- ✓ **`q1.max_span(bit);`**

  Operazione che restituisce la lunghezza della più lunga sottosequenza di bit uguali al booleano `bit` consecutivi all'interno della `BitQueue`.

- ✓ **`a.replace(seq_find, seq_len, seq_rep);`**

  Operazione che rimpiazza tutte le occorrenze della sequenza di bit `seq_find` lunga `seq_len` con la sequenza di bit `seq_rep` ugualmente lunga. Le sequenze di bit sono passate come array di booleani. Evitare i rimpiazzi "ricorsivi", cioè dopo aver rimpiazzato una sequenza, evitare di cercare `seq_find` o pezzi di essa all'interno della sequenza appena rimpiazzata.

Mediante il linguaggio C++, realizzare il tipo di dato astratto definito dalle precedenti specifiche. Non è permesso utilizzare funzionalità della libreria STL come il tipo `string`, il tipo `vector`, il tipo `list`, ecc.

USCITA ATTESA

```
--- PRIMA PARTE ---
Test del costruttore
16[]

Test di enqueue
16[1,0,1,1]

Test di dequeue
1
16[0,1,1]
0 1 1
16[1,1,0,1]

Test di parentesi quadra
1 0
```

```
--- SECONDA PARTE ---
Test del costruttore di copia
16[1,1,0,1]

Test dell'or bit-a-bit
16[0,1,0,0]
16[1,1,0,1]

Test di max_span
2 2

Test di replace
16[1,0,0,1]
```

Note per la consegna:

Affinché l'elaborato venga considerato valido, il programma **deve** produrre almeno la prima parte dell'output atteso. In questo caso, i docenti procederanno alla valutazione dell'elaborato **solo se** lo studente avrà completato l'autocorrezione del proprio elaborato.
In **tutti** gli altri casi (per esempio, il programma non compila, non collega, non esegue o la prima parte dell'output non coincide con quella attesa), l'elaborato è considerato **insufficiente** e, pertanto, **non verrà corretto**.