Turing Machine

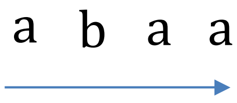
Topics to be covered

- What is Turing Machine?
- Formal Definition: Turing Machine
- Design of Turing Machine
- Universal Turing Machine
- Church Turing Thesis

Introduction

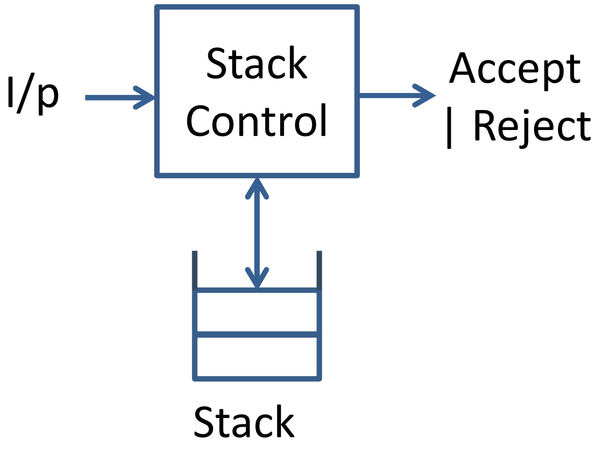
Finite Automata

- Input Strings



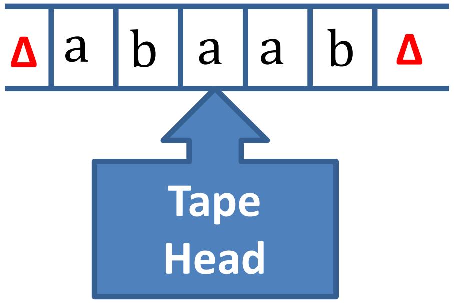
Pushdown Automata

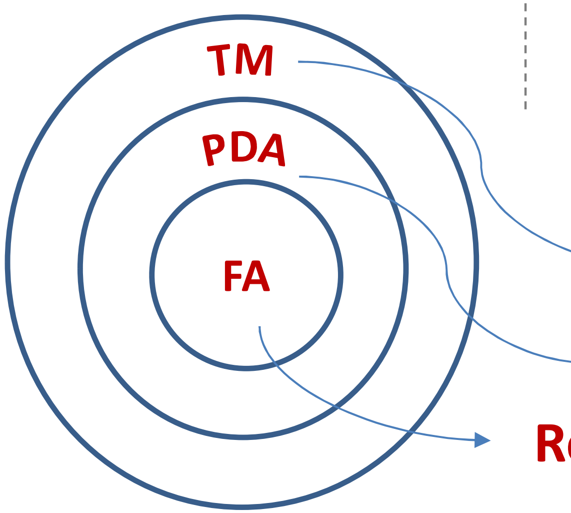
- Input Strings
- Stack



Turing Machine

- Input Strings
- Tape
- Special symbol





Recursively enumerable language

Context free language

Regular language

Turing Machine

- Turing Machine was invented by Alan Turing in 1936 and it is used to accept Recursive Enumerable Languages (generated by Type-0 Grammar)
- A Turing machine consists of a tape of infinite length on which read and writes operation can be performed.
- The tape consists of infinite cells on which each cell either contains input symbol or a special symbol called blank.
- It also consists of a head pointer which points to cell currently being read and it can move in both directions.

Turing Machine

There are various features of the Turing machine:

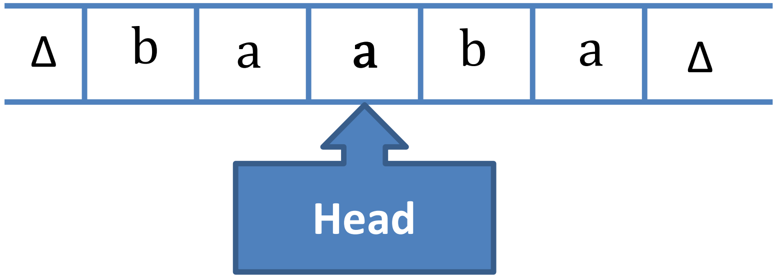
1. It has an external memory which remembers arbitrary long sequence of input.

2. It has unlimited memory capability.

3. The model has a facility by which the input at left or right on the tape can be read easily.

4. The machine can produce a certain output based on its input. Sometimes it may be required that the same input has to be used to generate the output. So in this machine, the distinction between input and output has been removed. Thus a common set of alphabets can be used for the Turing machine.

Turing Machine

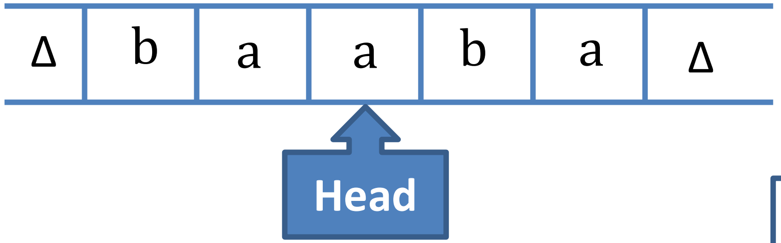


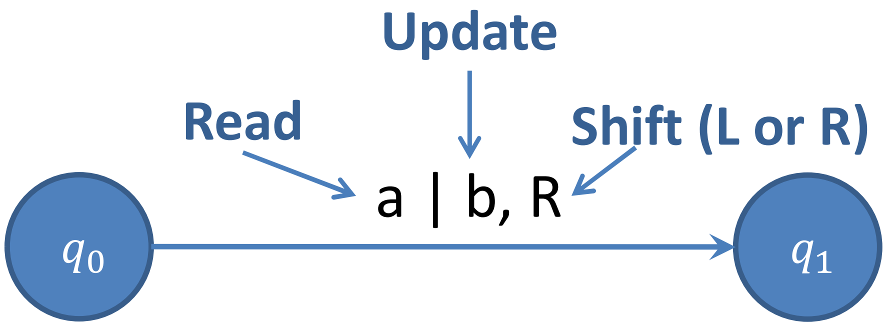
- The tape is capable of performing following three operations:
  - Read a symbol above the tape head.
  - Modify/Update a symbol above the tape head.
  - Shifting either to previous square or next square.

Turing Machine

Operation to be performed by TM:

- Read a symbol above the tape head.
- Modify/Update a symbol above the tape head.
- Shifting either to previous square or next square.





Some other Ex:

b | a, L

a | Δ, R

a | a, R

Definition: Turing Machine

- A Turing machine is a 5-tuple $T = (Q, \Sigma, \Gamma, q_0, \delta)$ where,

  Q : is a finite set of states, assumed not to contain h_a (Acceptance State) and h_r (Rejection State).

  Σ : *input* Symbols

  Γ : *tape* alphabets

  q_0 : is initial state, is an element of Q.

  δ : is a transition function.

Turing Machine Accepting ab^*a

Design a Turing Machine Accepting ab^*a

L = {ab*a}

Strings accepting in given L are:

aa

$a\ \underline{b}\ a$

$a\ \underline{b\ b}\ a$

$a\ \underline{b\ b\ b}\ a$

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)
   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting ab^*a

L = {ab*a}

Strings accepting in given L are:

aa

a b a

a b b a

a b b b a

Tape

Δ	a	b	a	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)
   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting $ab^{*}a$

L = {ab*a}

Strings accepting in given L are:

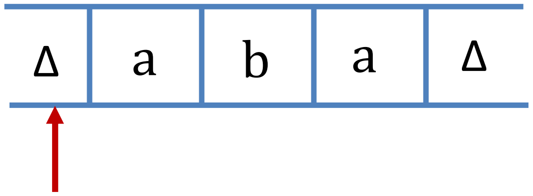
aa

a $\underline{\text{b}}$ a

a $\underline{\text{b b}}$ a

a $\underline{\text{b b b}}$ a

Tape



Δ	a	b	a	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)
   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting ab^*a

L = {ab*a}

Strings accepting in given L are:

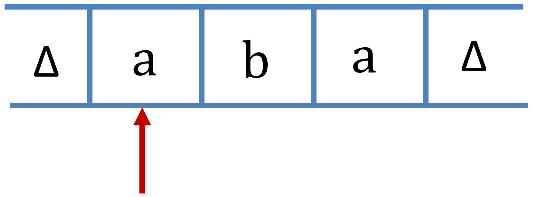
aa

a b a

a b b a

a b b b a

Tape



Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)
   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting ab^*a

L = {ab*a}

Strings accepting in given L are:

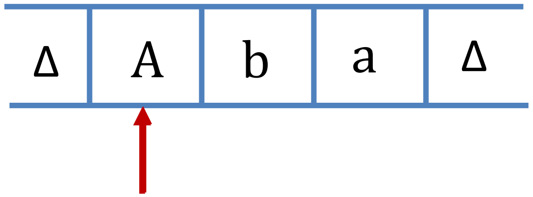
aa

$a\ \underline{b}\ a$

$a\ \underline{b\ b}\ a$

$a\ \underline{b\ b\ b}\ a$

Tape



Δ	A	b	a	Δ

Logic:

1. Read 'a', replace 'a' by 'A' (a→A)
2. Read all 'b' one by one & replace it by 'B' (b→B)

   if no 'b' found: SKIP step 2
3. Read 'a', replace 'a' by 'A' (a→A)

Design a Turing Machine Accepting ab^*a

L = {ab*a}

Strings accepting in given L are:

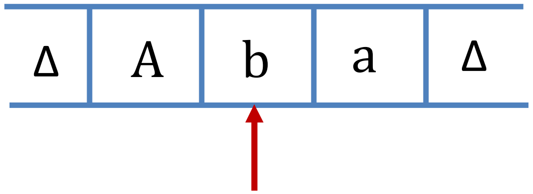
aa

a $\underline{b}$ a

a $\underline{b\ b}$ a

a $\underline{b\ b\ b}$ a

Tape



Δ	A	b	a	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)
   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting ab^*a

L = {ab*a}

Strings accepting in given L are:

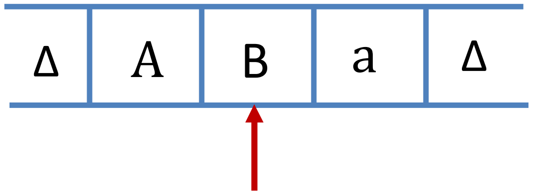
aa

a b a

a b b a

a b b b a

Tape



Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)
   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting ab^*a

L = {ab*a}

Strings accepting in given L are:

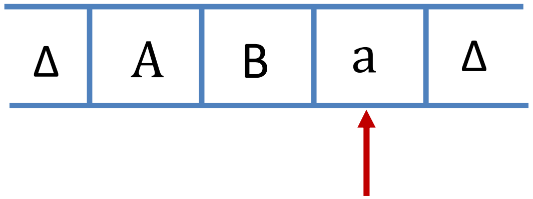
aa

a $\underline{b}$ a

a $\underline{b\ b}$ a

a $\underline{b\ b\ b}$ a

Tape



Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)
   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting ab^*a

L = {ab*a}

Strings accepting in given L are:

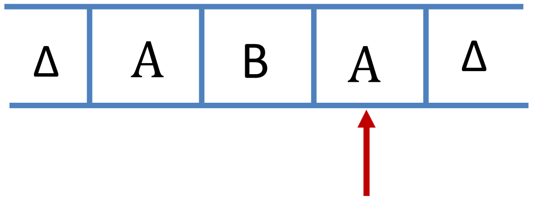
aa

$a\ \underline{b}\ a$

$a\ \underline{b\ b}\ a$

$a\ \underline{b\ b\ b}\ a$

Tape



Δ	A	B	A	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)
    if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting ab^*a

L = {ab*a}

Strings accepting in given L are:

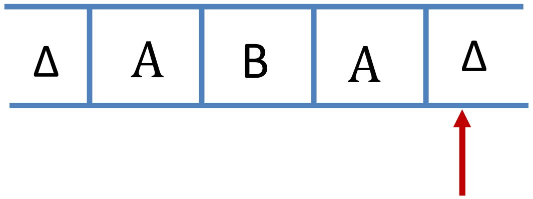
aa

$a\ \underline{b}\ a$

$a\ \underline{b\ b}\ a$

$a\ \underline{b\ b\ b}\ a$

Tape



Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)
   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting ab^*a

L = {ab*a}

Strings accepting in given L are:

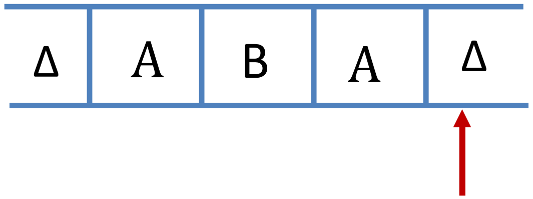
aa

$a\ \underline{b}\ a$

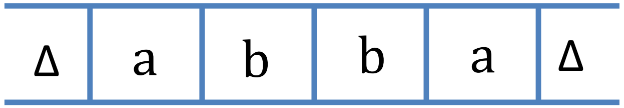
$a\ \underline{b\ b}\ a$

$a\ \underline{b\ b\ b}\ a$

Tape



Δ	A	B	A	Δ



Δ	a	b	b	a	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)
   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting ab^*a

L = {ab*a}

Strings accepting in given L are:

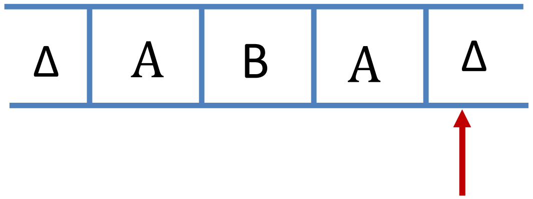
aa

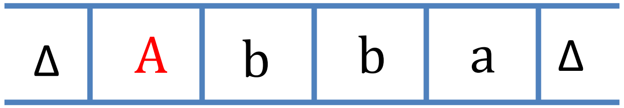
a b a

a b b a

a b b b a

Tape





Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)

   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting $ab^{*}a$

L = {ab*a}

Strings accepting in given L are:

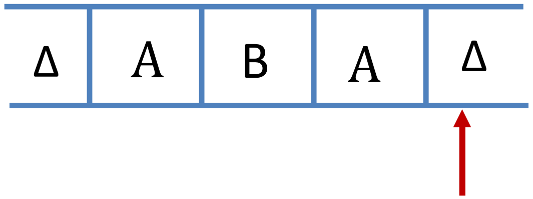
aa

a b a

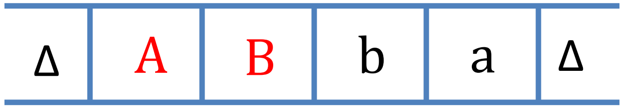
a b b a

a b b b a

Tape



Δ	A	B	A	Δ



Δ	A	B	b	a	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)
   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting $ab^{*}a$

L = {ab*a}

Strings accepting in given L are:

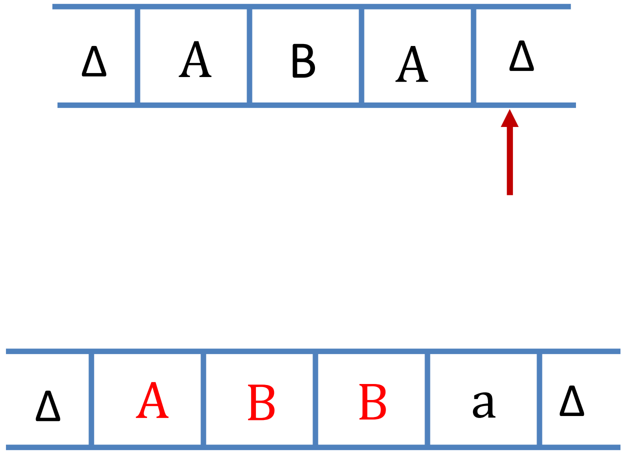
aa

a b a

a b b a

a b b b a

Tape



Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)

   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting ab^*a

L = {ab*a}

Strings accepting in given L are:

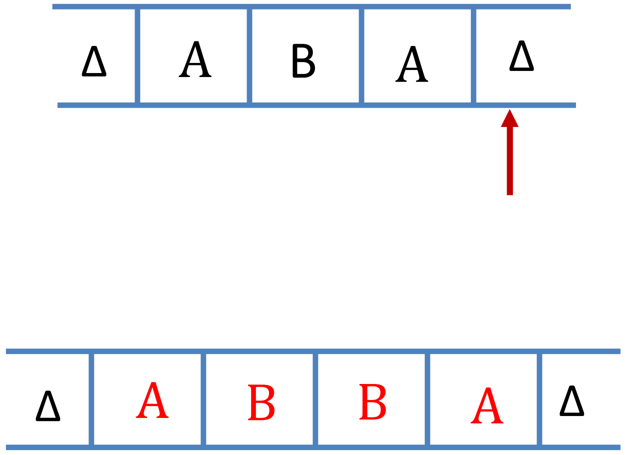
aa

a $\underline{b}$ a

a $\underline{b\ b}$ a

a $\underline{b\ b\ b}$ a

Tape



Δ	A	B	A	Δ	
Δ	A	B	B	A	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)
   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting ab^*a

L = {ab*a}

Strings accepting in given L are:

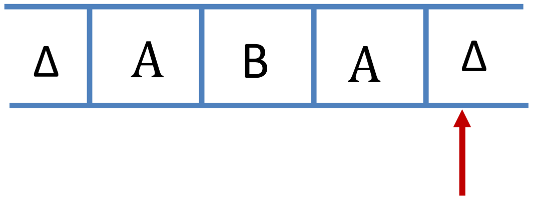
aa

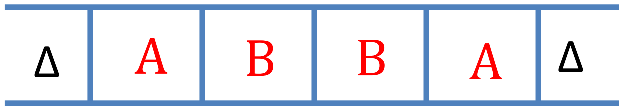
a b a

a b b a

a b b b a

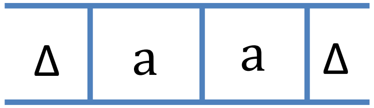
Tape





Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)
   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)



Design a Turing Machine Accepting ab^*a

L = {ab*a}

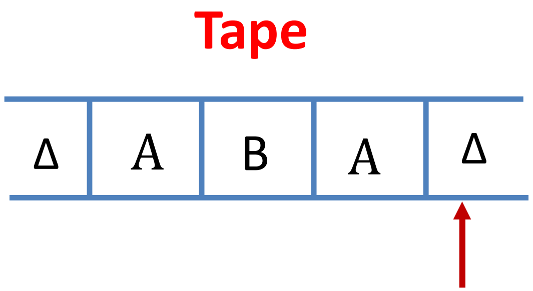
Strings accepting in given L are:

aa

a $\underline{\text{b}}$ a

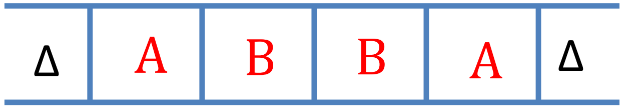
a $\underline{\text{b b}}$ a

a $\underline{\text{b b b}}$ a



Tape

Δ	A	B	A	Δ

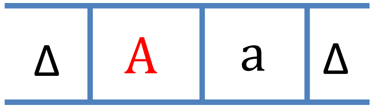


Δ	A	B	B	A	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)

   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)



Δ	A	a	Δ

Design a Turing Machine Accepting $ab^* a$

L = {ab*a}

Strings accepting in given L are:

aa

a b a

a b b a

a b b b a

Tape

Δ	A	B	A	Δ

Δ	A	B	B	A	Δ

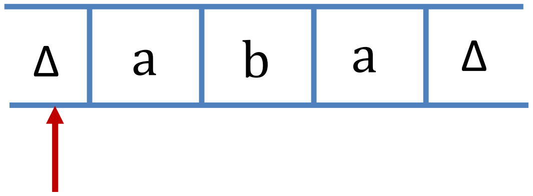
Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Read all ‘b’ one by one & replace it by ‘B’ (b→B)
   if no ‘b’ found: SKIP step 2
3. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

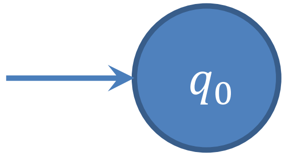
Δ	A	A	Δ

Design a Turing Machine Accepting ab^*a

Tape

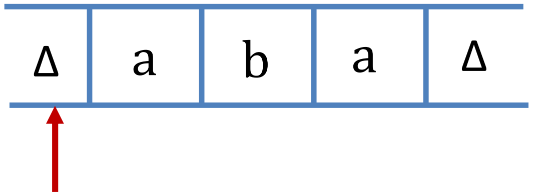


Δ	a	b	a	Δ

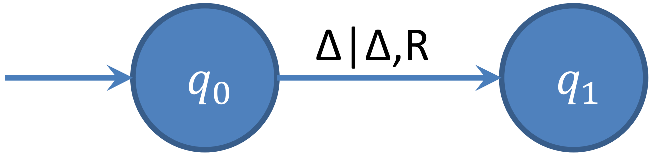


Design a Turing Machine Accepting ab^*a

Tape

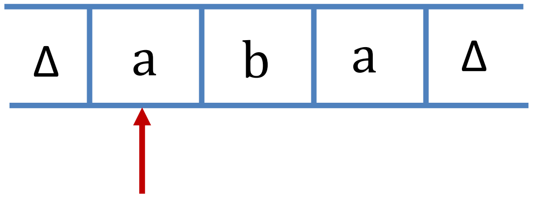


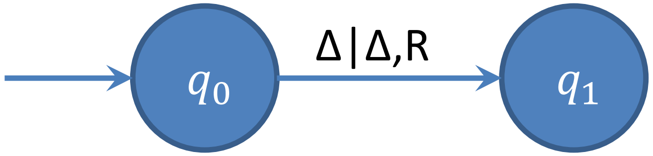
Δ	a	b	a	Δ



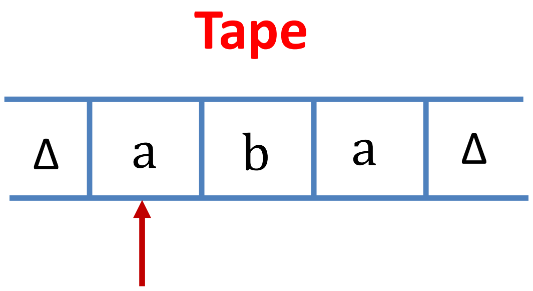
Design a Turing Machine Accepting ab^*a

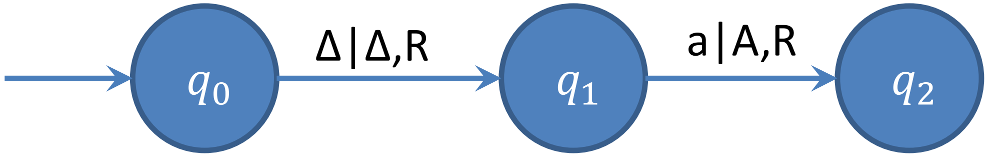
Tape





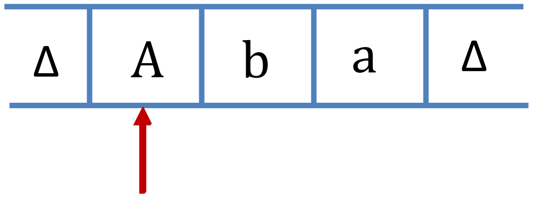
Design a Turing Machine Accepting ab^*a

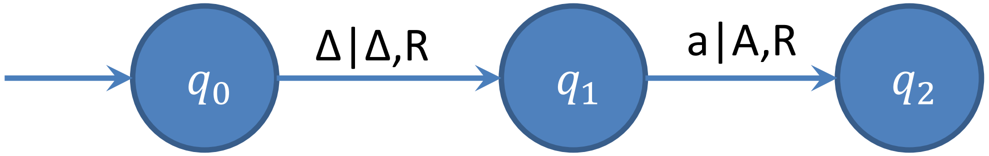




Design a Turing Machine Accepting ab^*a

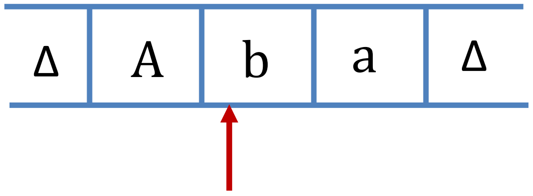
Tape



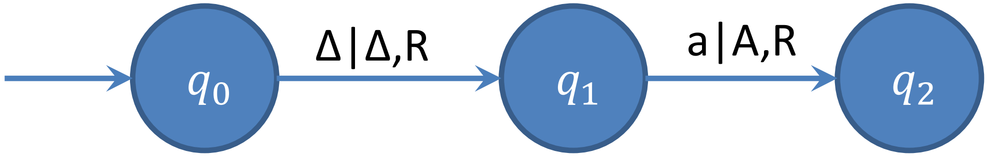


Design a Turing Machine Accepting ab^*a

Tape

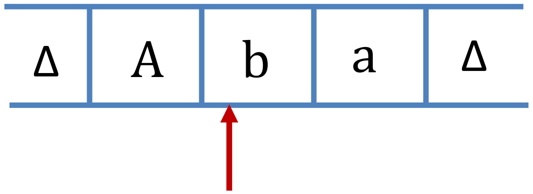


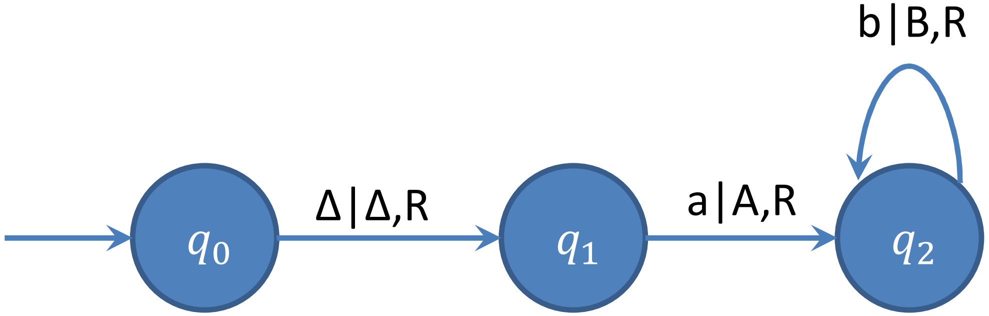
Δ	A	b	a	Δ



Design a Turing Machine Accepting ab^*a

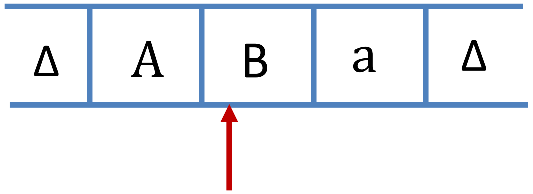
Tape



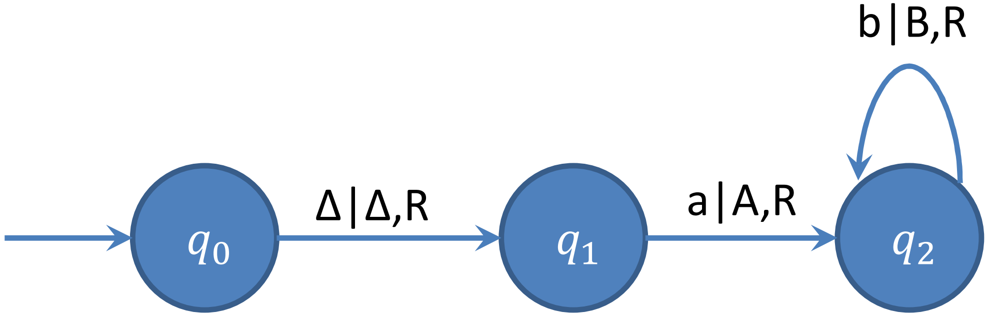


Design a Turing Machine Accepting ab^*a

Tape

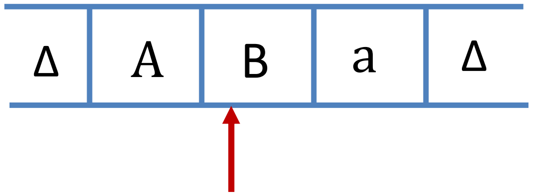


Δ	A	B	a	Δ

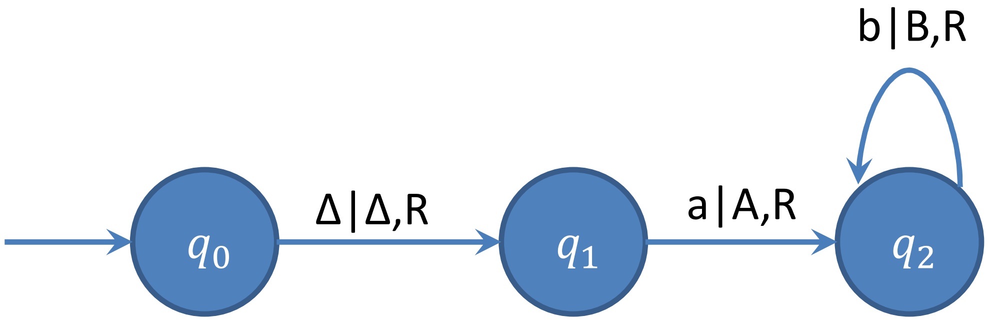


Design a Turing Machine Accepting ab^*a

Tape

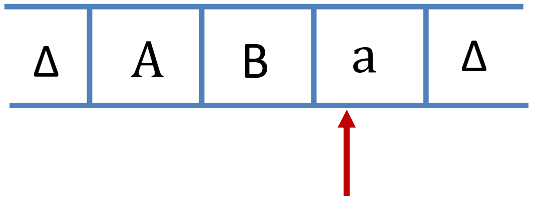


Δ	A	B	a	Δ

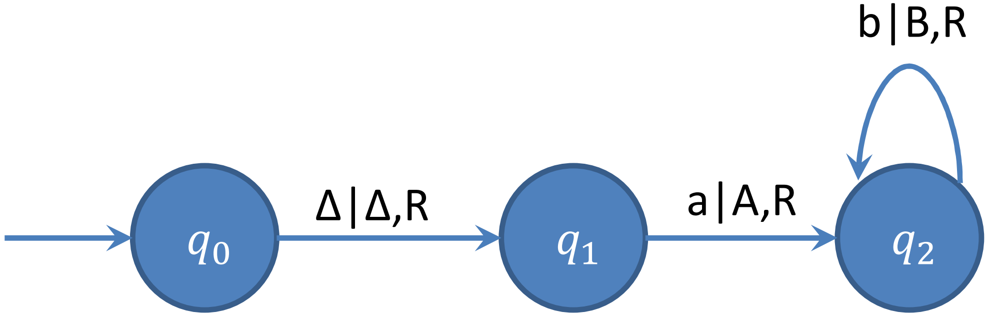


Design a Turing Machine Accepting ab^*a

Tape

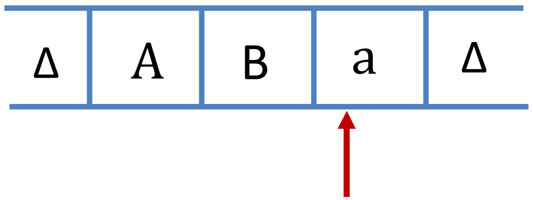


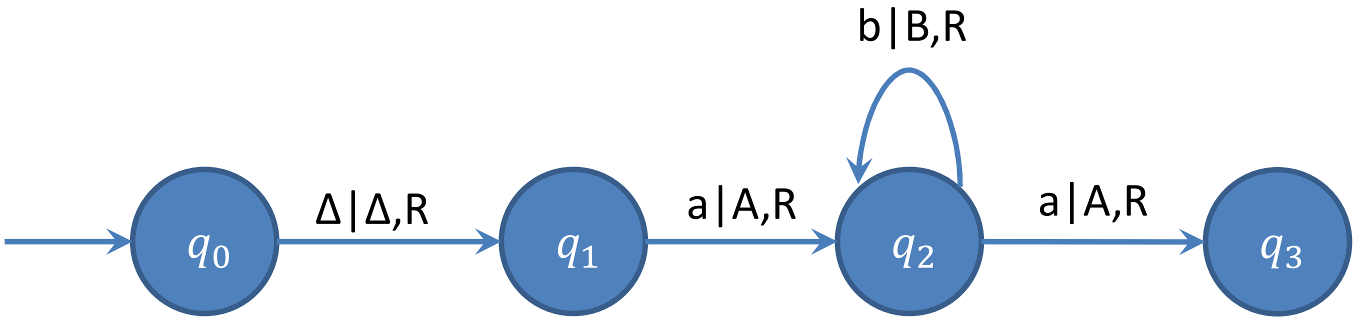
Δ	A	B	a	Δ



Design a Turing Machine Accepting ab^*a

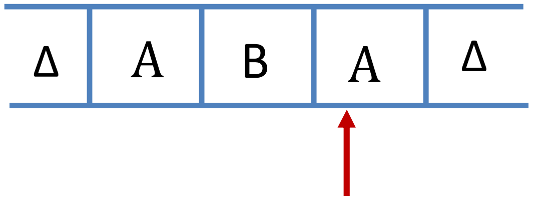
Tape



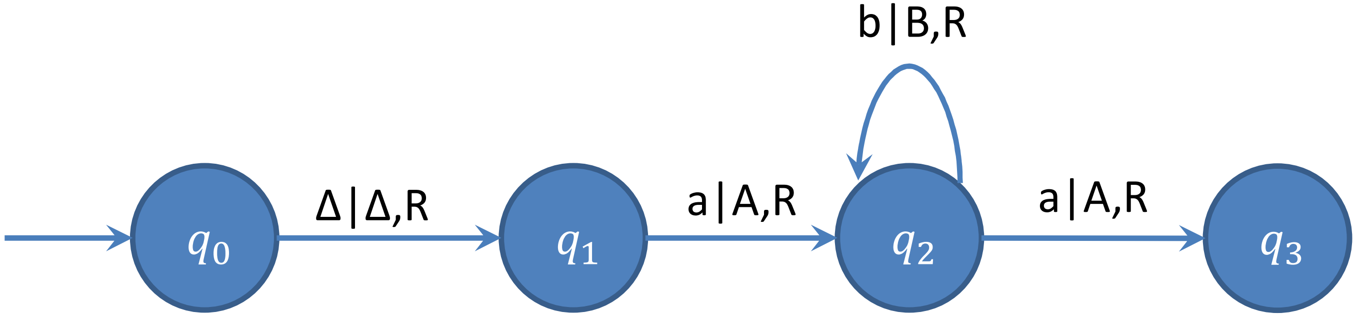


Design a Turing Machine Accepting ab^*a

Tape

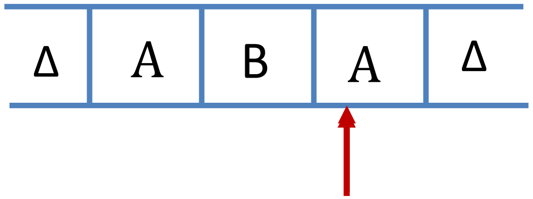


Δ	A	B	A	Δ

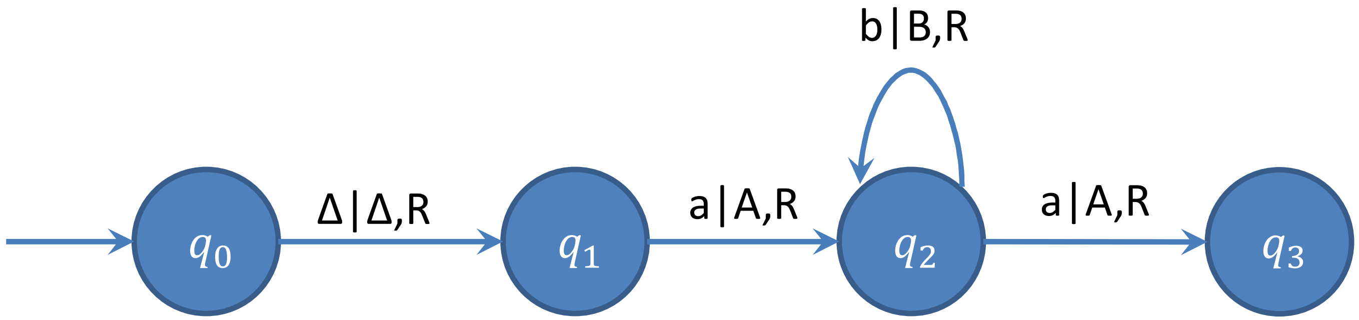


Design a Turing Machine Accepting ab^*a

Tape

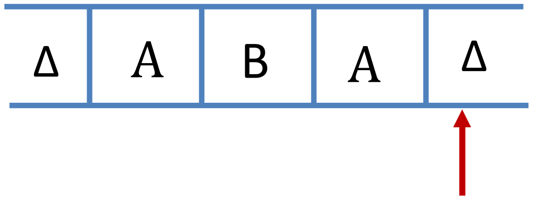


Δ	A	B	A	Δ

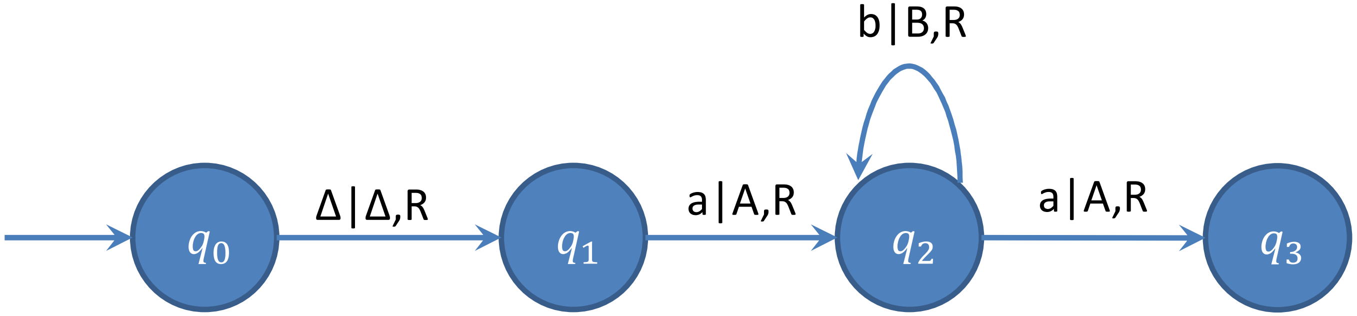


Design a Turing Machine Accepting ab^*a

Tape



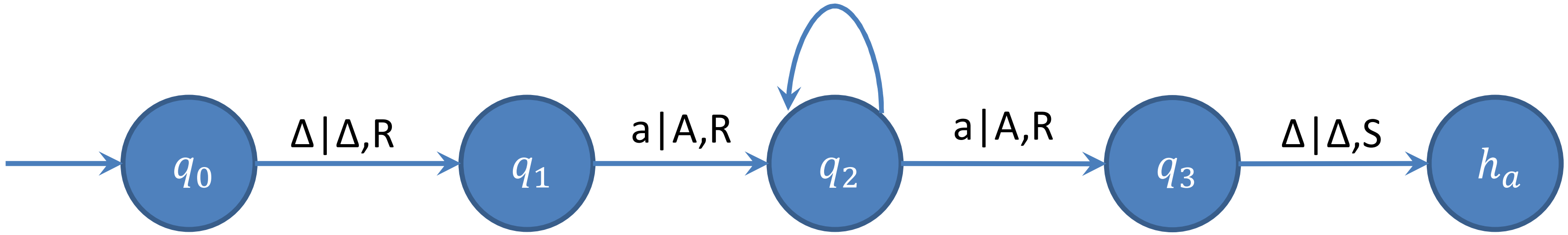
Δ	A	B	A	Δ



Design a Turing Machine Accepting ab^*a

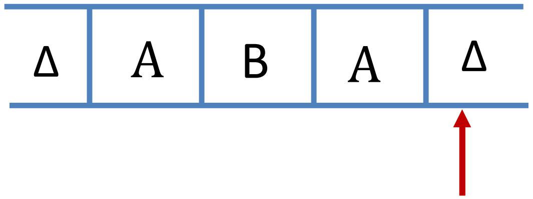
Tape

Δ	A	B	A	Δ

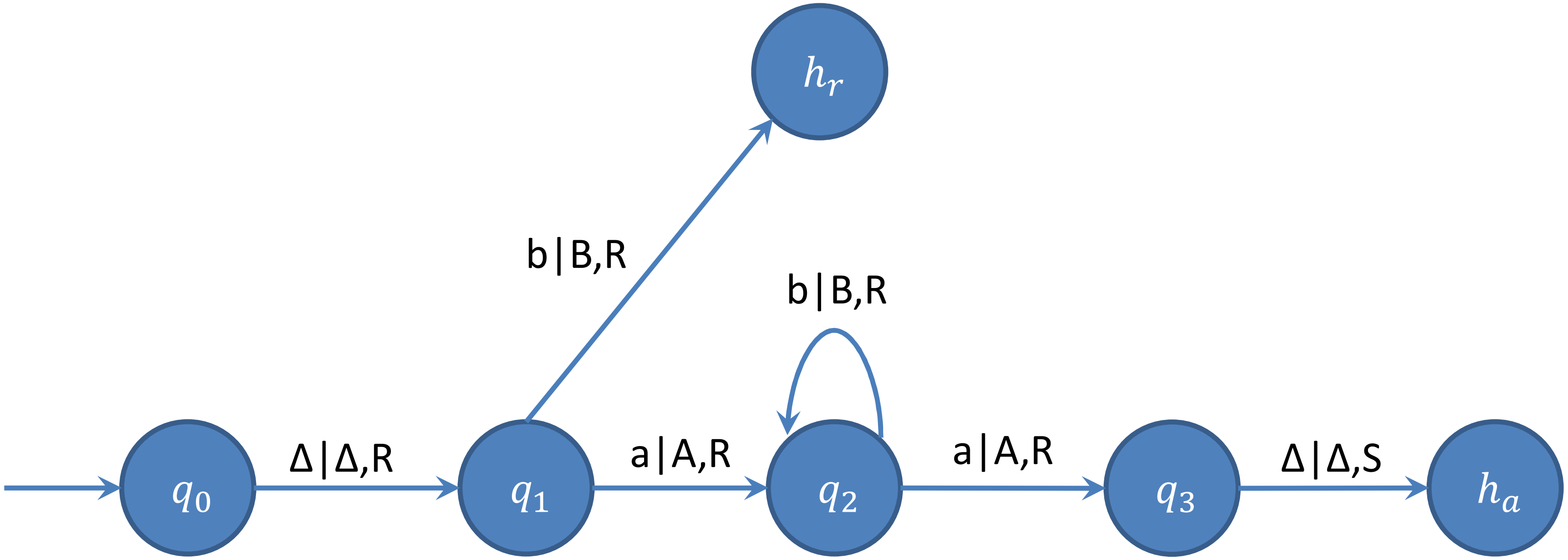


Design a Turing Machine Accepting ab^*a

Tape



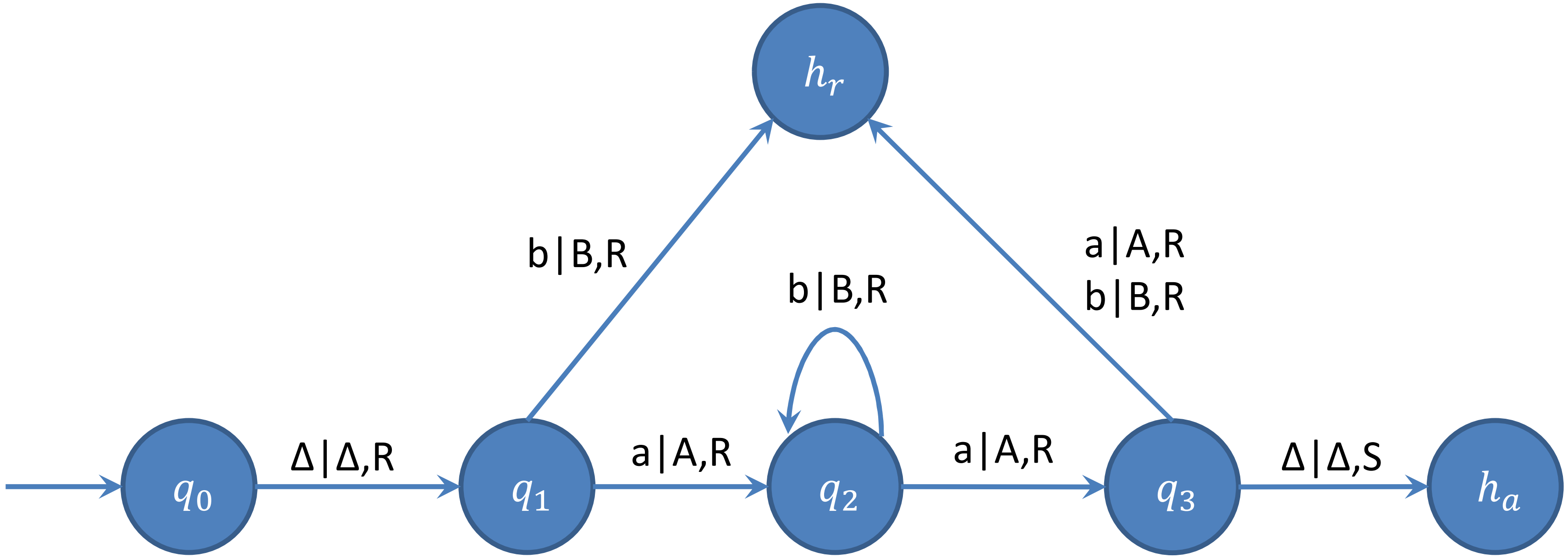
Δ	A	B	A	Δ



Design a Turing Machine Accepting ab^*a

Tape

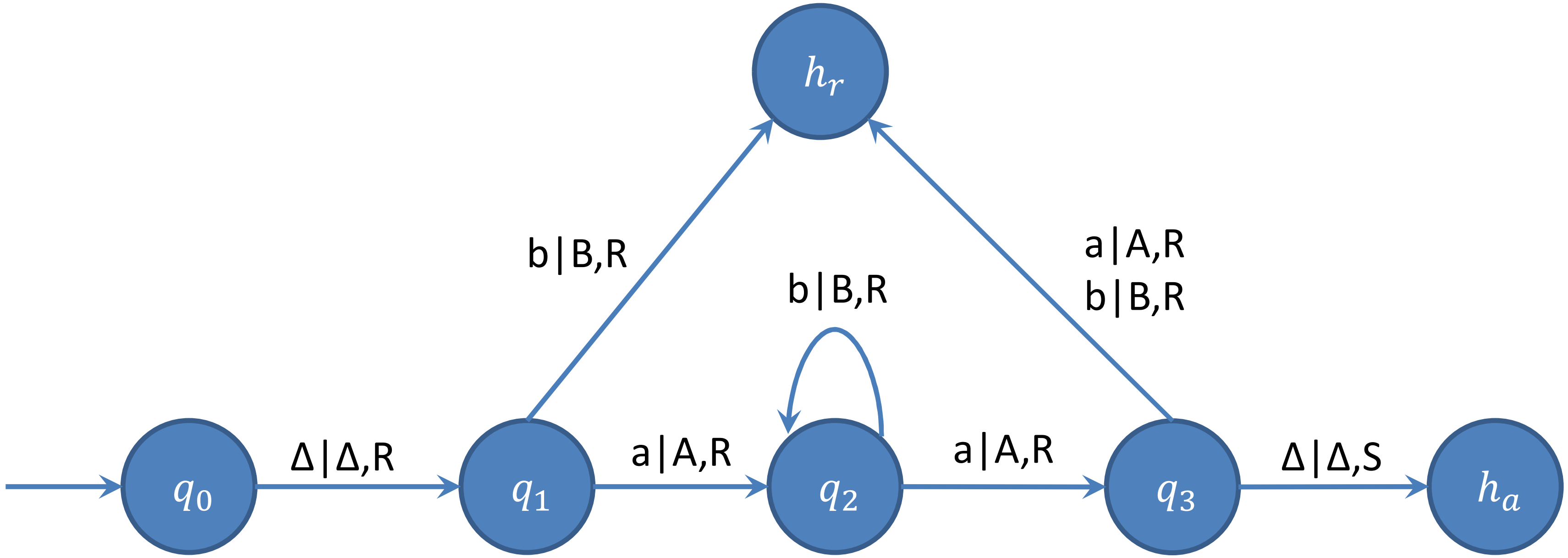
Δ	A	B	A	Δ



Design a Turing Machine Accepting ab^*a

Tape

Δ	A	B	A	Δ



Control portion

Turing Machine Accepting $\{a^n b^n | \ n \geq 1 \ \}$

Design a Turing Machine Accepting $\{a^n b^n | \ n \geq 1 \}$

L = $\{a^n b^n\}$

Tape

Strings accepting in given L are:

a b

a a b b

a a a b b b

Logic:

Design a Turing Machine Accepting $\{a^n b^n \mid n \geq 1\}$

L = $\{a^n b^n\}$

Strings accepting in given L are:

a b

a a b b

a a a b b b

Logic:

Tape

Δ	a	a	b	b	Δ

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1 \}$

L = {$a^n b^n$}

Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape

Δ	a	a	b	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting $\{a^n b^n | \ n \geq 1 \ \}$

L = $\{a^n b^n\}$

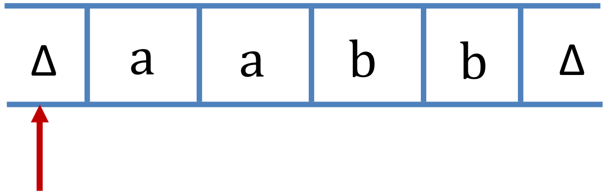
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting $\{a^n b^n \mid n \geq 1\}$

L = $\{a^n b^n\}$

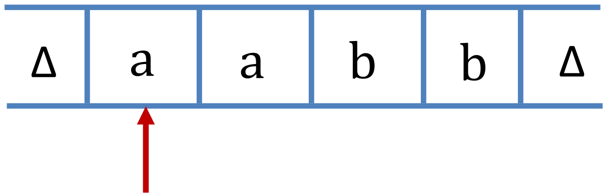
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	a	a	b	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting $\{a^n b^n | \ n \geq 1 \ \}$

L = $\{a^n b^n\}$

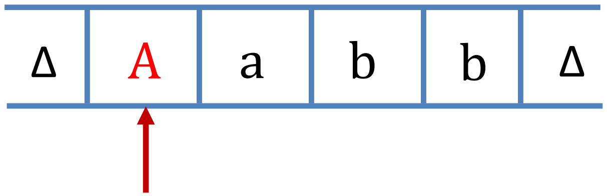
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	a	b	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)

Design a Turing Machine Accepting $\{a^n b^n | \ n \geq 1 \}$

L = $\{a^n b^n\}$

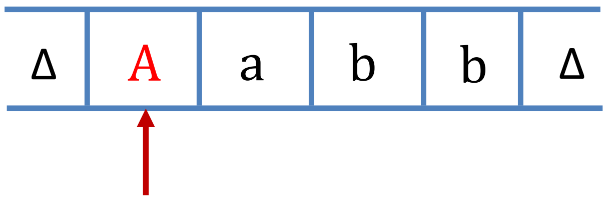
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	a	b	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1 \}$

L = $\{a^n b^n\}$

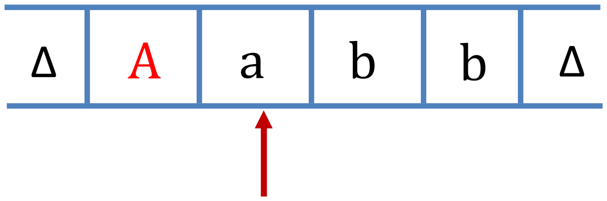
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1 \}$

L = $\{a^n b^n\}$

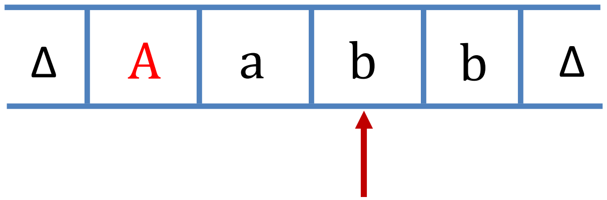
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	a	b	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT

Design a Turing Machine Accepting $\{a^n b^n | \ n \geq 1 \ \}$

L = {$a^n b^n$}

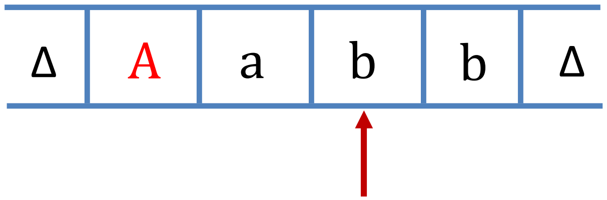
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	a	b	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1 \}$

L = $\{a^n b^n\}$

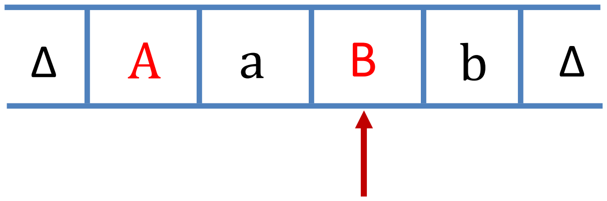
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	a	B	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)

Design a Turing Machine Accepting $\{a^n b^n | \ n \geq 1 \}$

L = {$a^n b^n$}

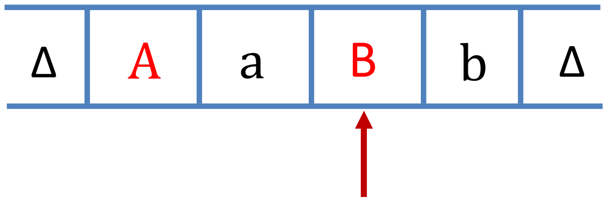
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’

Design a Turing Machine Accepting $\{a^n b^n | \ n \geq 1 \ \}$

L = $\{a^n b^n\}$

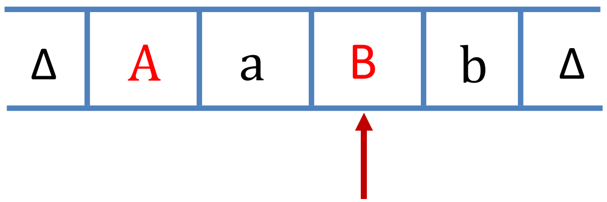
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	a	B	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’
   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s

Design a Turing Machine Accepting $\{a^n b^n \mid n \geq 1\}$

L = {$a^n b^n$}

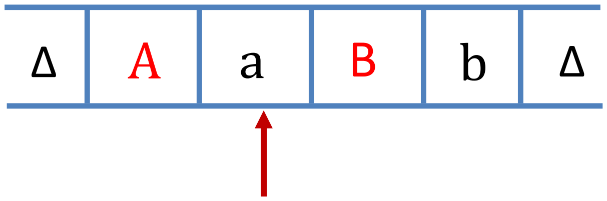
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	a	B	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’
   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s

Design a Turing Machine Accepting $\{a^n b^n \mid n \geq 1\}$

L = $\{a^n b^n\}$

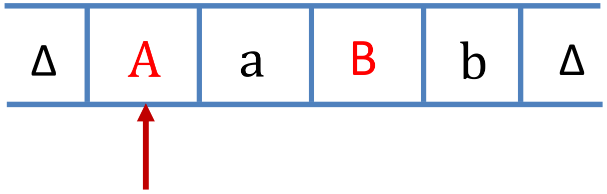
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	a	B	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’
   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1 \}$

L = $\{a^n b^n\}$

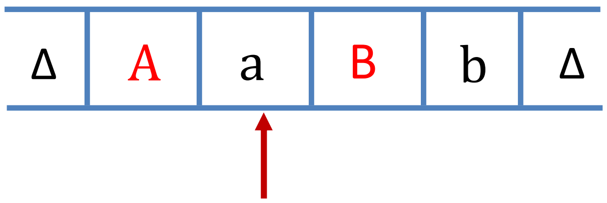
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	a	B	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1 \}$

L = {$a^n b^n$}

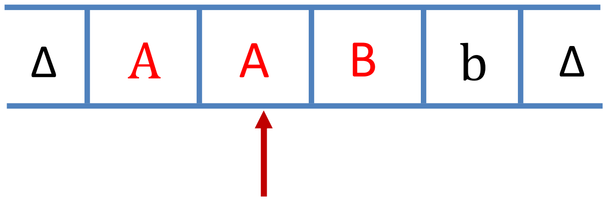
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	A	B	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s

Design a Turing Machine Accepting $\{a^n b^n |\, n \geq 1\,\}$

L = $\{a^n b^n\}$

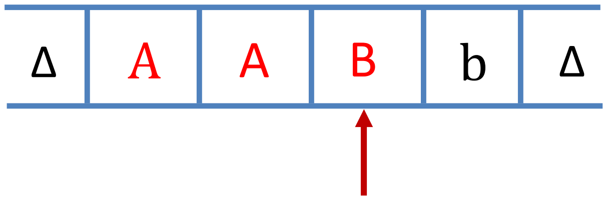
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	A	B	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1 \}$

L = $\{a^n b^n\}$

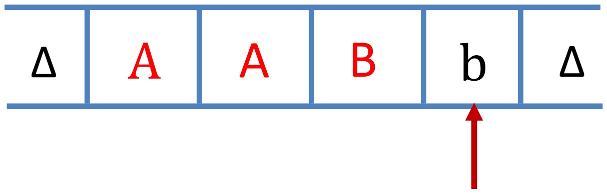
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	A	B	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’
   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1 \}$

L = $\{a^n b^n\}$

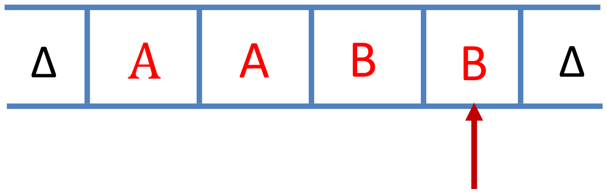
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	A	B	B	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1 \}$

L = {$a^n b^n$}

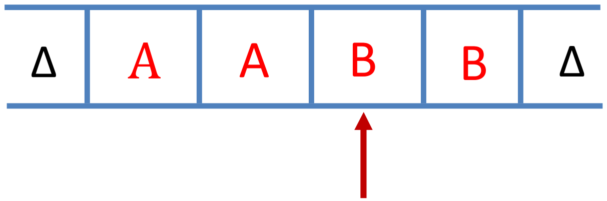
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1 \}$

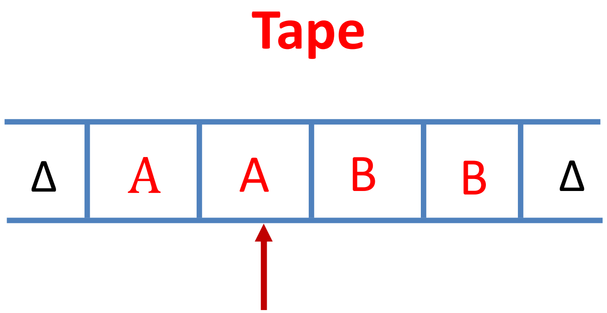
$L = \{a^n b^n\}$

Strings accepting in given L are:

a b

a a b b

a a a b b b



Tape

Δ	A	A	B	B	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s

Design a Turing Machine Accepting $\{a^n b^n | \ n \geq 1 \ \}$

L = {a^{n}b^n}

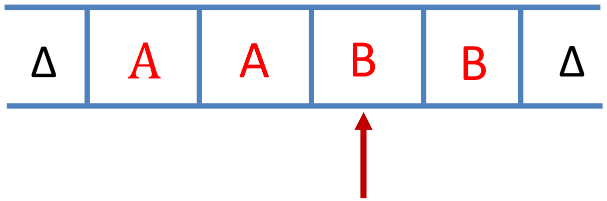
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	A	B	B	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’
   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1 \}$

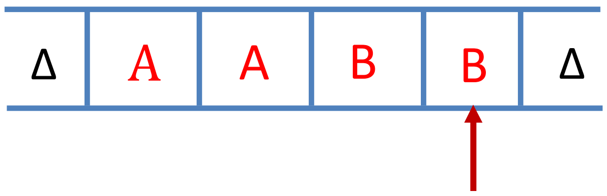
L = $\{a^n b^n\}$

Strings accepting in given L are:

a b

a a b b

a a a b b b



Tape

Δ	A	A	B	B	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’
   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1 \}$

L = $\{a^n b^n\}$

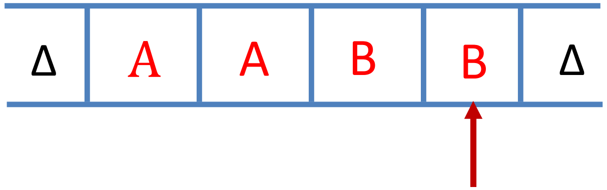
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	A	B	B	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s
6. Make sure no more b’s remain.

Design a Turing Machine Accepting $\{a^n b^n | \; n \geq 1 \}$

L = $\{a^n b^n\}$

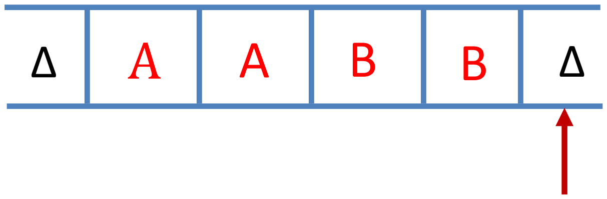
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	A	B	B	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s
6. Make sure no more b’s remain.

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1\}$

L = $\{a^n b^n\}$

Strings accepting in given L are:

a b

a a b b

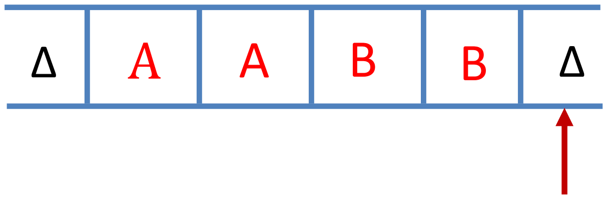
a a a b b b

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s
6. Make sure no more b’s remain.

Tape



Δ	A	A	B	B	Δ

Δ	a	a	a	b	b	b	Δ

Design a Turing Machine Accepting $\{a^n b^n |\ n \geq 1\ \}$

L = $\{a^n b^n\}$

Strings accepting in given L are:

a b

a a b b

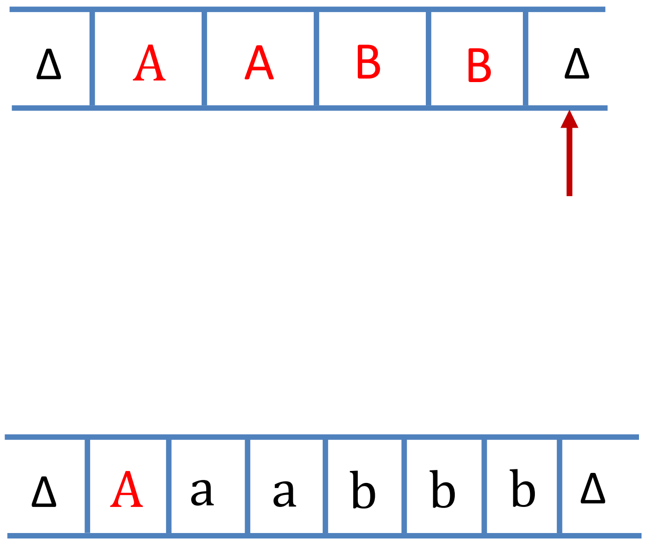
a a a b b b

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’

   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s
6. Make sure no more b’s remain.

Tape



Δ	A	A	B	B	Δ

Δ	A	a	a	b	b	b	Δ

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1 \}$

L = {$a^n b^n$}

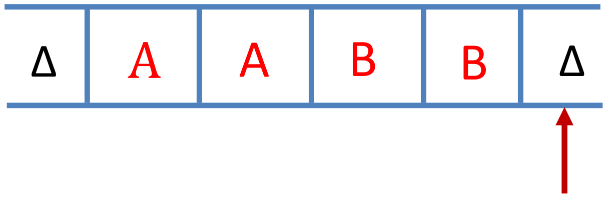
Strings accepting in given L are:

a b

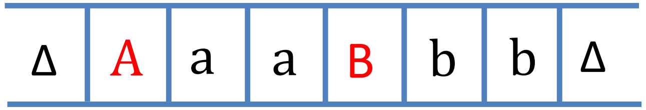
a a b b

a a a b b b

Tape



Δ	A	A	B	B	Δ



Δ	A	a	a	B	b	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’
   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s
6. Make sure no more b’s remain.

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1 \}$

L = {$a^n b^n$}

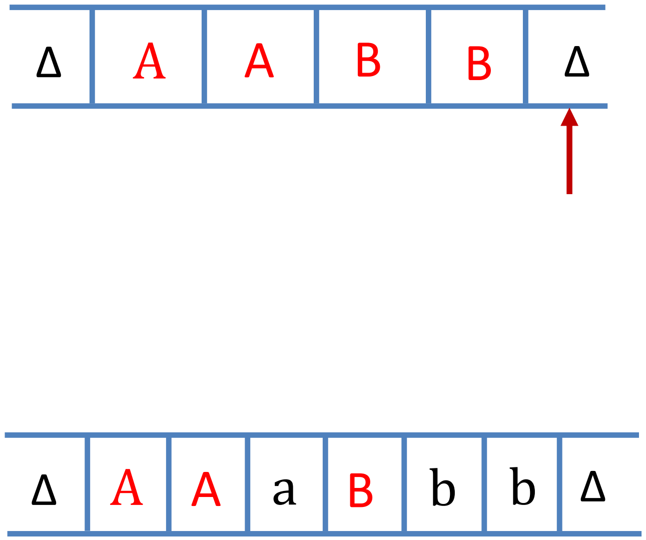
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ	A	A	B	B	Δ

Δ	A	A	a	B	b	b	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’
   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s
6. Make sure no more b’s remain.

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1 \}$

L = {$a^n b^n$}

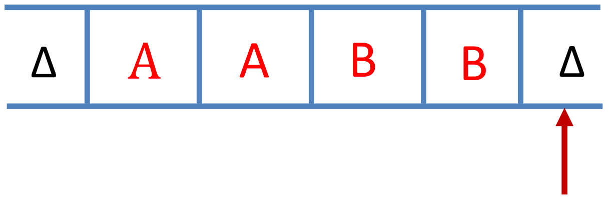
Strings accepting in given L are:

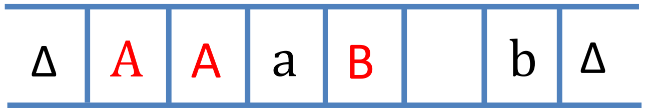
a b

a a b b

a a a b b b

Tape





Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’
   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s
6. Make sure no more b’s remain.

Design a Turing Machine Accepting $\{a^n b^n | \ n \geq 1 \}$

L = {a^{n}b^n}

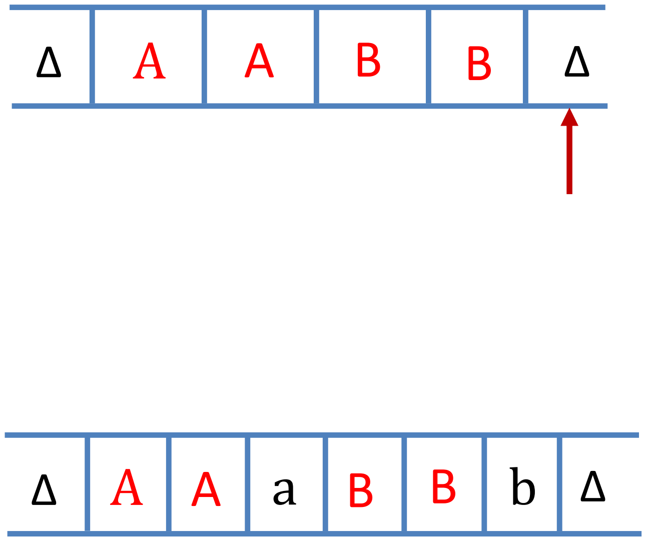
Strings accepting in given L are:

a b

a a b b

a a a b b b

Tape



Δ A A B B Δ

Δ A A a B B b Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’
   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s
6. Make sure no more b’s remain.

Design a Turing Machine Accepting $\{a^n b^n | \ n \geq 1 \ \}$

L = {$a^n b^n$}

Strings accepting in given L are:

a b

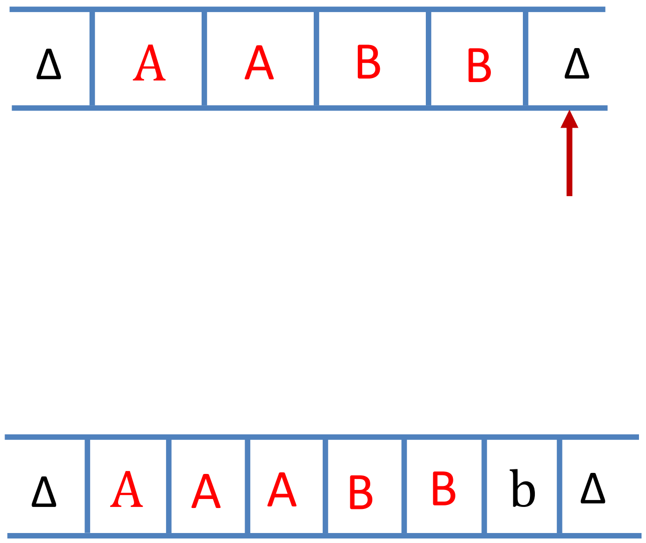
a a b b

a a a b b b

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’
   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s
6. Make sure no more b’s remain.

Tape



Δ	A	A	B	B	Δ

Δ	A	A	A	B	B	b	Δ

Design a Turing Machine Accepting $\{a^n b^n | \ n \geq 1 \ \}$

L = $\{a^n b^n\}$

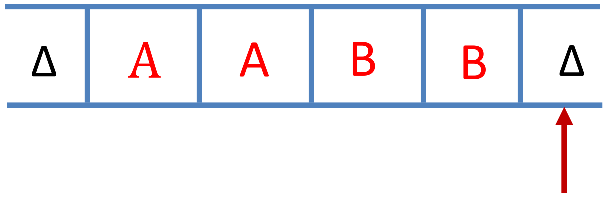
Strings accepting in given L are:

a b

a a b b

a a a b b b

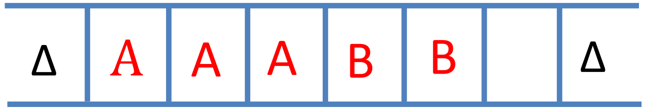
Tape



Δ	A	A	B	B	Δ

Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’
   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s
6. Make sure no more b’s remain.



Δ	A	A	A	B	B		Δ

Design a Turing Machine Accepting $\{a^n b^n | \, n \geq 1\}$

L = {$a^n b^n$}

Strings accepting in given L are:

a b

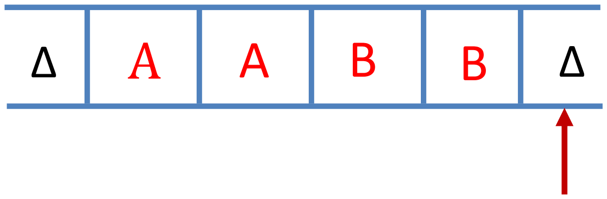
a a b b

a a a b b b

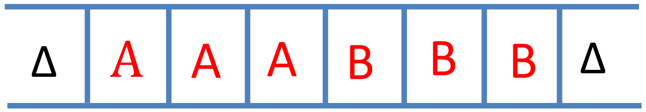
Logic:

1. Read ‘a’, replace ‘a’ by ‘A’ (a→A)
2. Move RIGHT to first ‘b’
   if none : REJECT
3. Replace ‘b’ by ‘B’ (b→B)
4. Move LEFT to leftmost ‘a’
5. Repeat the above steps until no more a’s
6. Make sure no more b’s remain.

Tape



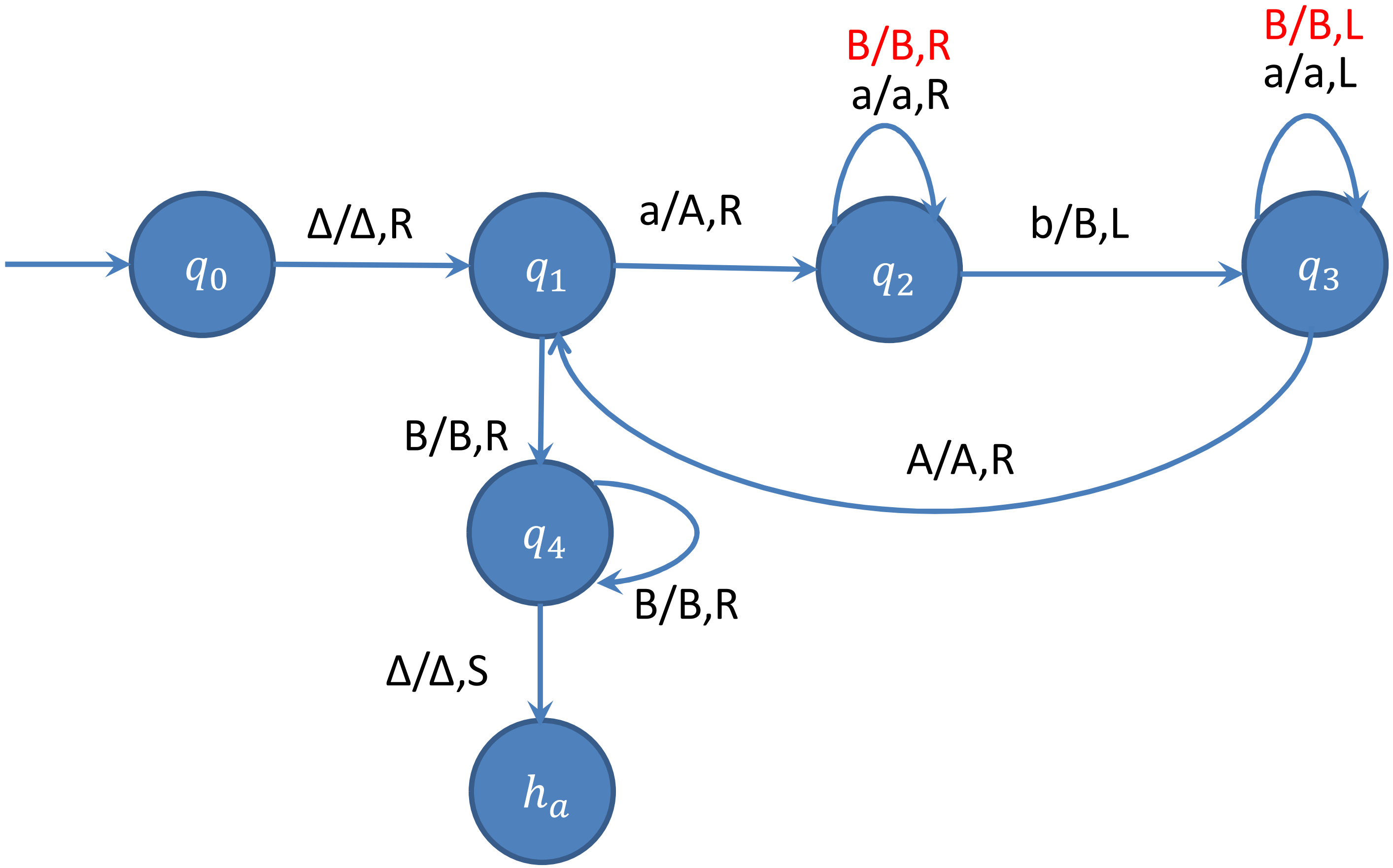
Δ	A	A	B	B	Δ



Δ	A	A	A	B	B	B	Δ

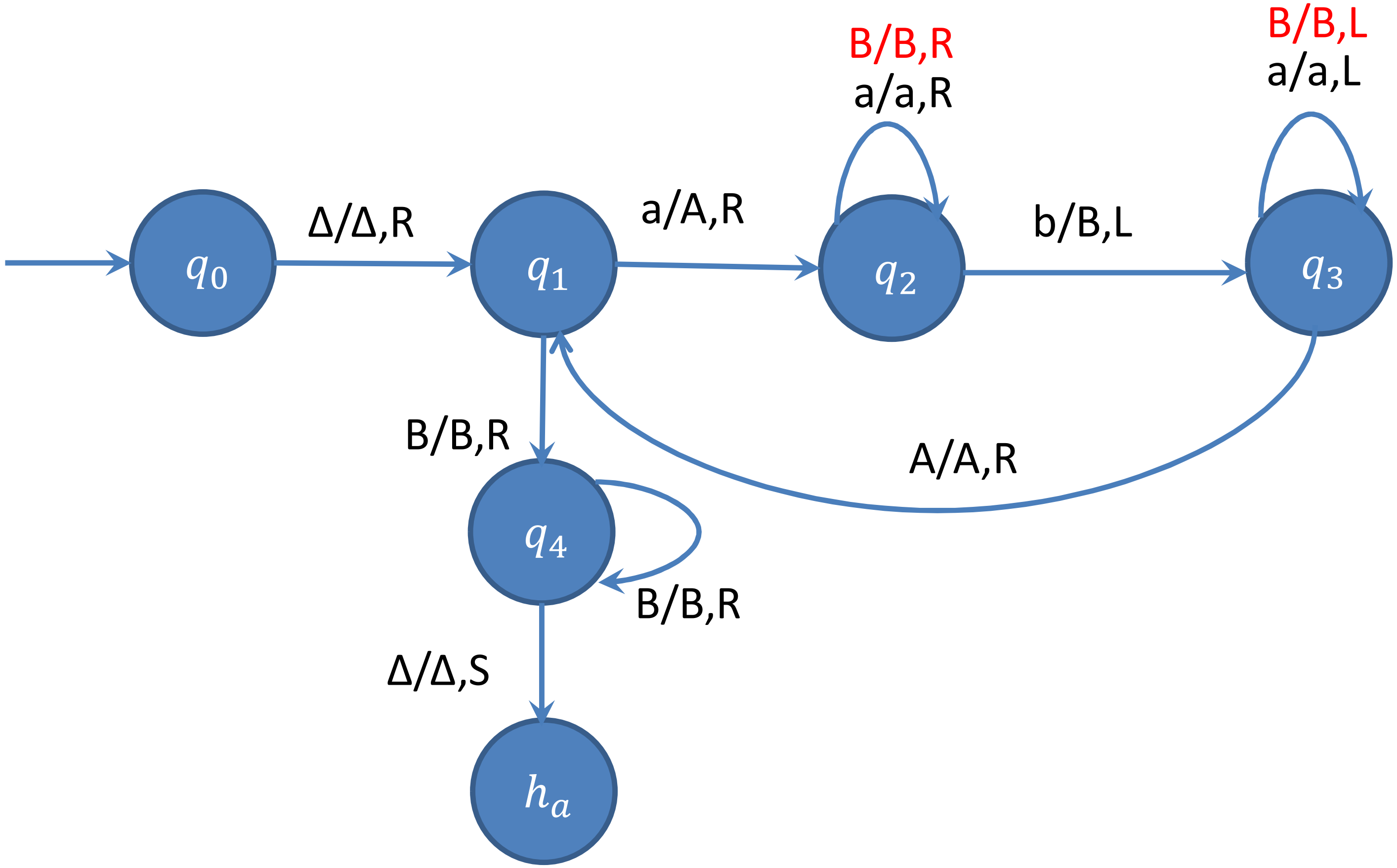
Design a Turing machine for accepting $\{a^n b^n \mid n \geq 1\}$

Δ	a	a	b	b	Δ

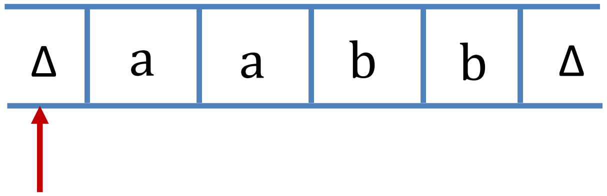


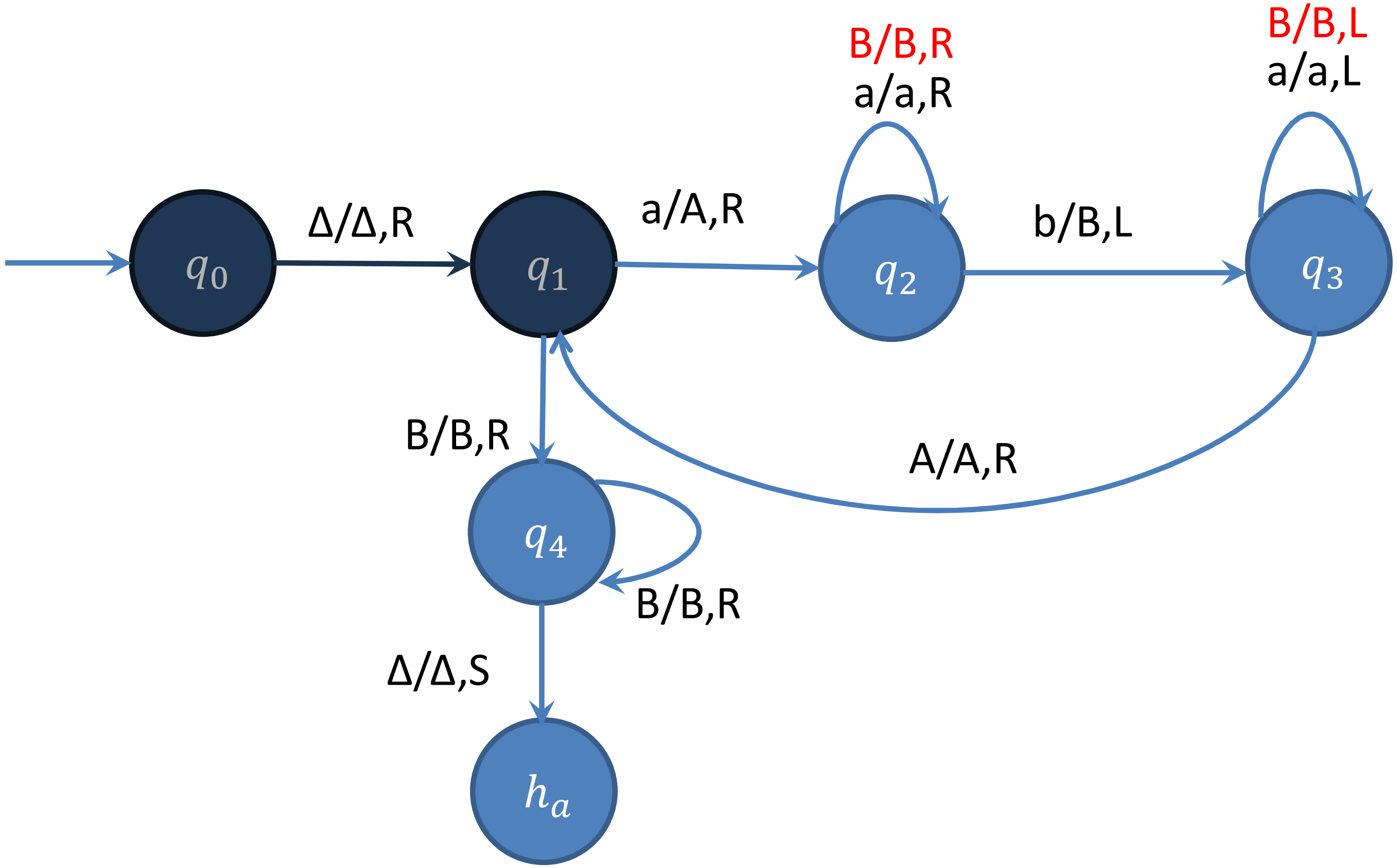
Design a Turing machine for accepting $\{a^n b^n | \, n \geq 1 \}$

Δ	a	a	b	b	Δ

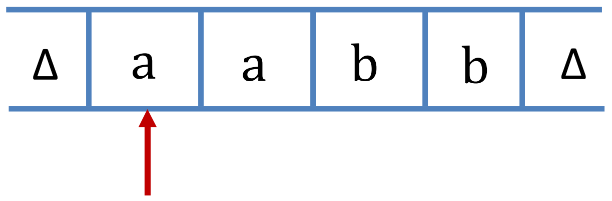


Design a Turing machine for accepting $\{a^n b^n | \, n \geq 1 \}$

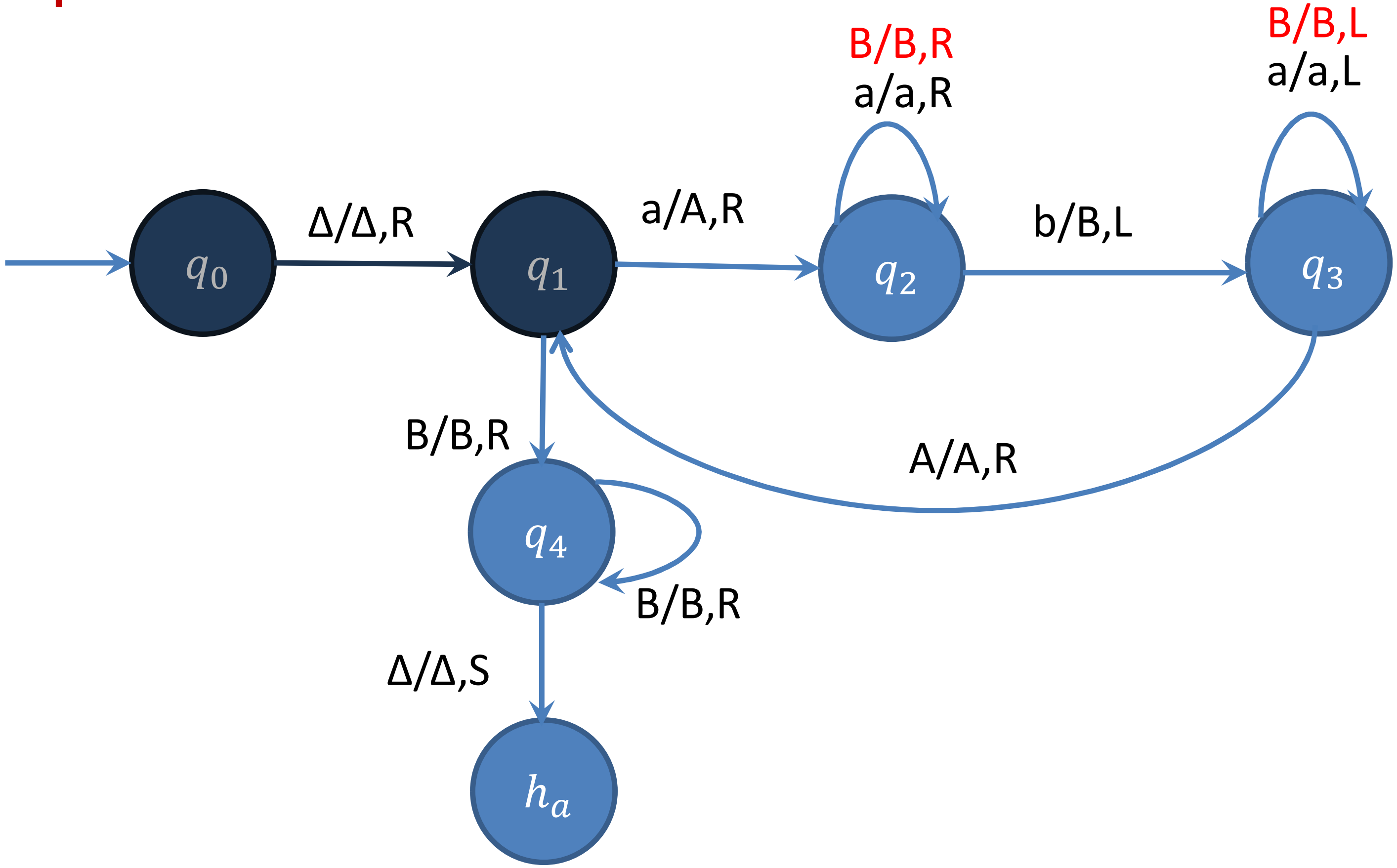




Design a Turing machine for accepting $\{a^n b^n |\ n \geq 1\}$

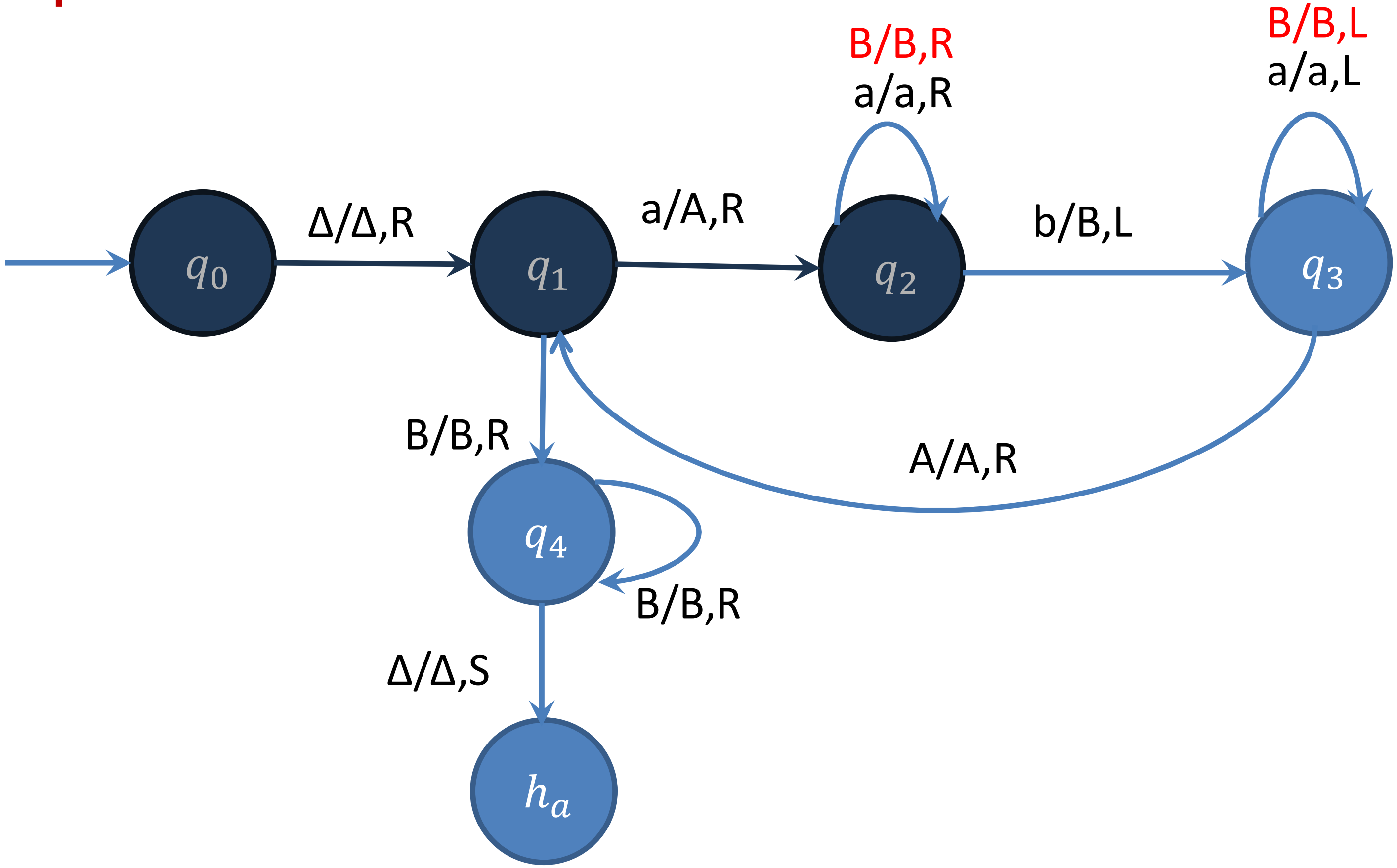


Δ	a	a	b	b	Δ



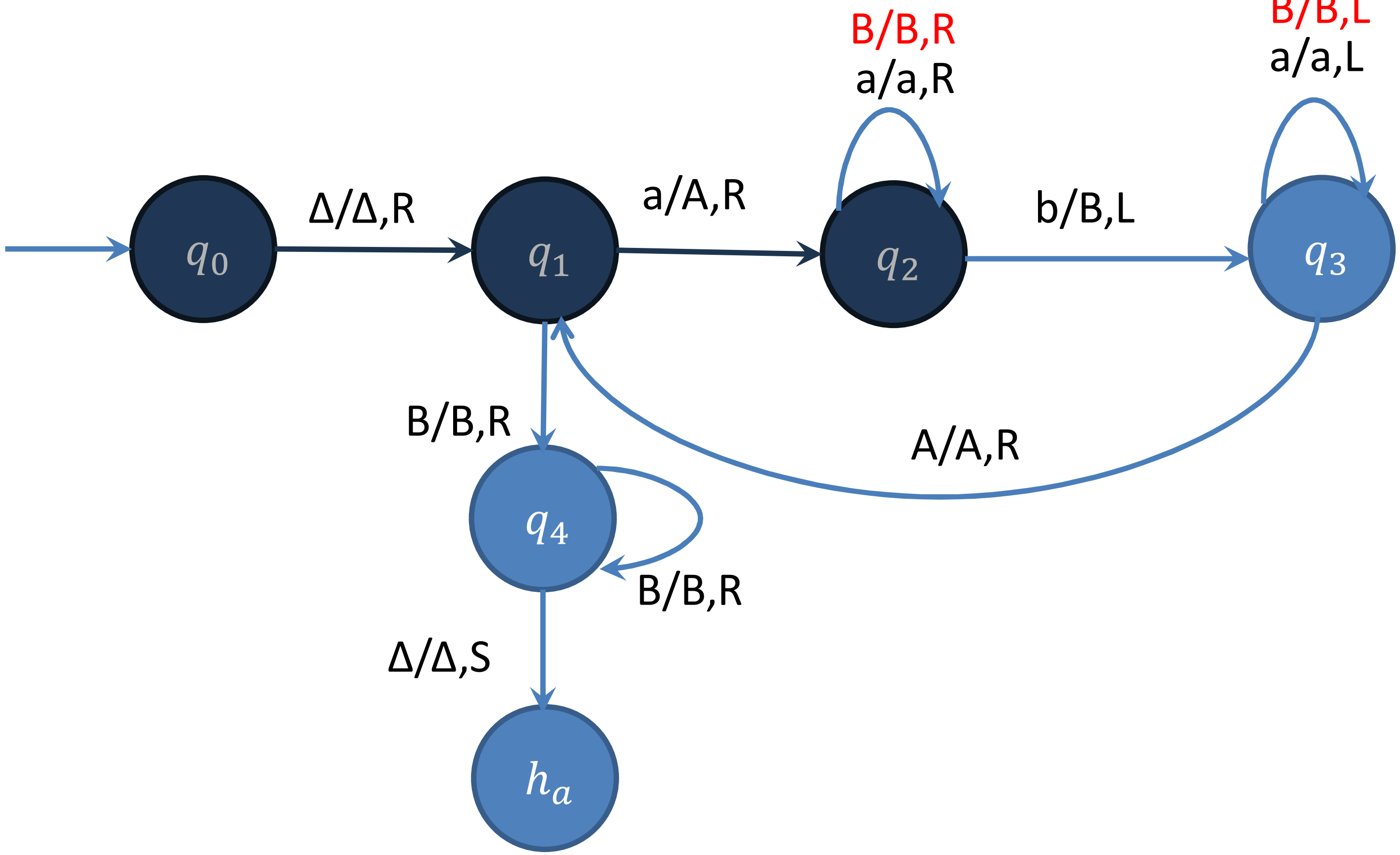
Design a Turing machine for accepting $\{a^n b^n | \, n \geq 1 \}$

Δ	a	a	b	b	Δ



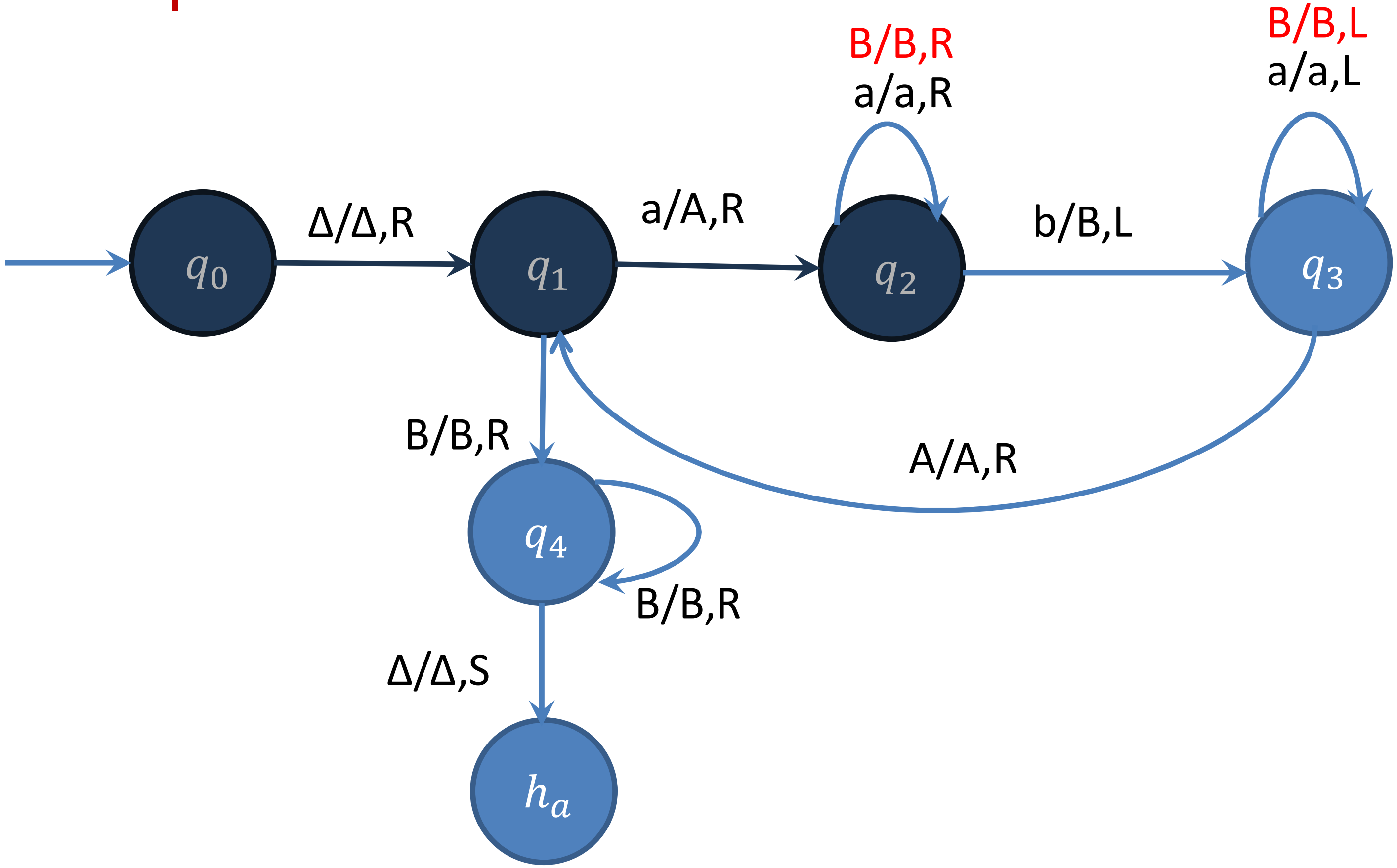
Design a Turing machine for accepting $\{a^n b^n |\ n \geq 1\ \}$

Δ	A	a	b	b	Δ

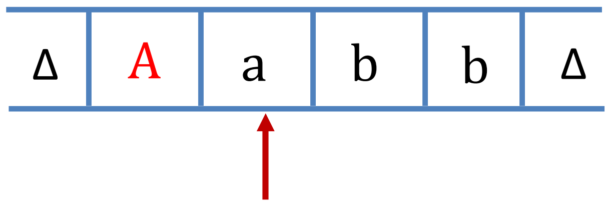


Design a Turing machine for accepting $\{a^n b^n | \, n \geq 1\}$

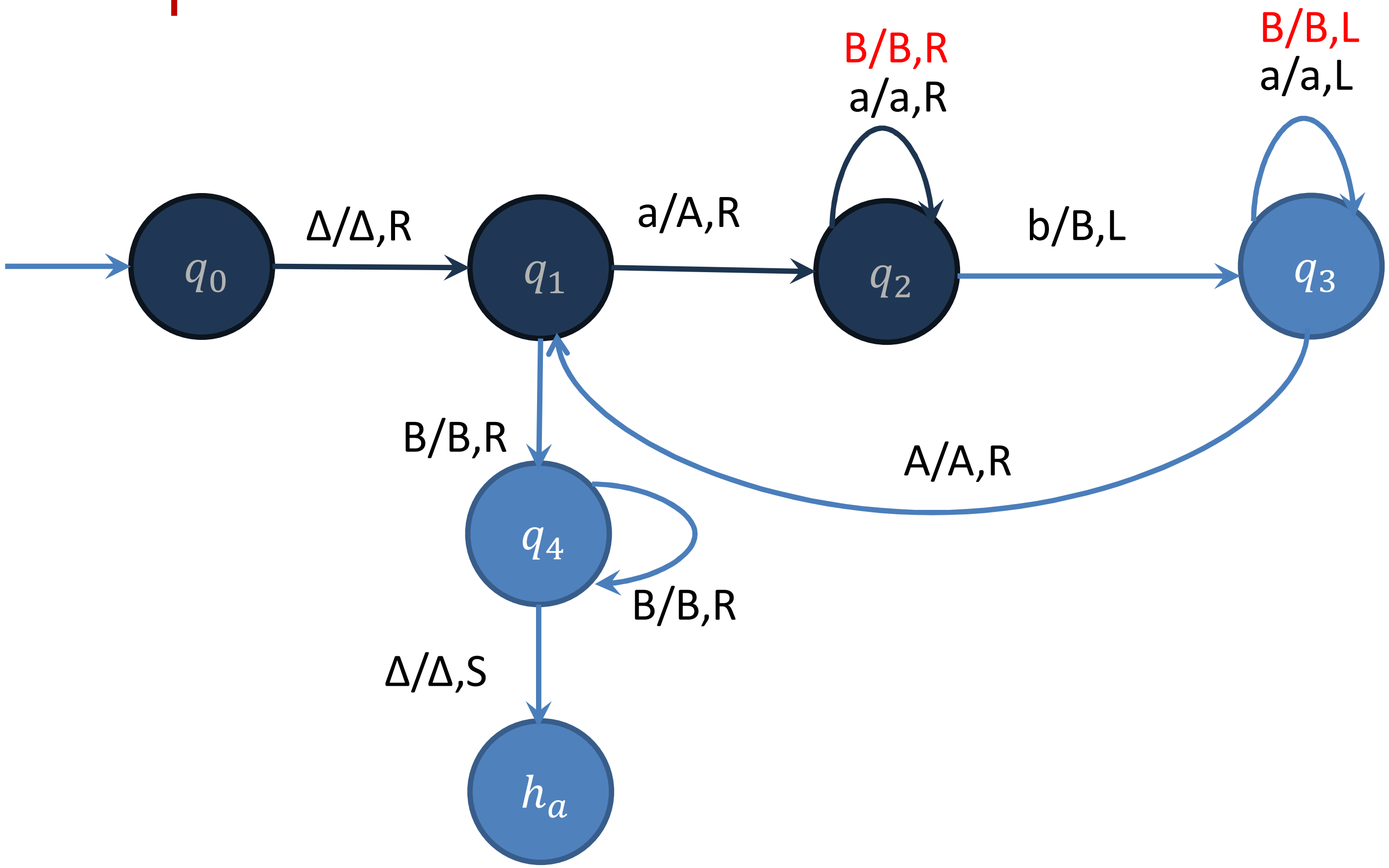
Δ	A	a	b	b	Δ



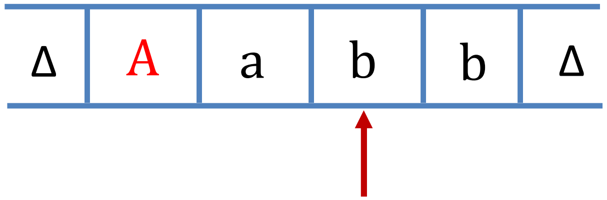
Design a Turing machine for accepting $\{a^n b^n | \, n \geq 1 \}$



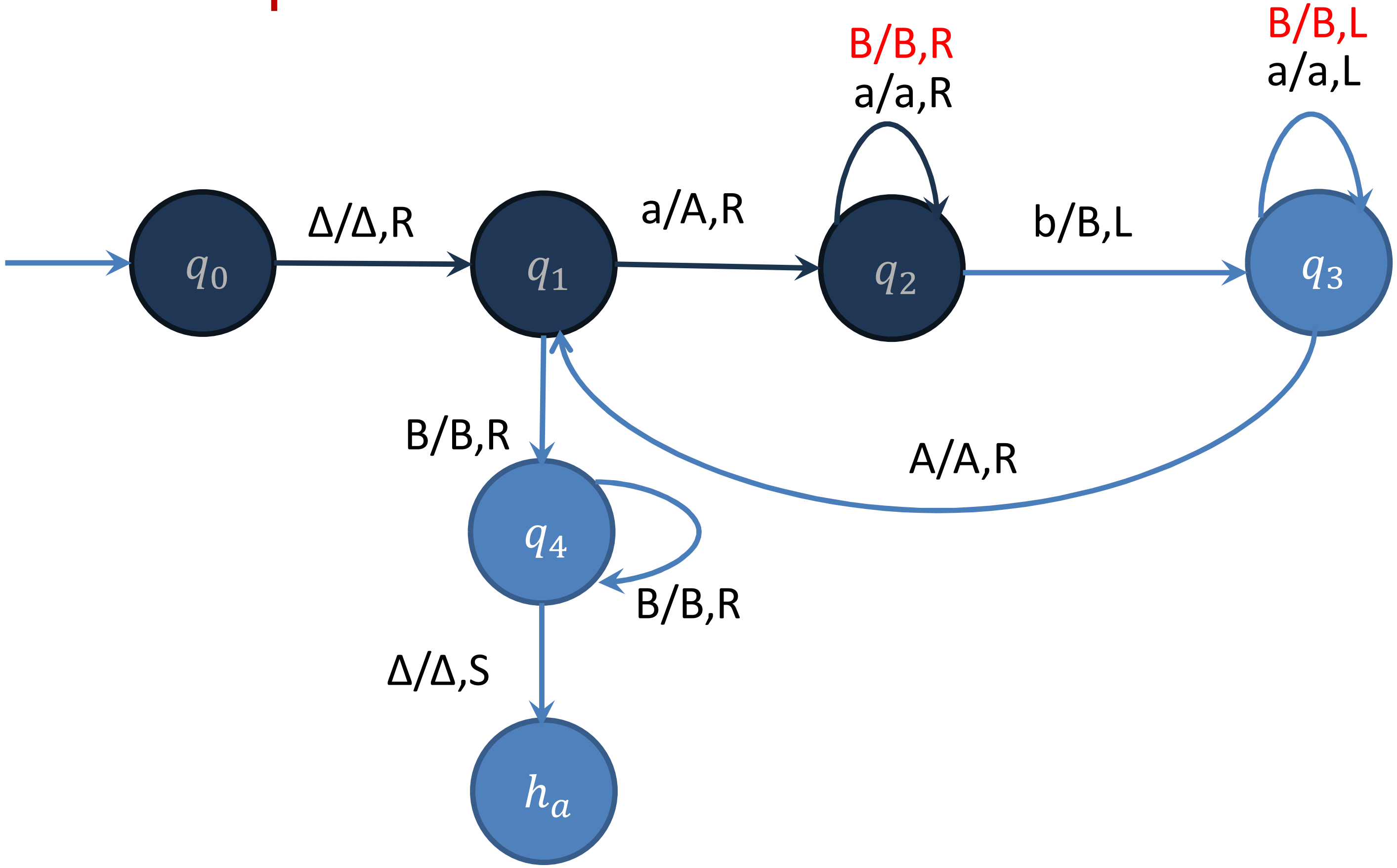
Δ	A	a	b	b	Δ



Design a Turing machine for accepting $\{a^n b^n \mid n \geq 1\}$

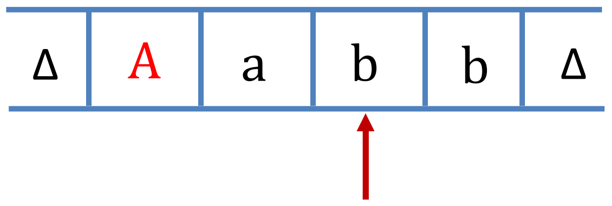


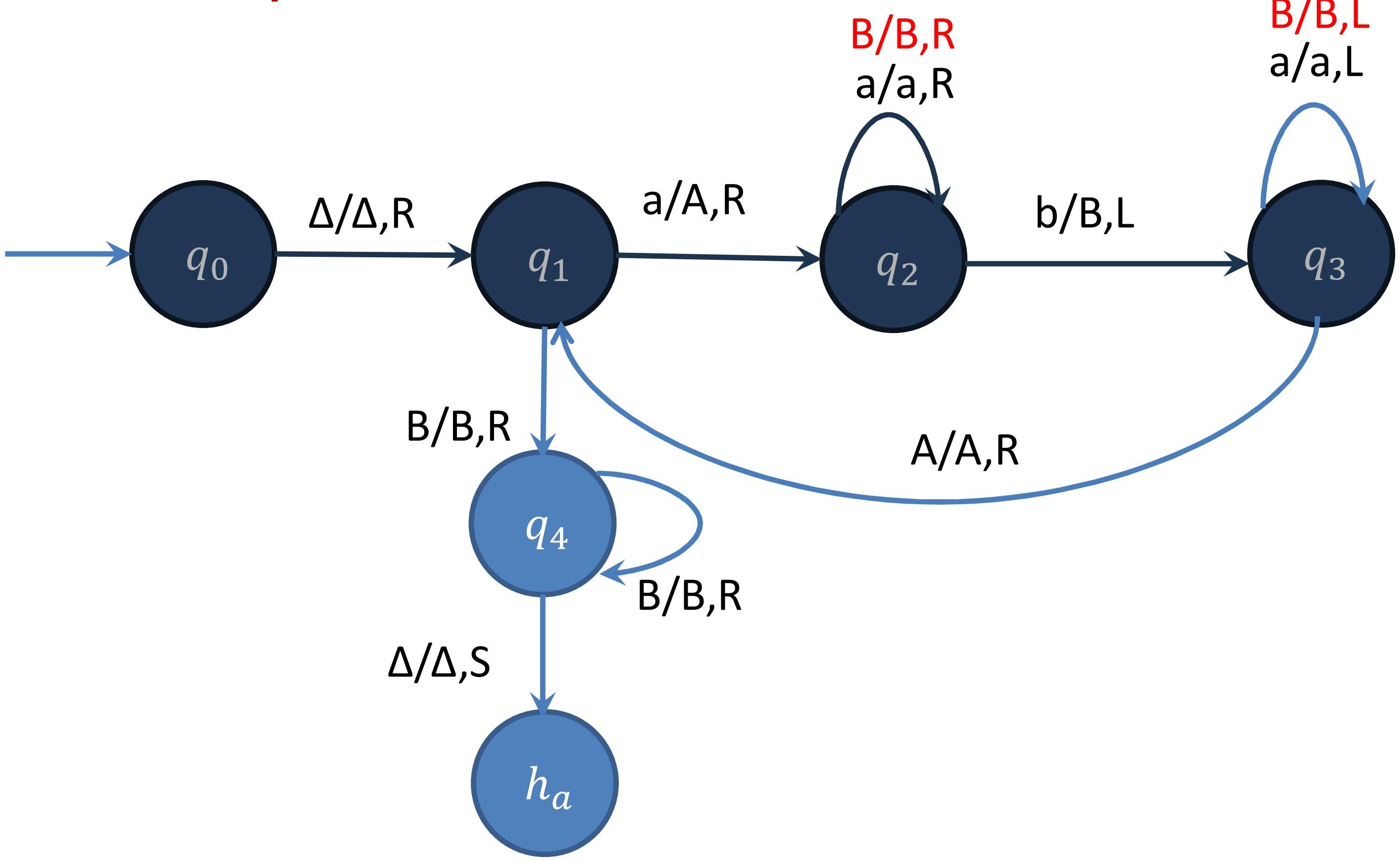
Δ	A	a	b	b	Δ



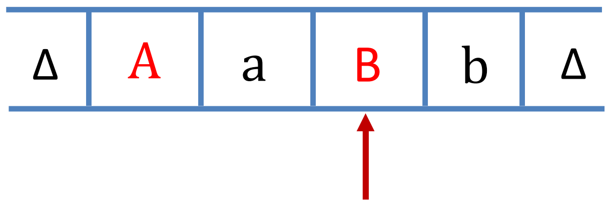
Design a Turing machine for accepting $\{a^n b^n | \, n \geq 1\}$

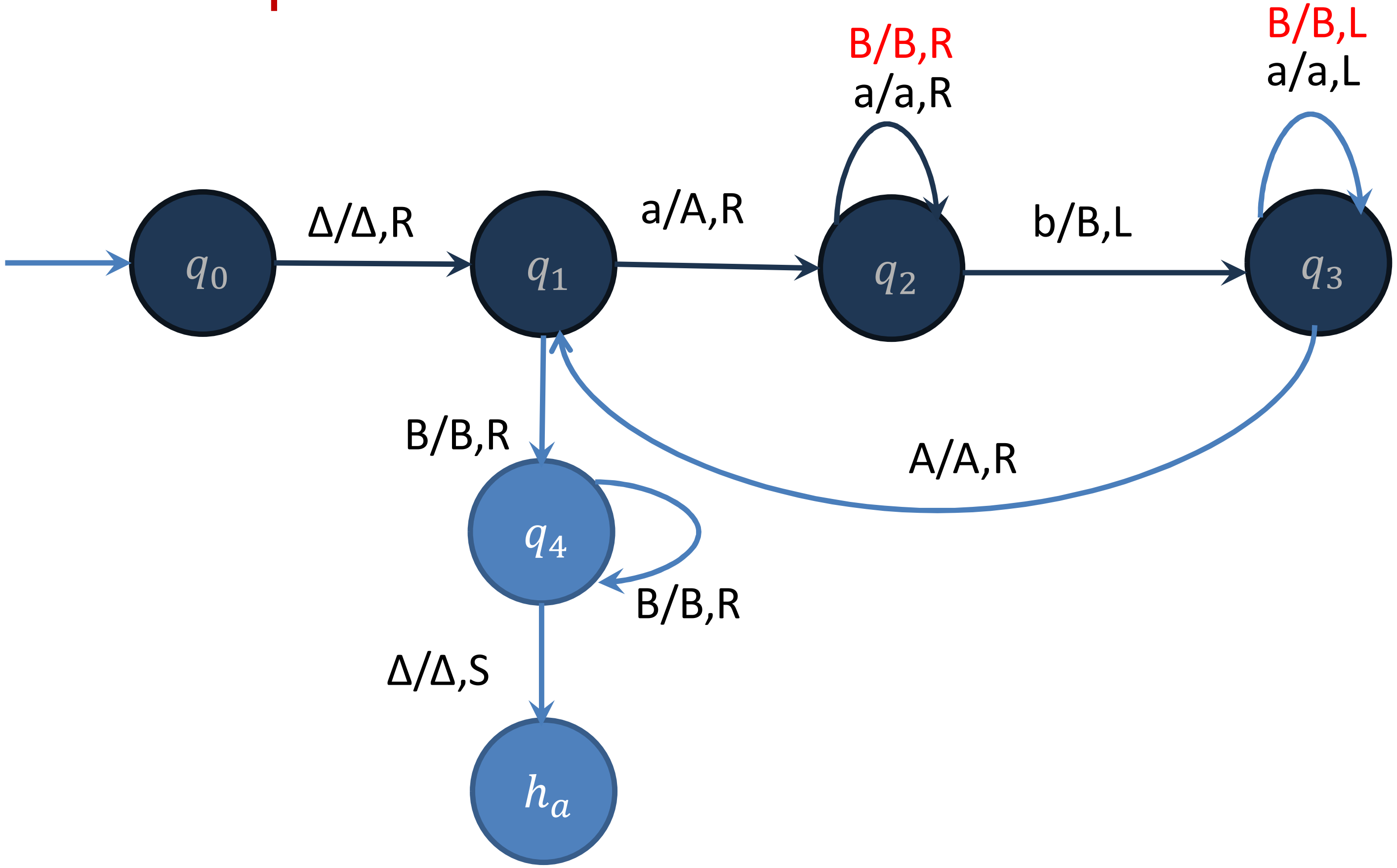
Δ	A	a	b	b	Δ



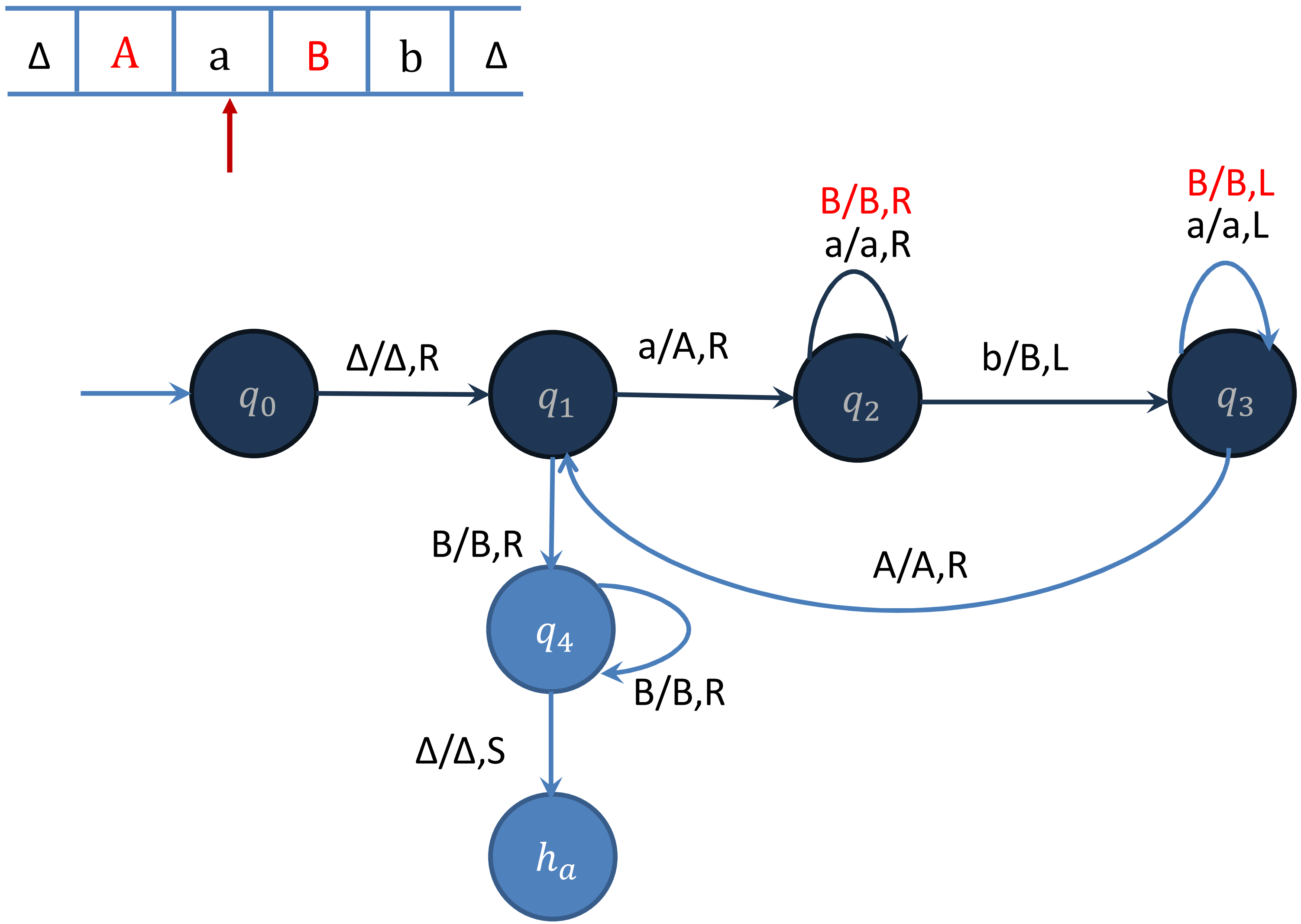


Design a Turing machine for accepting $\{a^n b^n | \; n \geq 1\}$

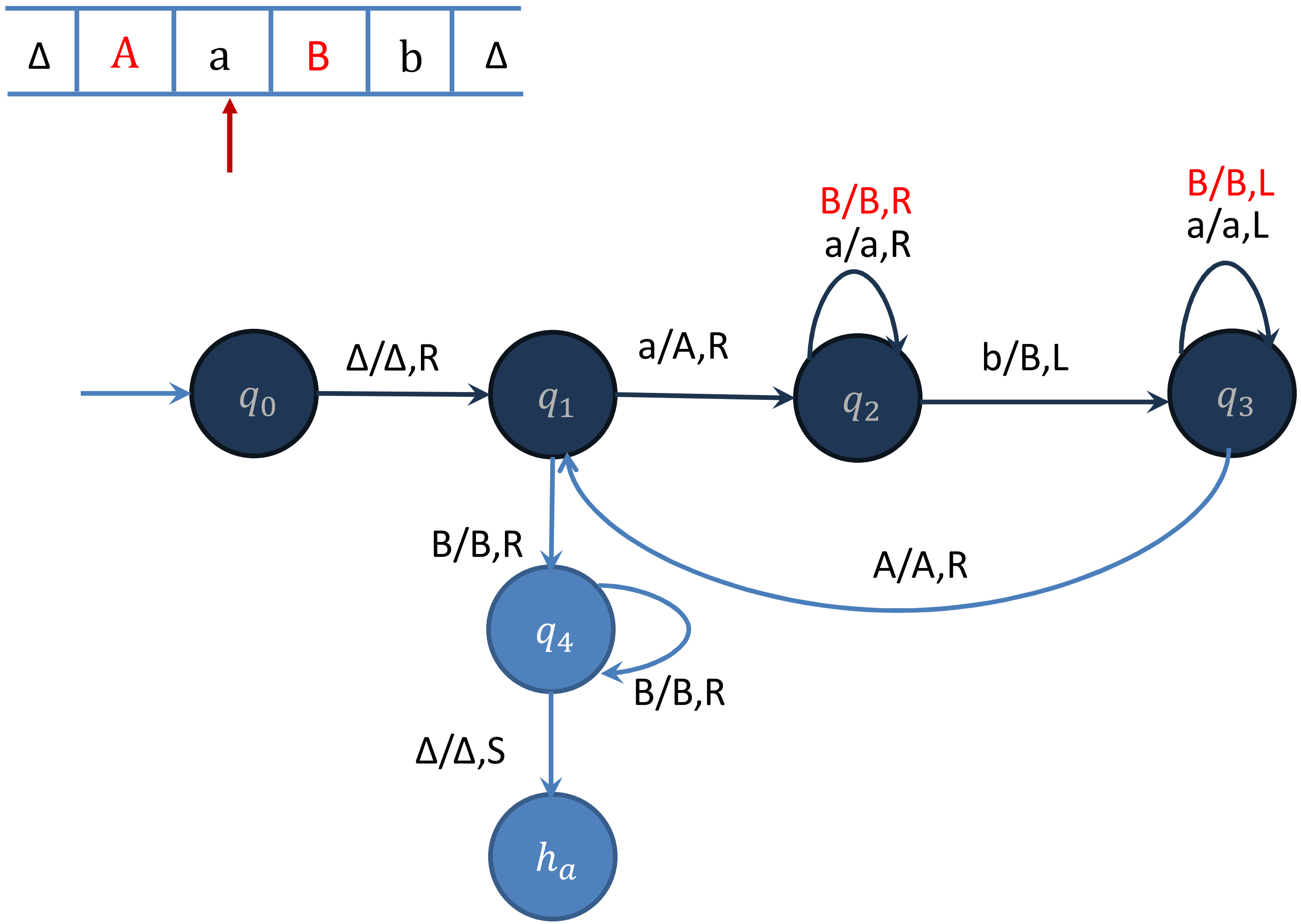




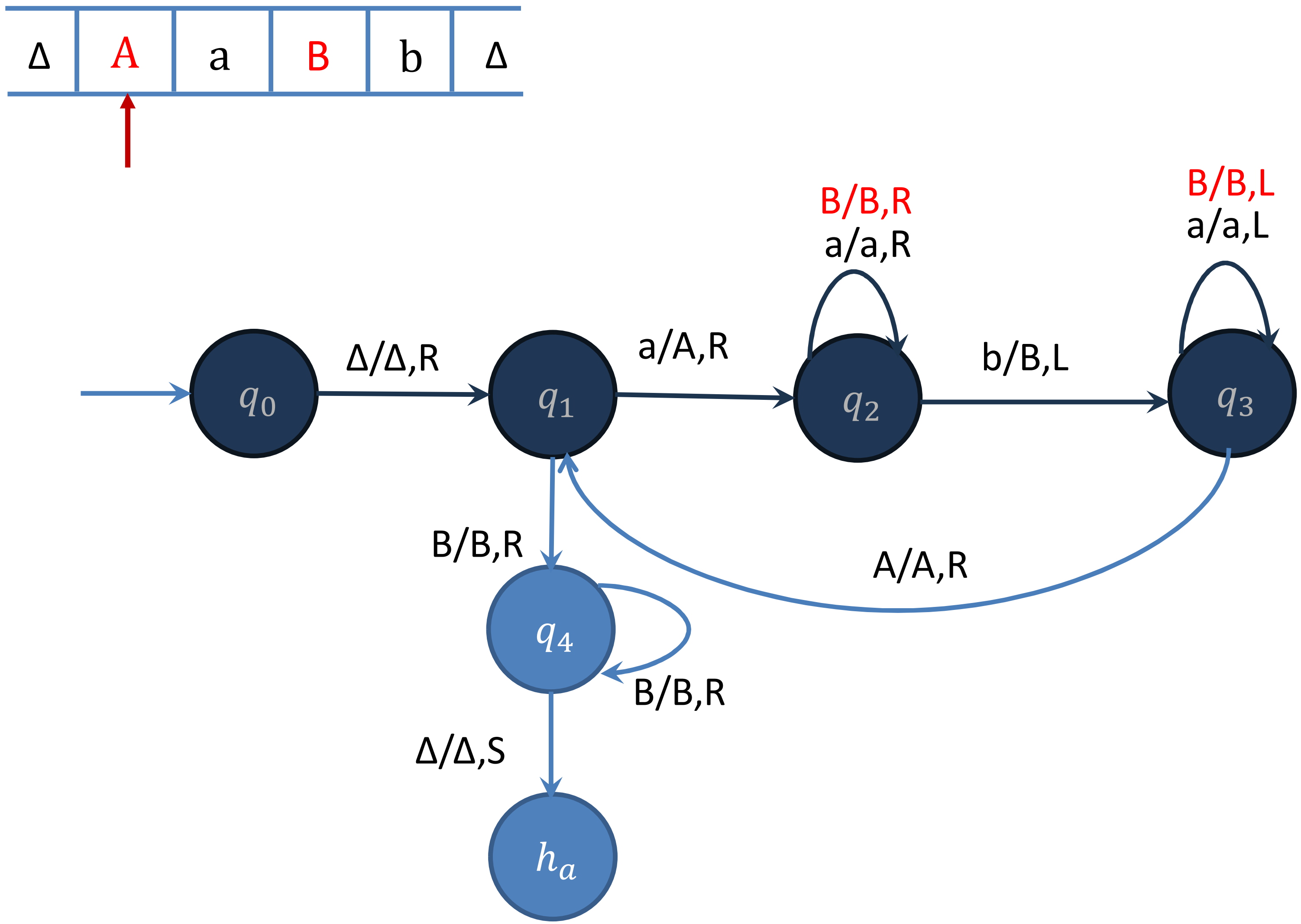
Design a Turing machine for accepting $\{a^n b^n | \, n \geq 1\}$



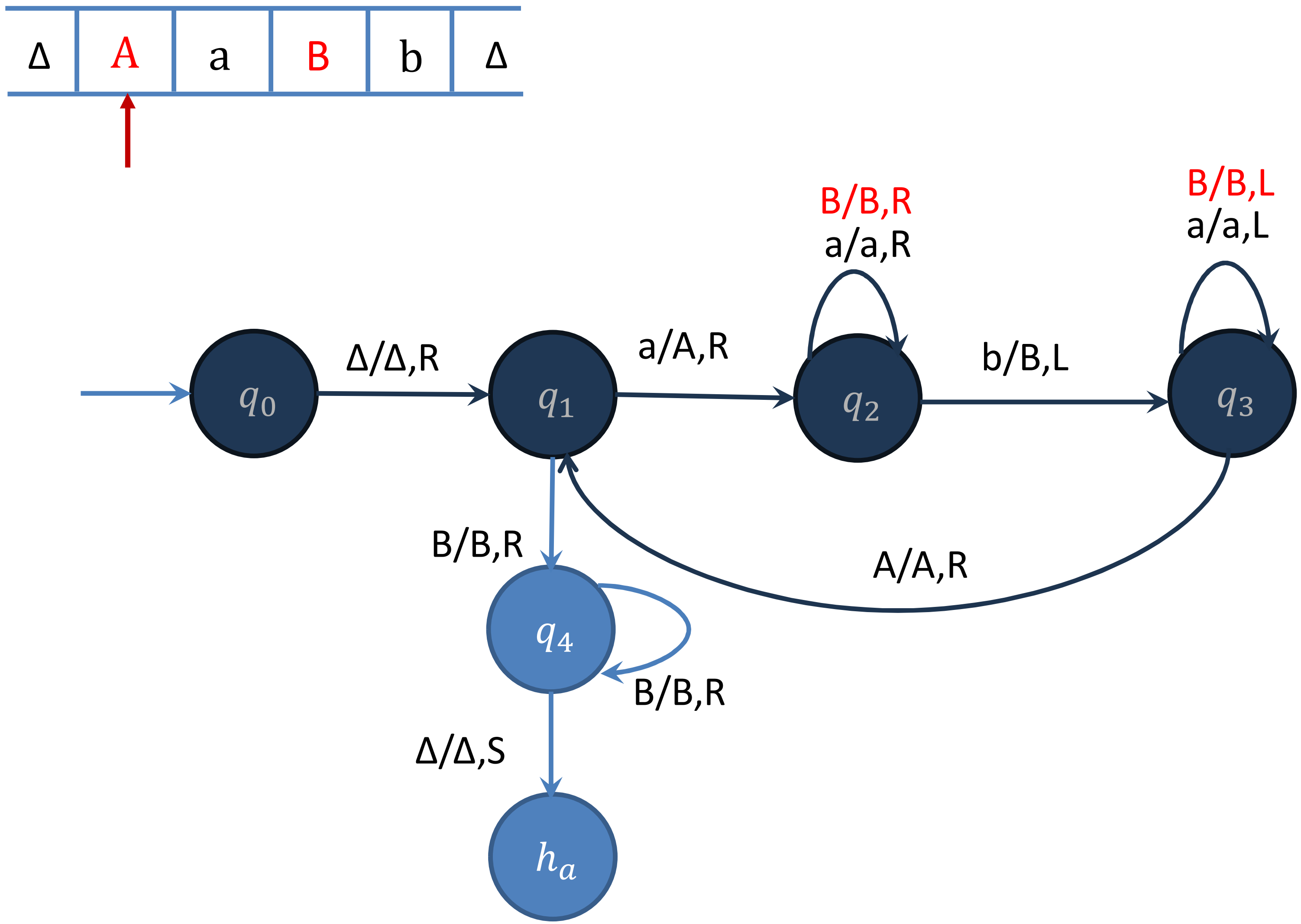
Design a Turing machine for accepting $\{a^n b^n \mid n \geq 1\}$



Design a Turing machine for accepting $\{a^n b^n | \; n \geq 1 \}$

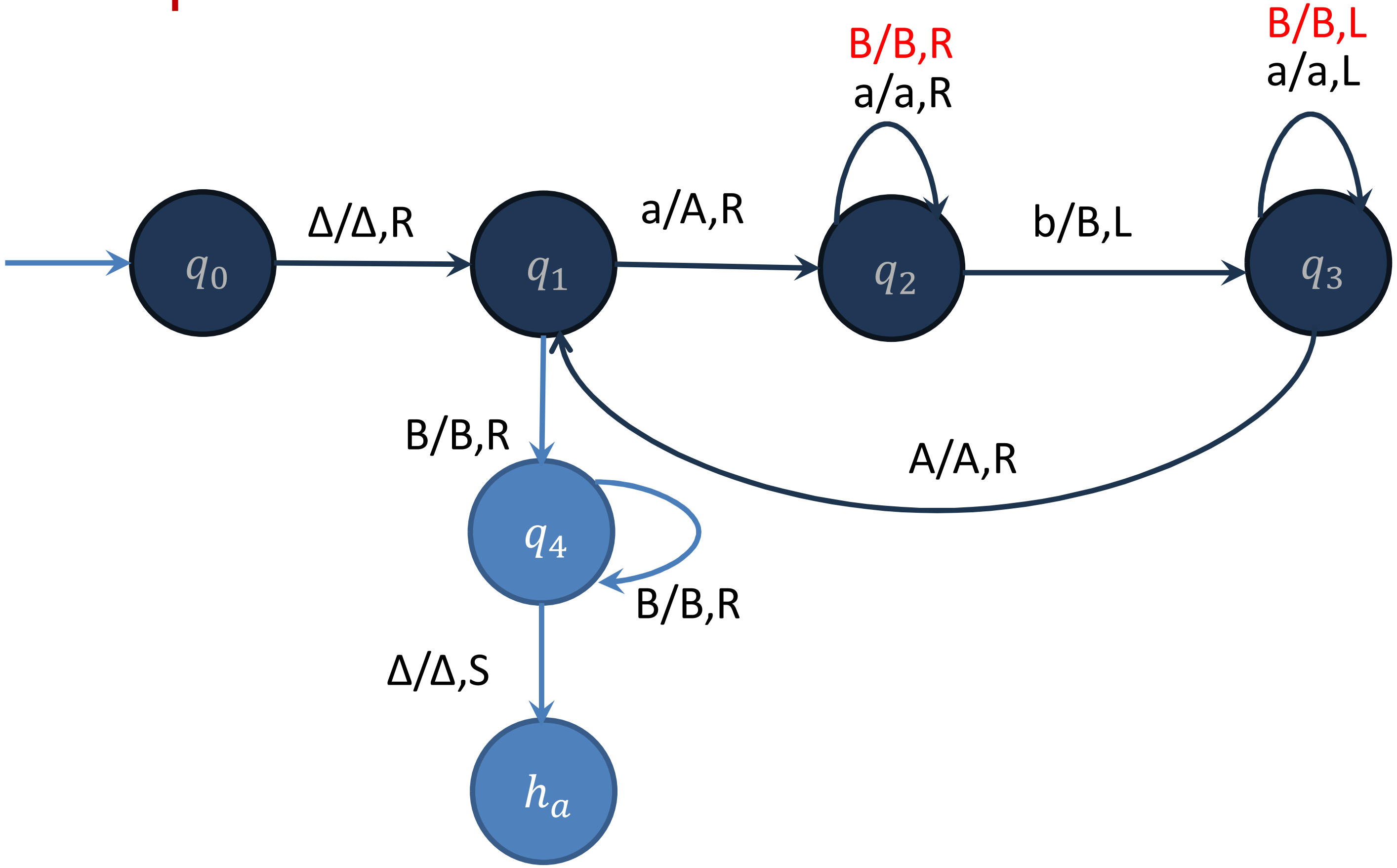


Design a Turing machine for accepting $\{a^n b^n |\, n \geq 1\}$

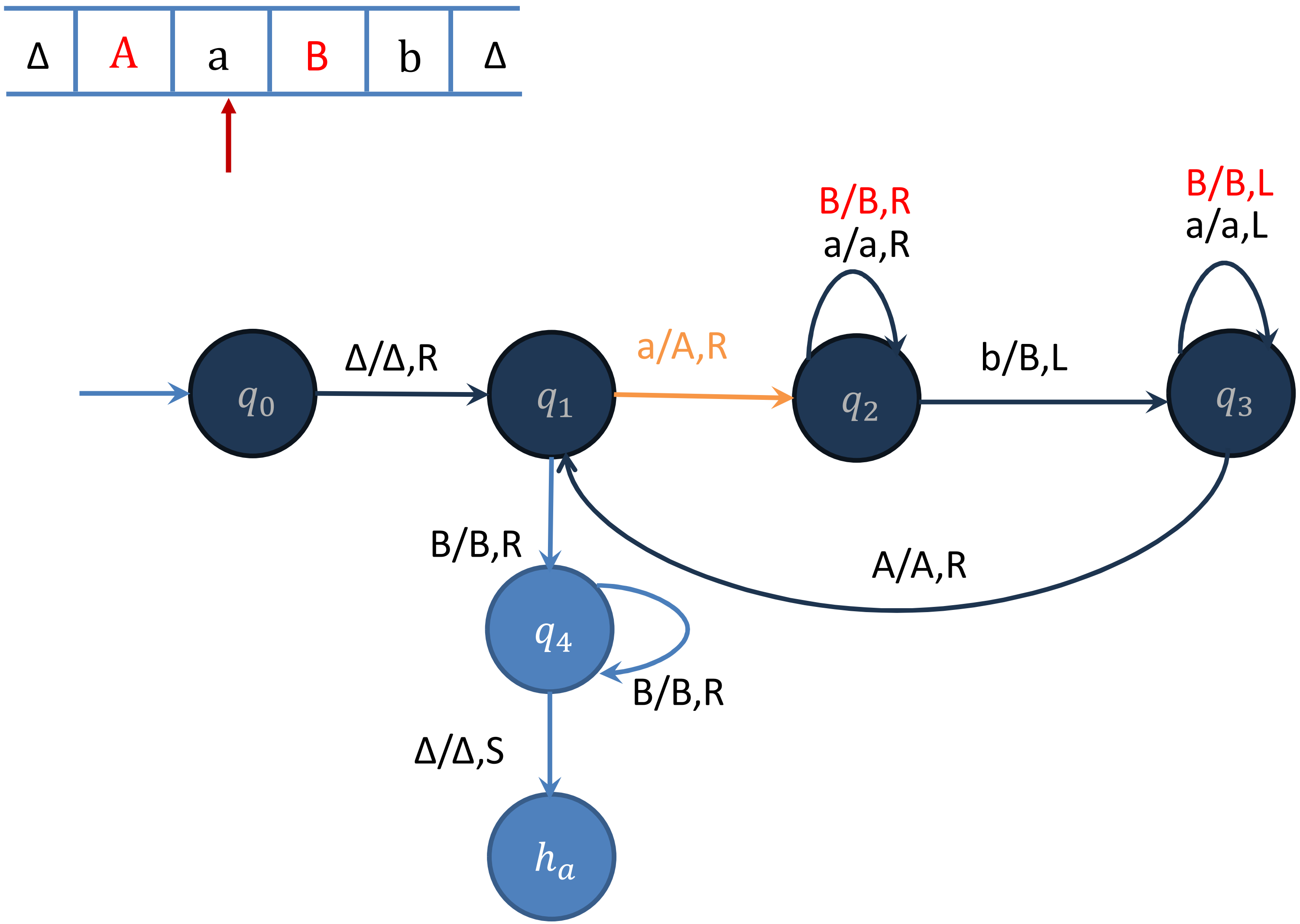


Design a Turing machine for accepting $\{a^n b^n \mid n \geq 1\}$

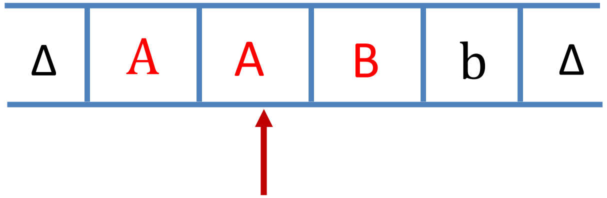
Δ	A	a	B	b	Δ



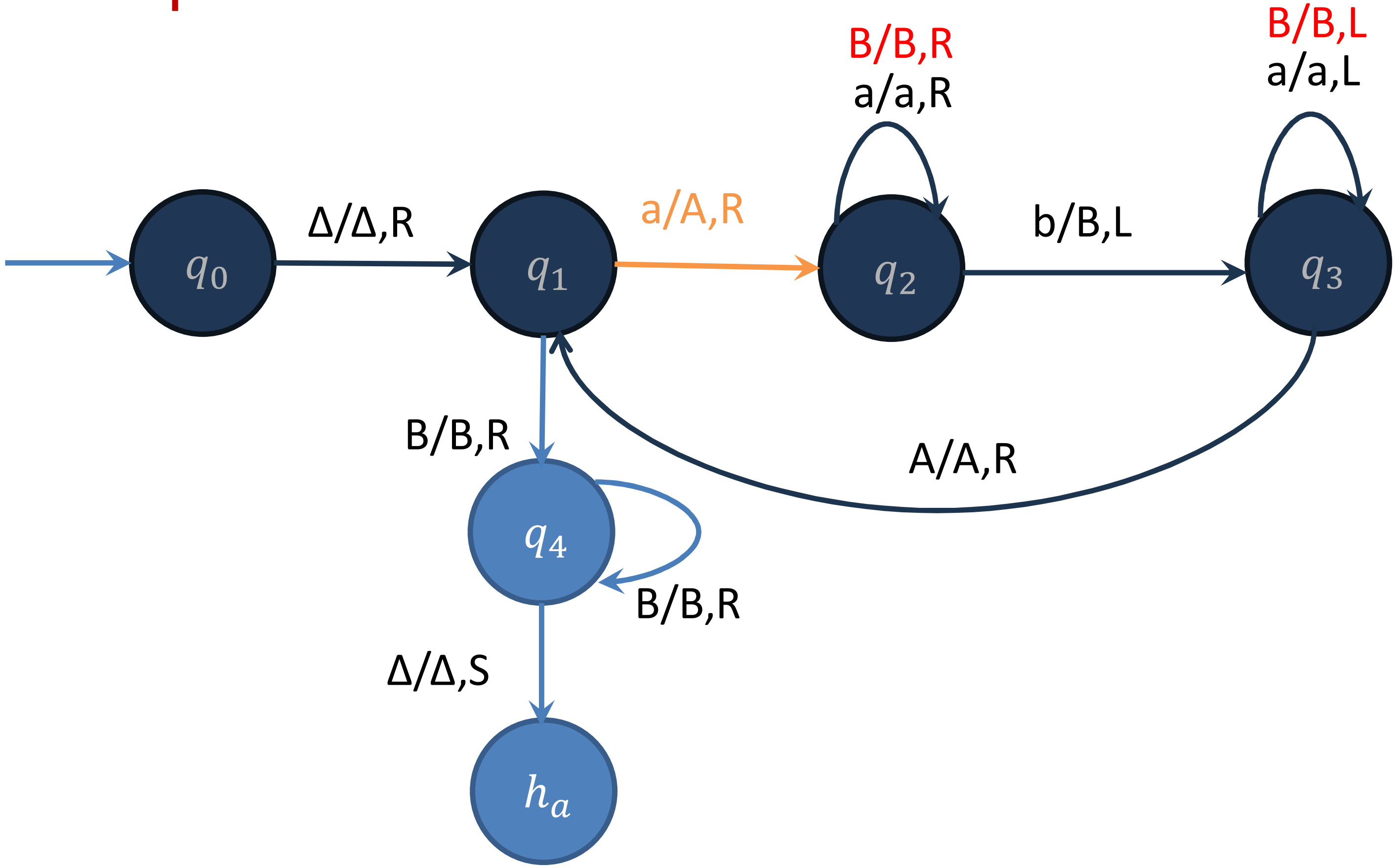
Design a Turing machine for accepting $\{a^n b^n | \, n \geq 1 \}$



Design a Turing machine for accepting $\{a^n b^n | \ n \geq 1 \}$

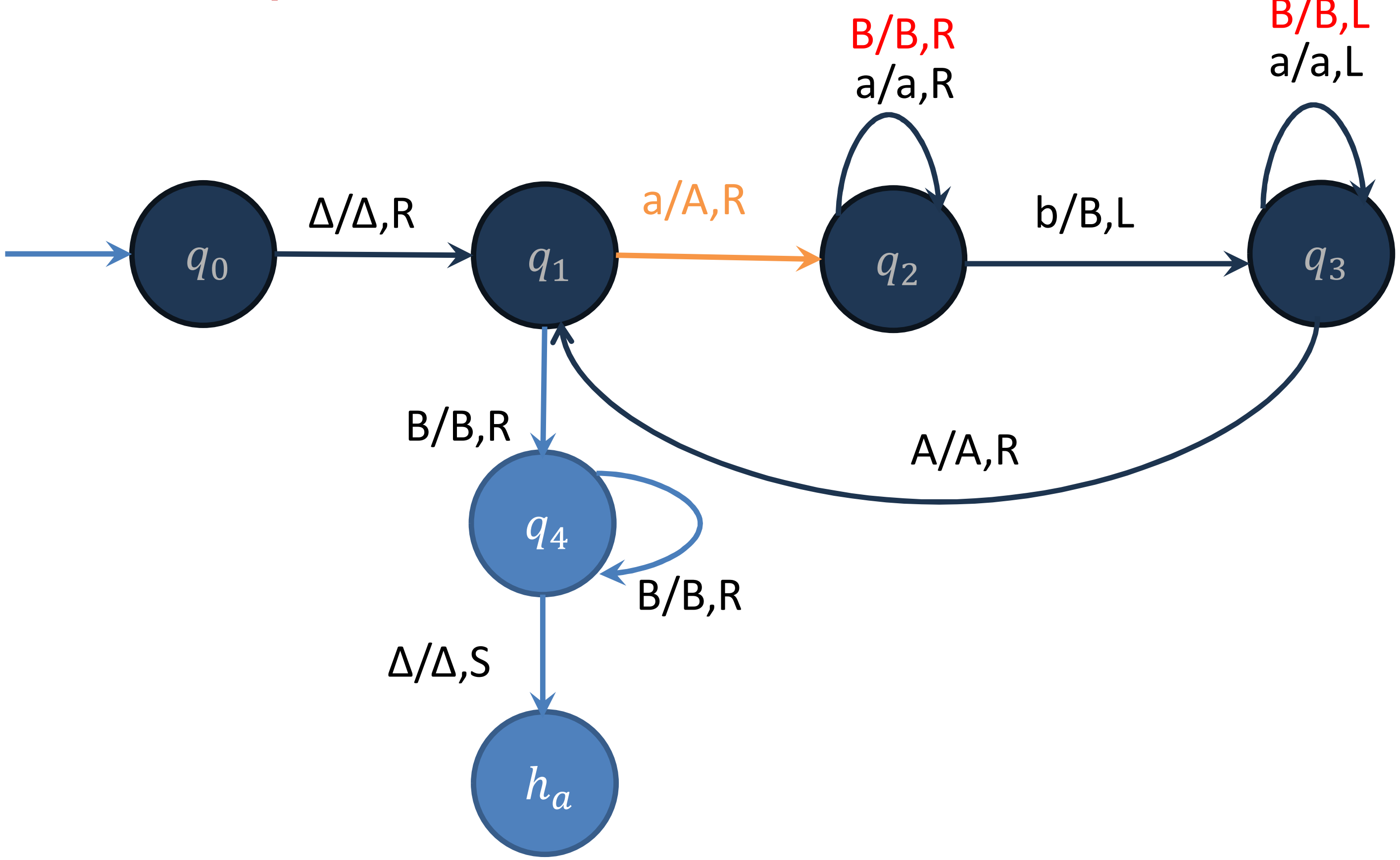


Δ	A	A	B	b	Δ

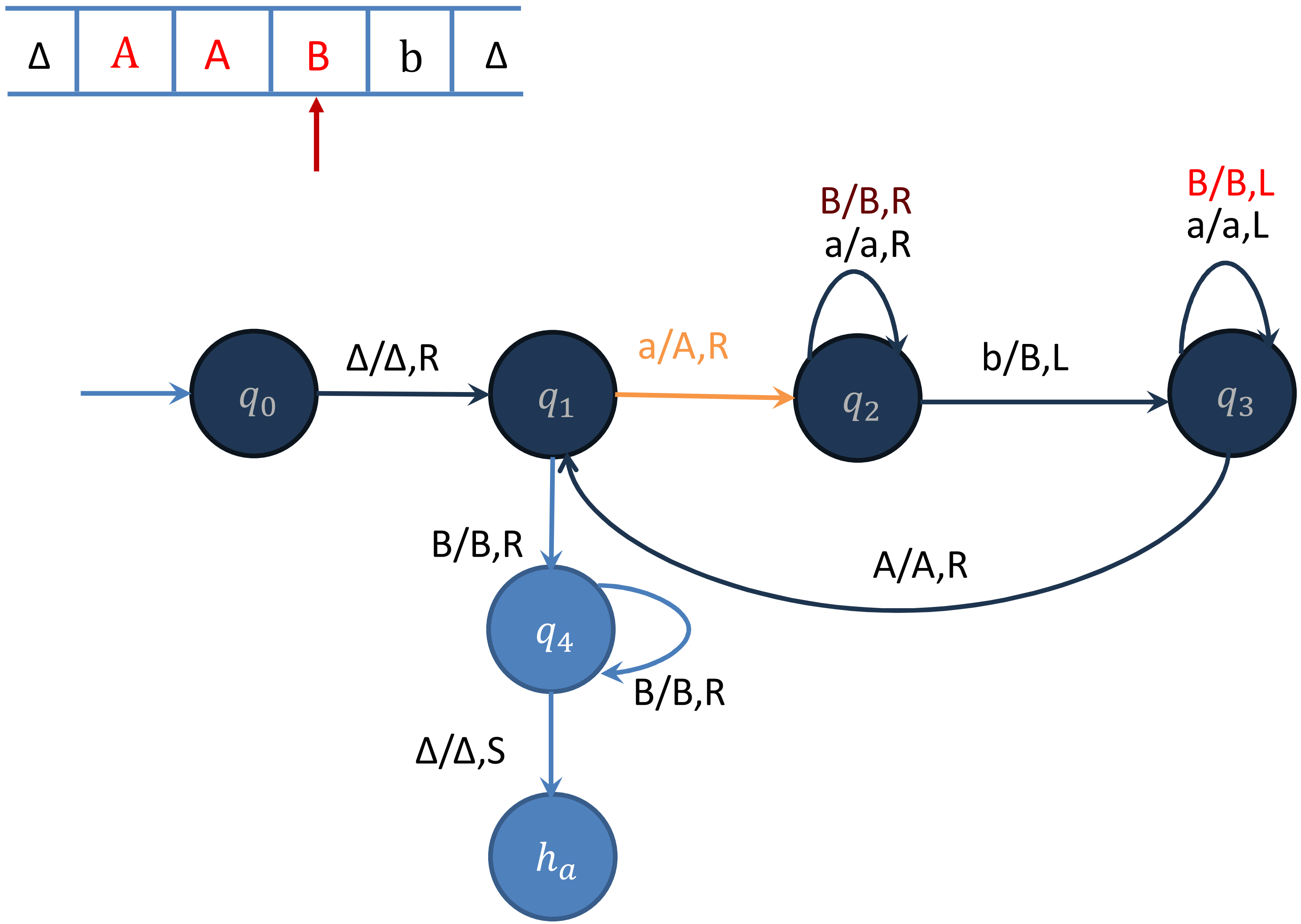


Design a Turing machine for accepting $\{a^n b^n \mid n \geq 1\}$

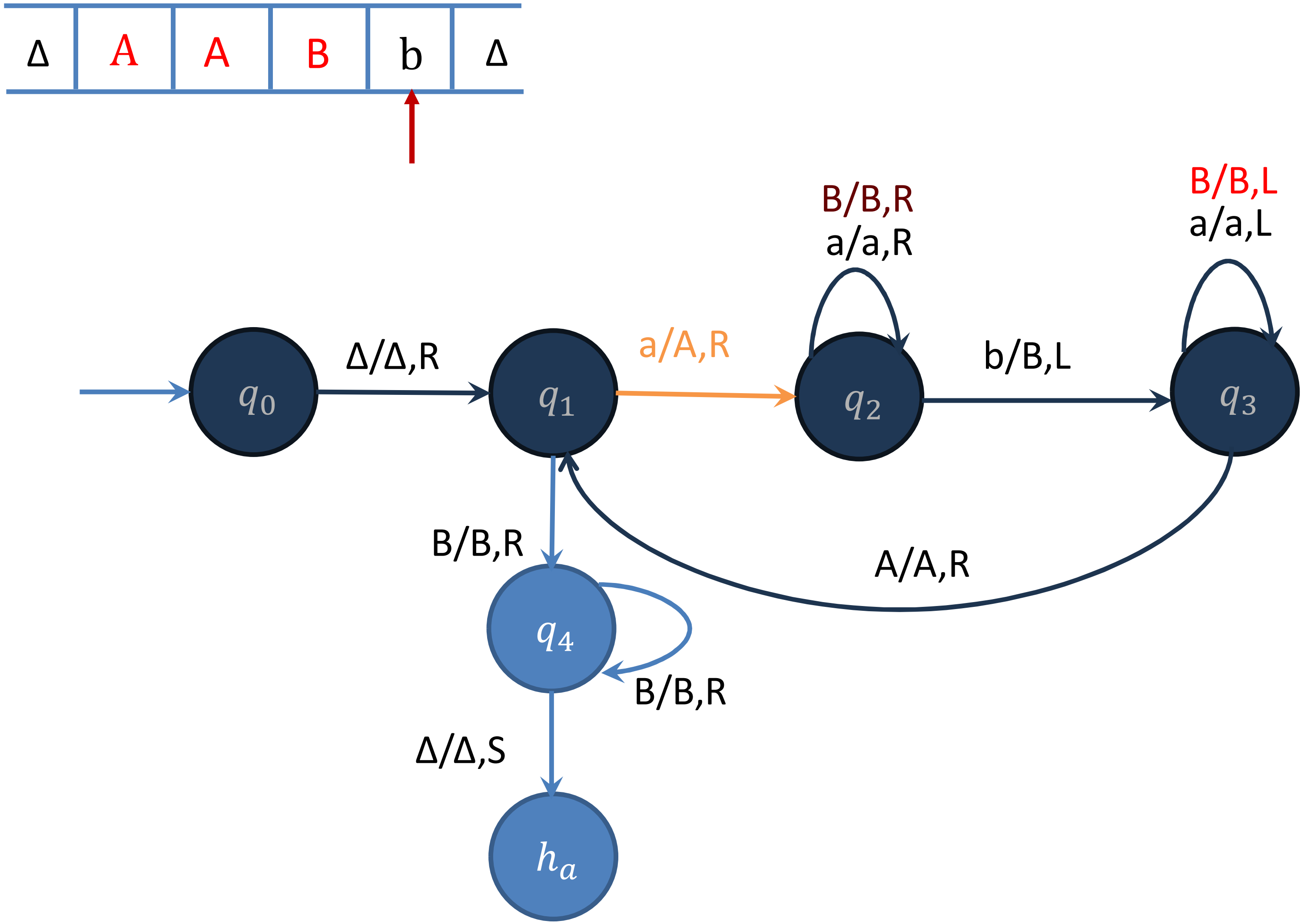
Δ	A	A	B	b	Δ



Design a Turing machine for accepting $\{a^n b^n | \ n \geq 1 \}$



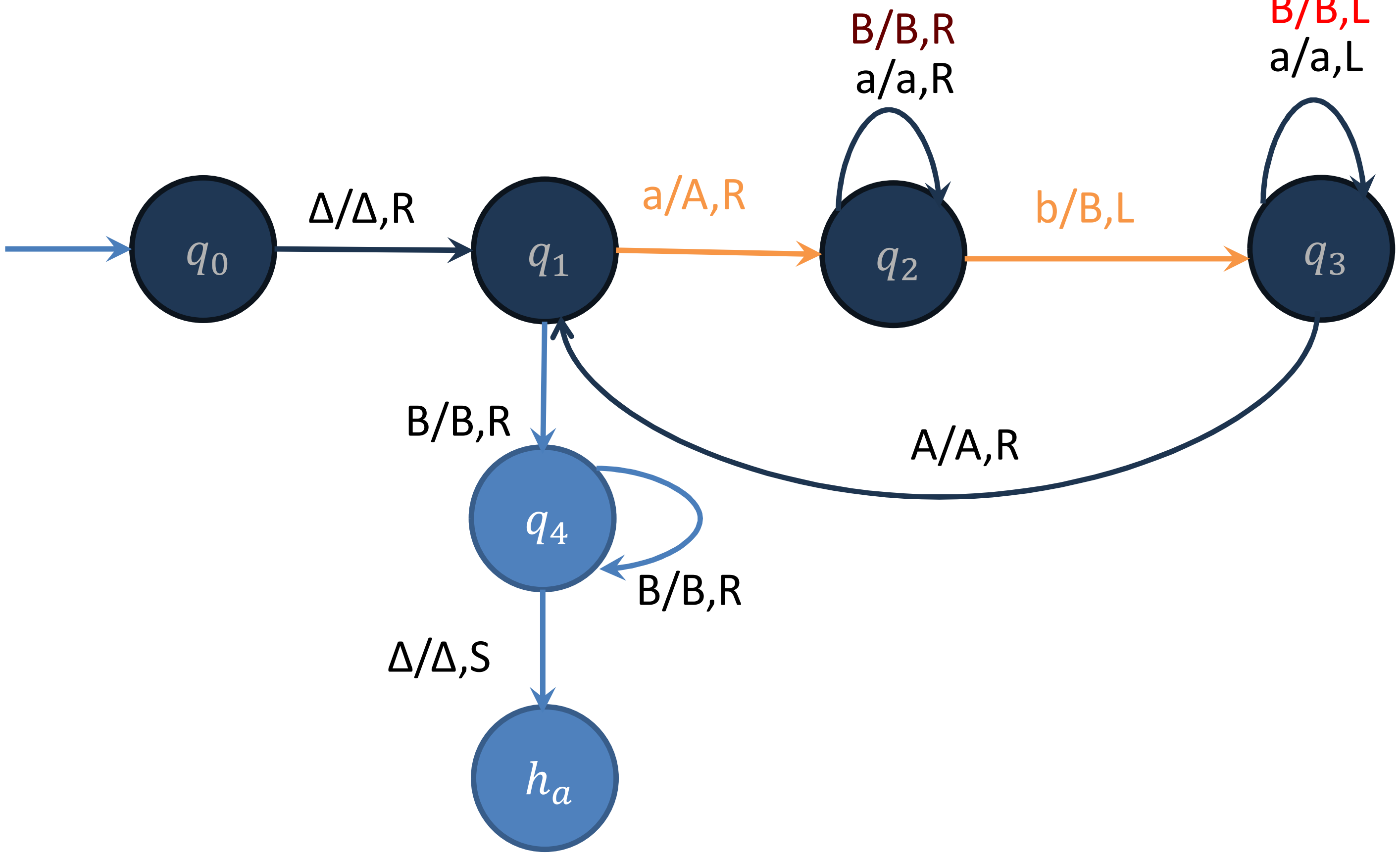
Design a Turing machine for accepting $\{a^n b^n | \; n \geq 1 \}$



Δ	A	A	B	b	Δ

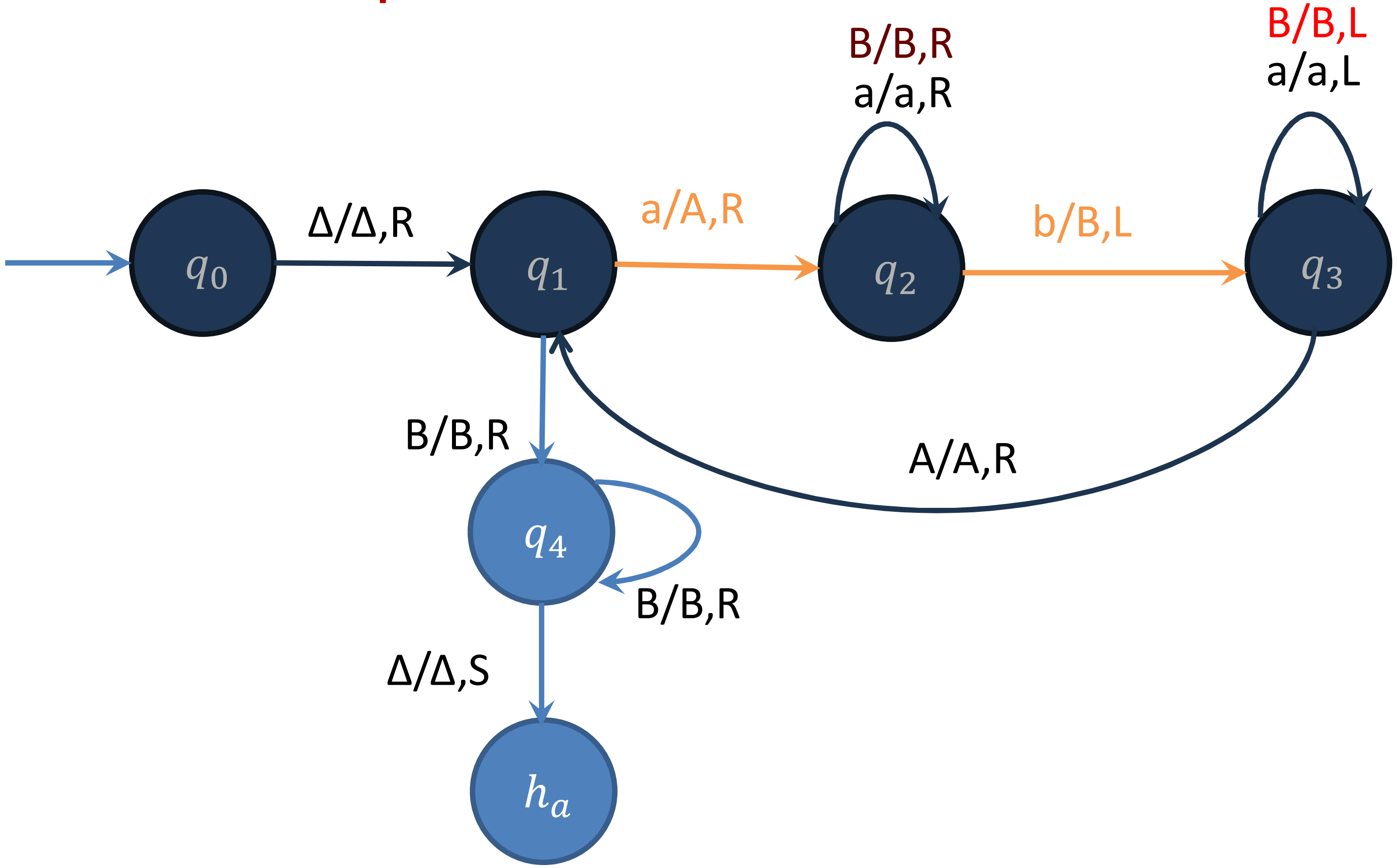
Design a Turing machine for accepting $\{a^n b^n \mid n \geq 1\}$

Δ	A	A	B	b	Δ

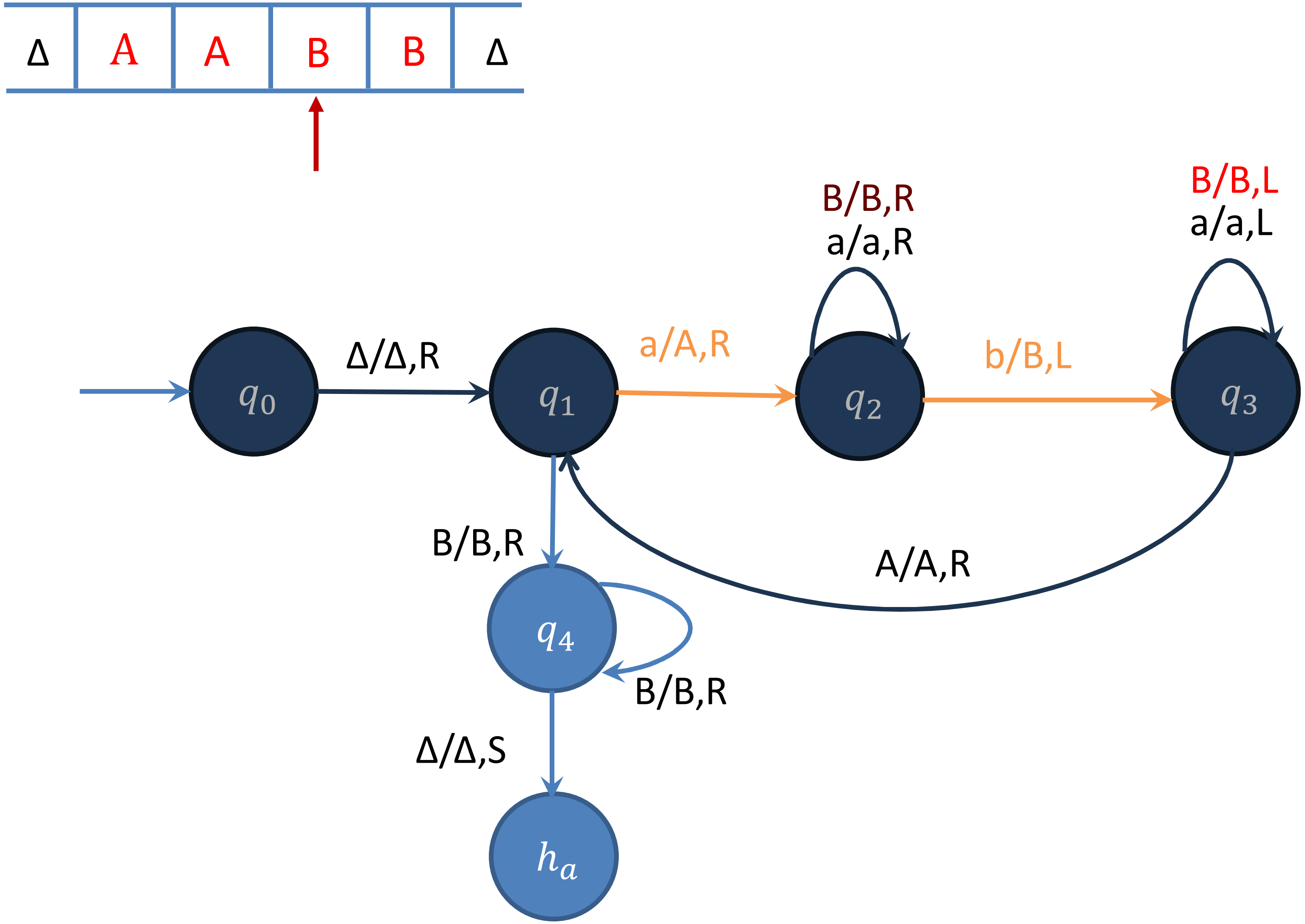


Design a Turing machine for accepting $\{a^n b^n | \, n \geq 1 \}$

Δ	A	A	B	B	Δ

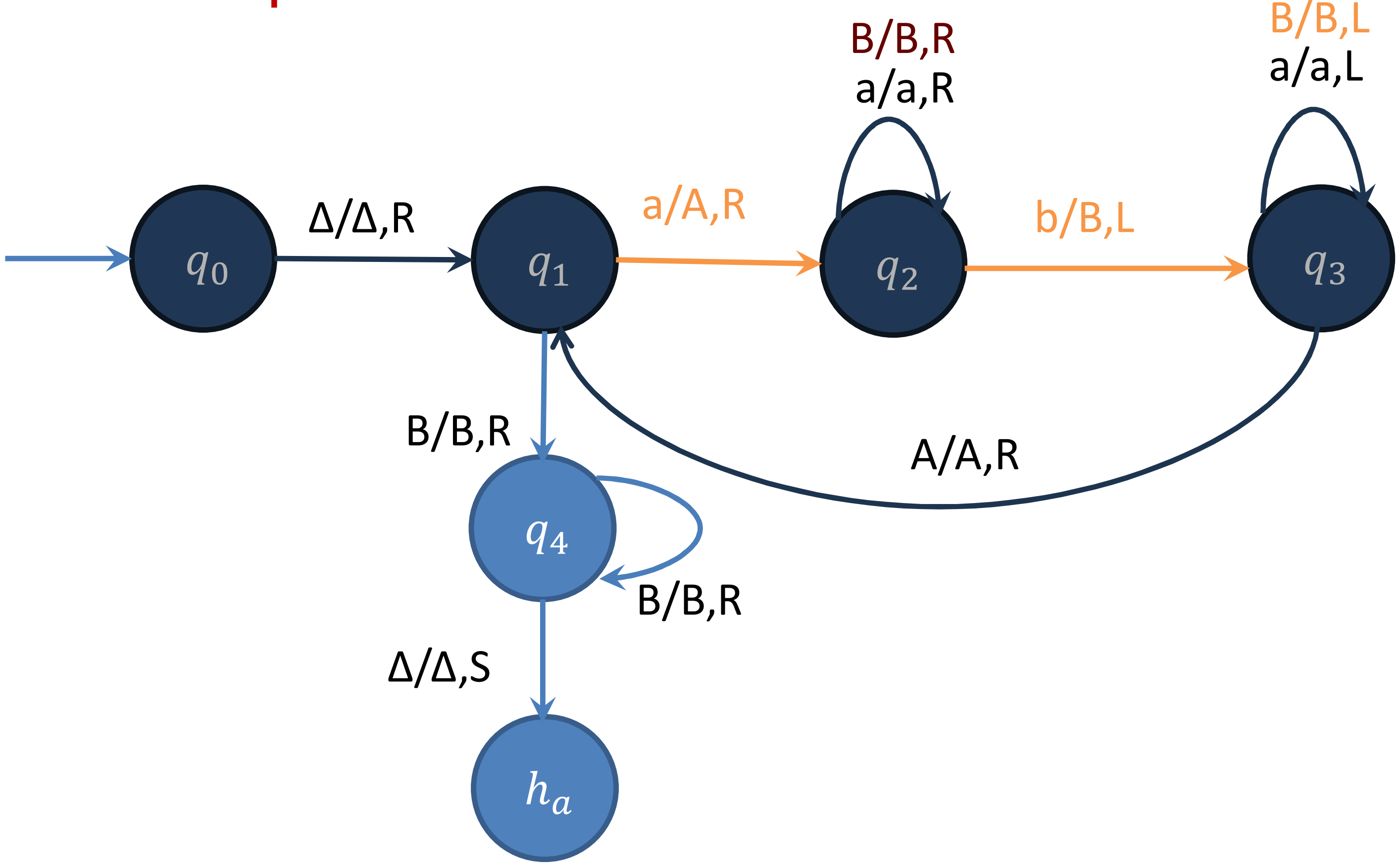


Design a Turing machine for accepting $\{a^n b^n | \; n \geq 1 \}$

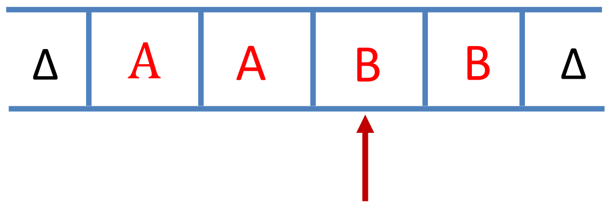


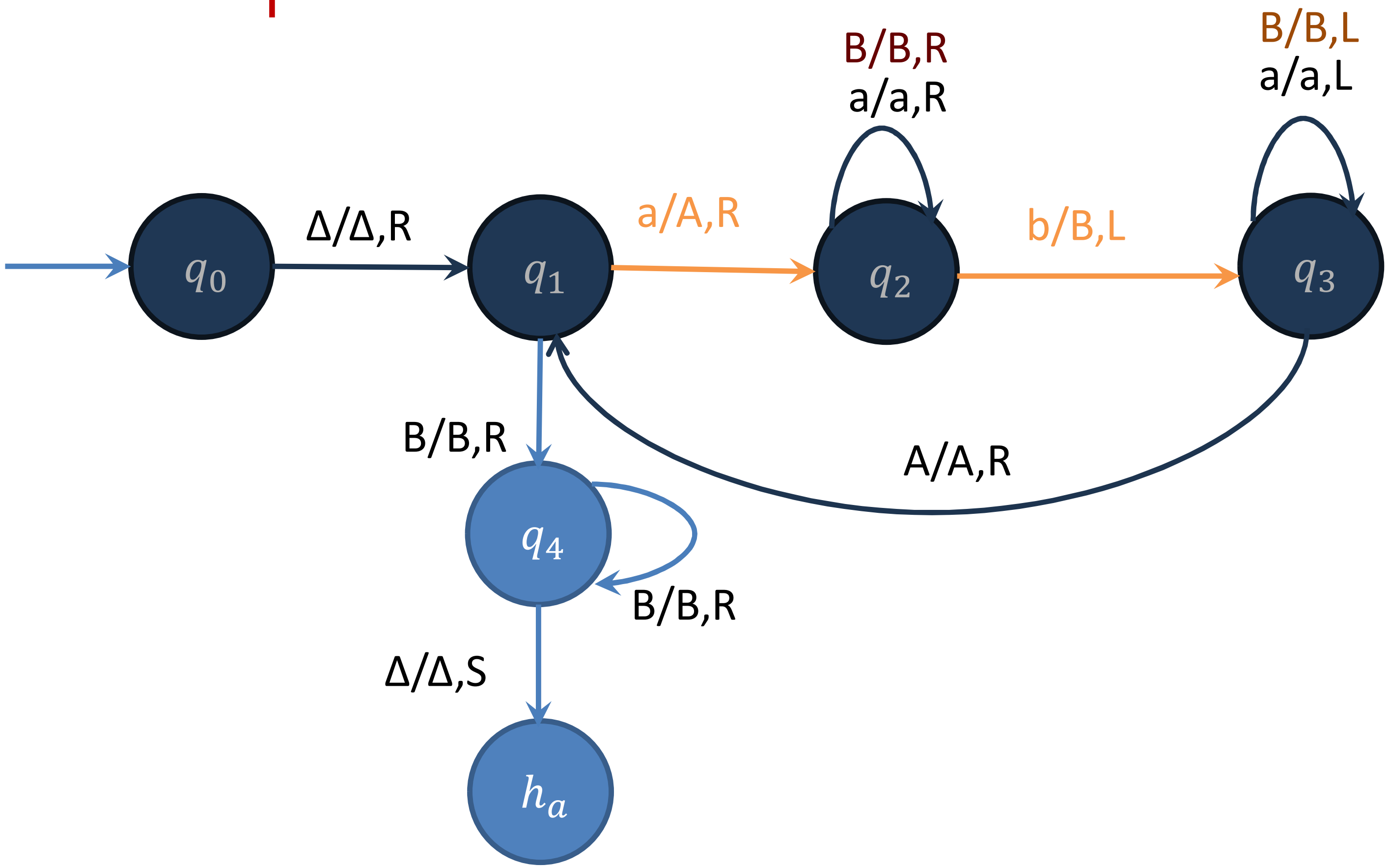
Design a Turing machine for accepting $\{a^n b^n | \, n \geq 1 \}$

Δ	A	A	B	B	Δ

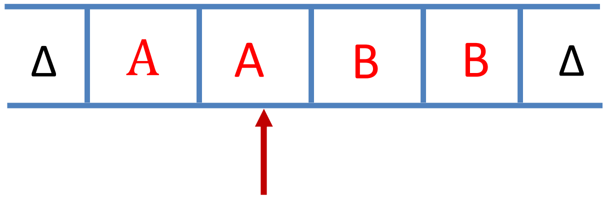


Design a Turing machine for accepting $\{a^n b^n | \, n \geq 1\}$

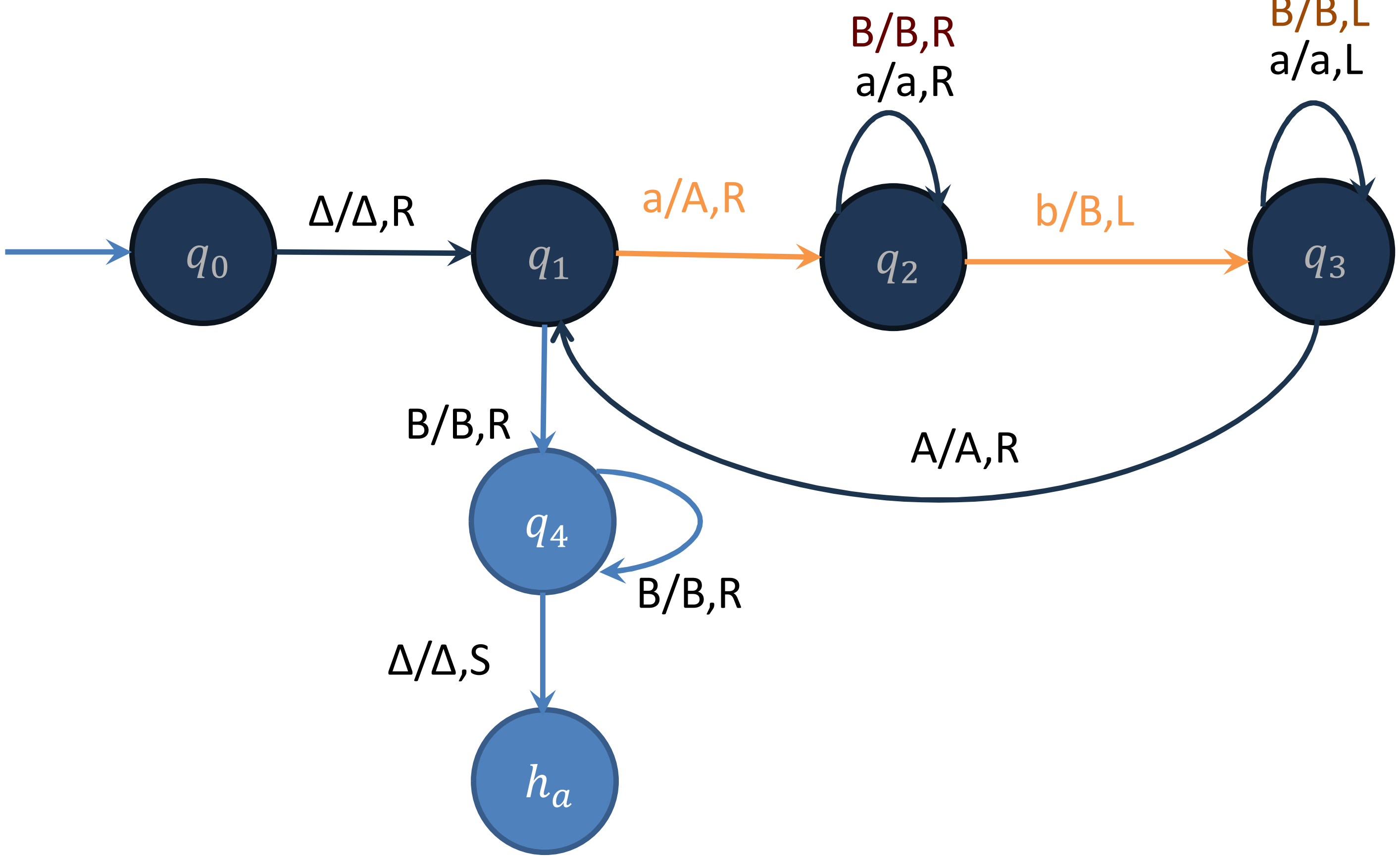




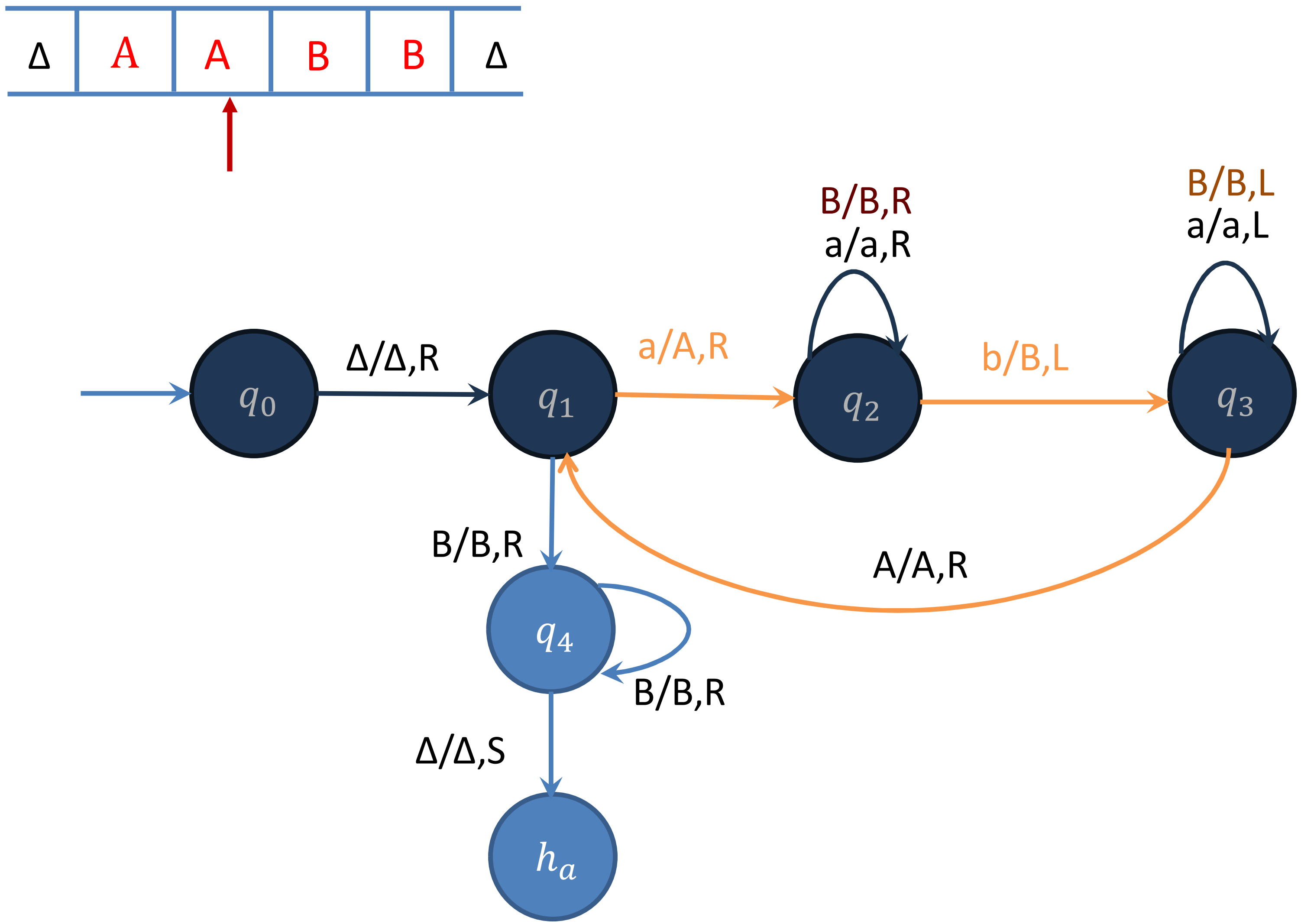
Design a Turing machine for accepting $\{a^n b^n \mid n \geq 1\}$



Δ	A	A	B	B	Δ

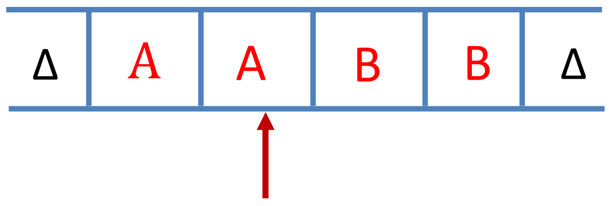


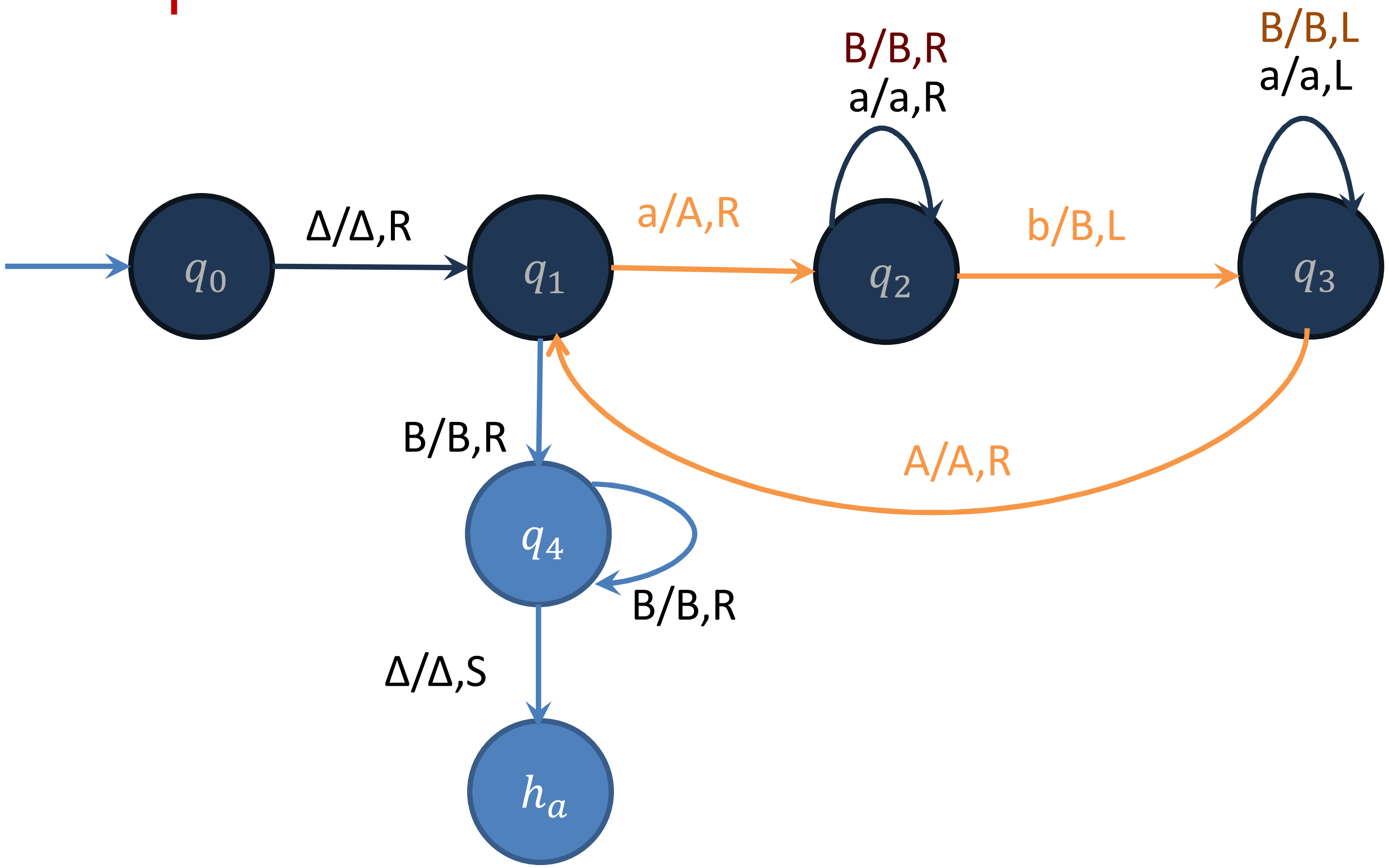
Design a Turing machine for accepting $\{a^n b^n | \ n \geq 1 \}$



Design a Turing machine for accepting $\{a^n b^n \mid n \geq 1\}$

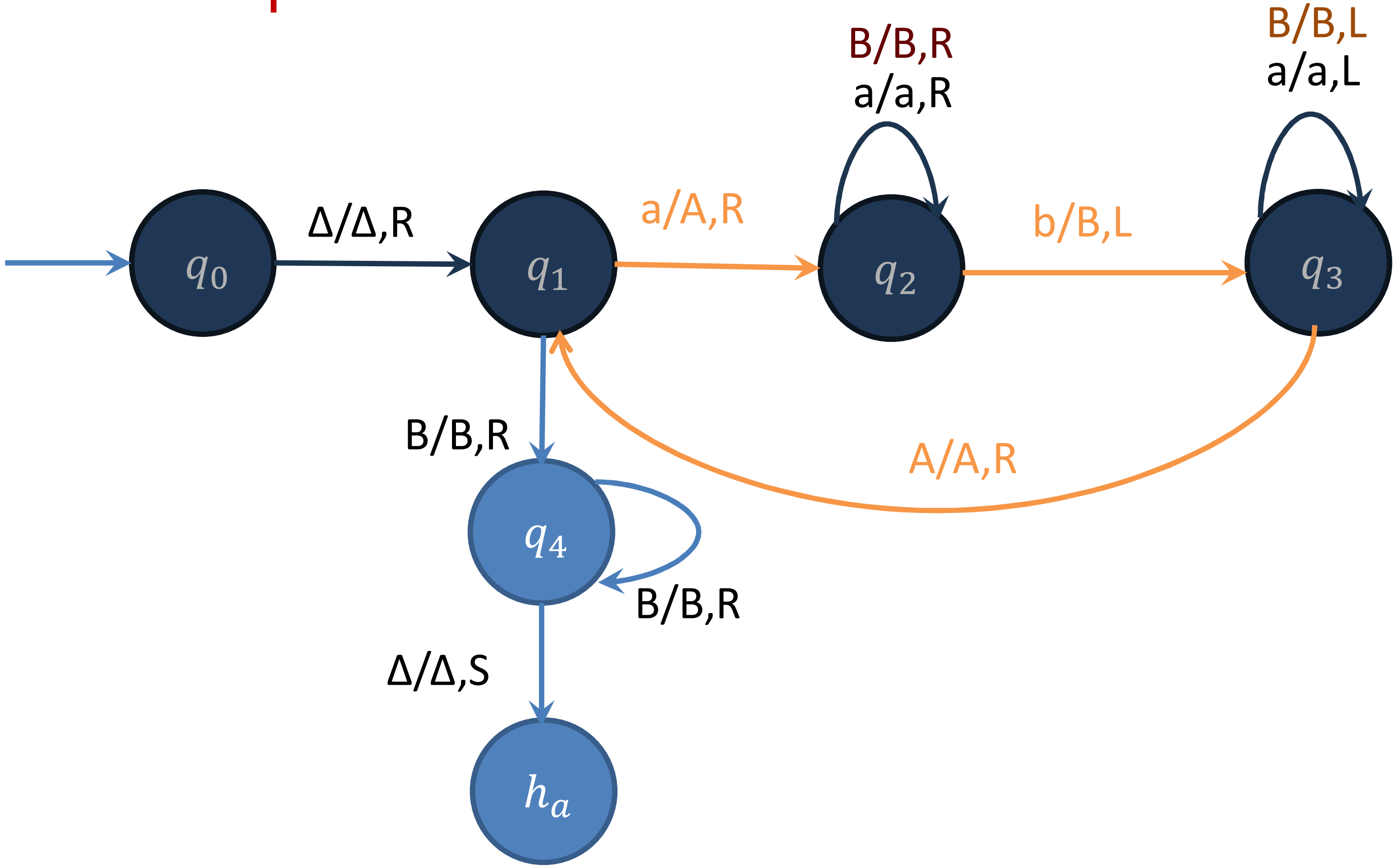
Δ	A	A	B	B	Δ



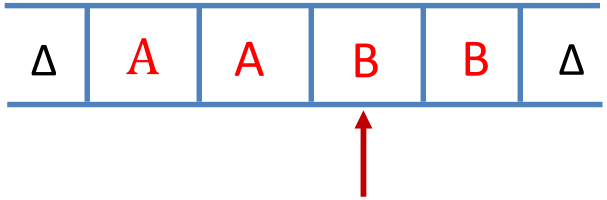


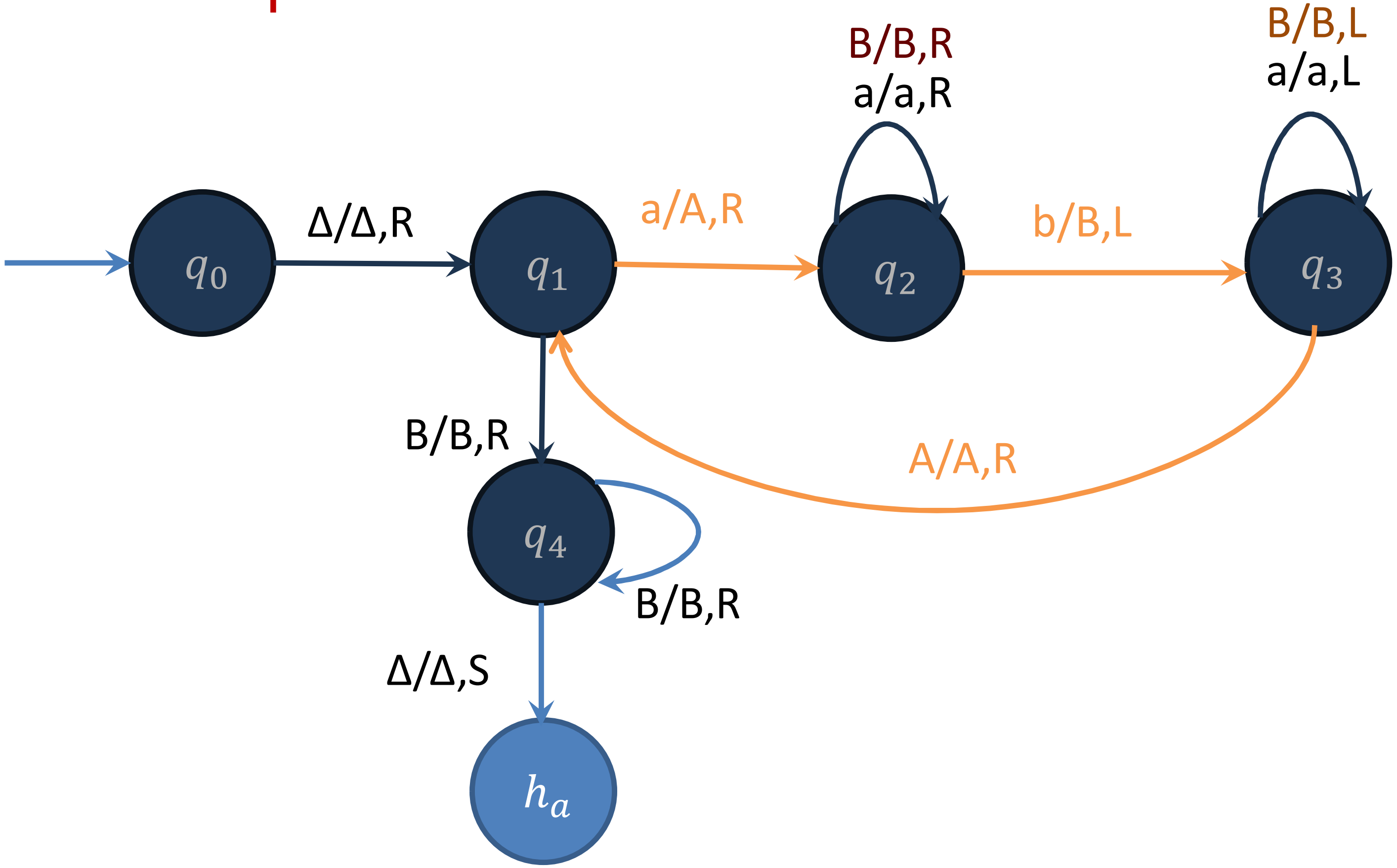
Design a Turing machine for accepting $\{a^n b^n \mid n \geq 1\}$

Δ	A	A	B	B	Δ

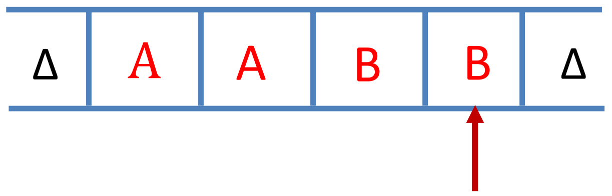


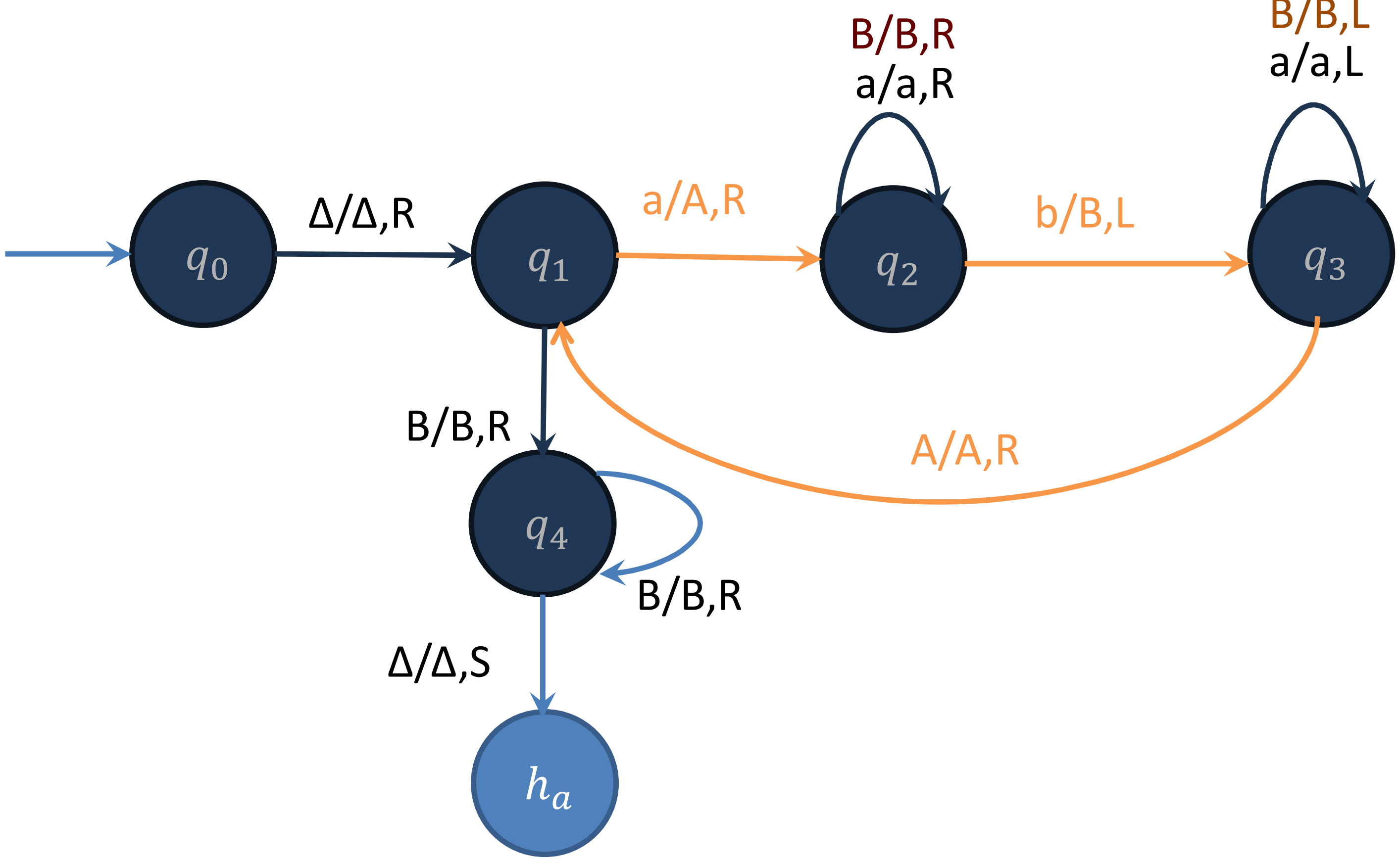
Design a Turing machine for accepting $\{a^n b^n \mid n \geq 1\}$



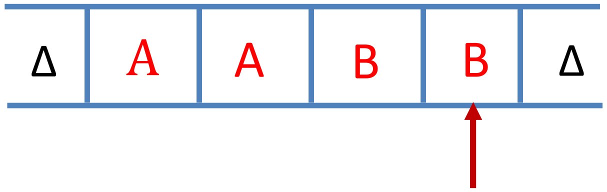


Design a Turing machine for accepting $\{a^n b^n | \ n \geq 1 \ \}$

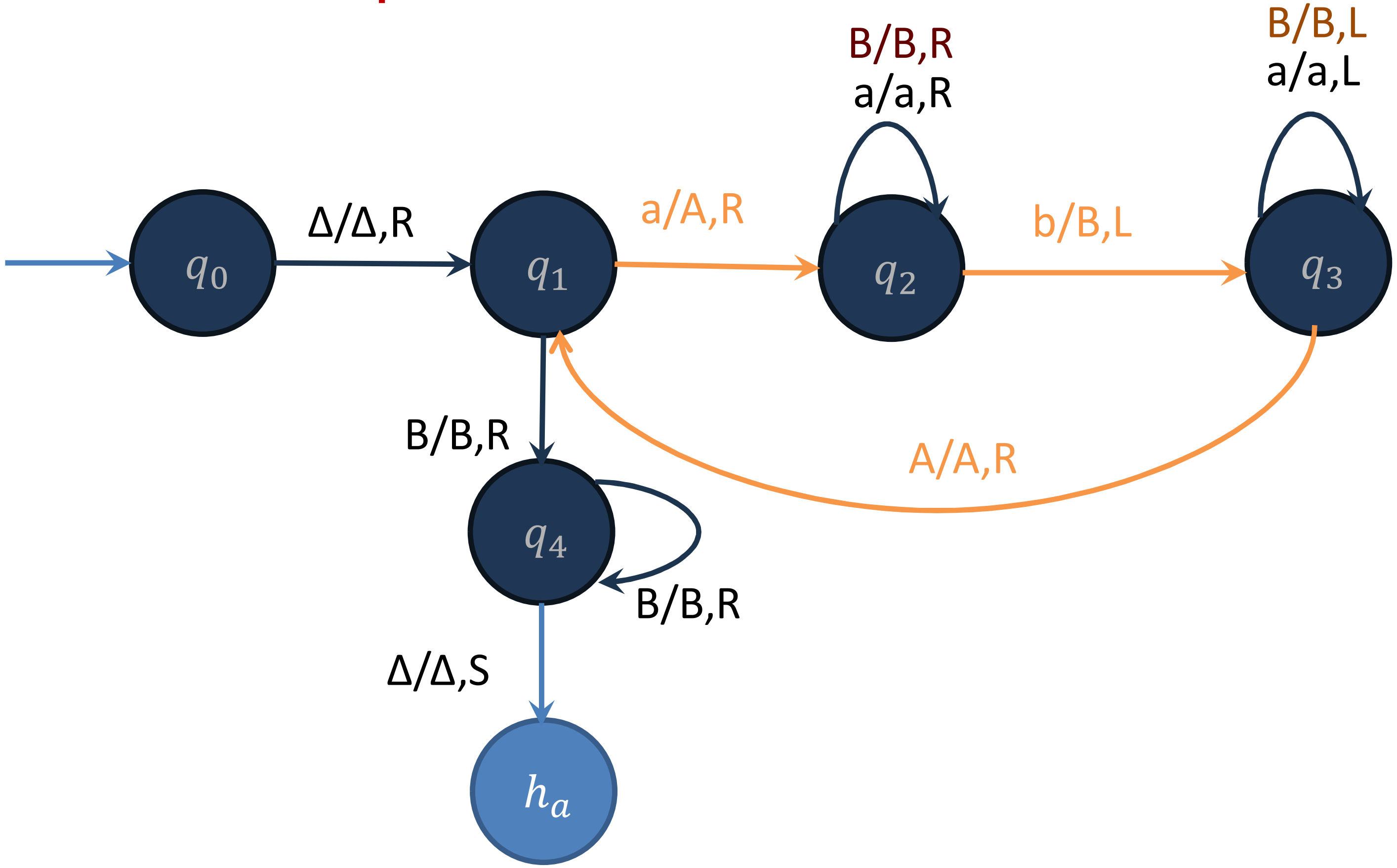




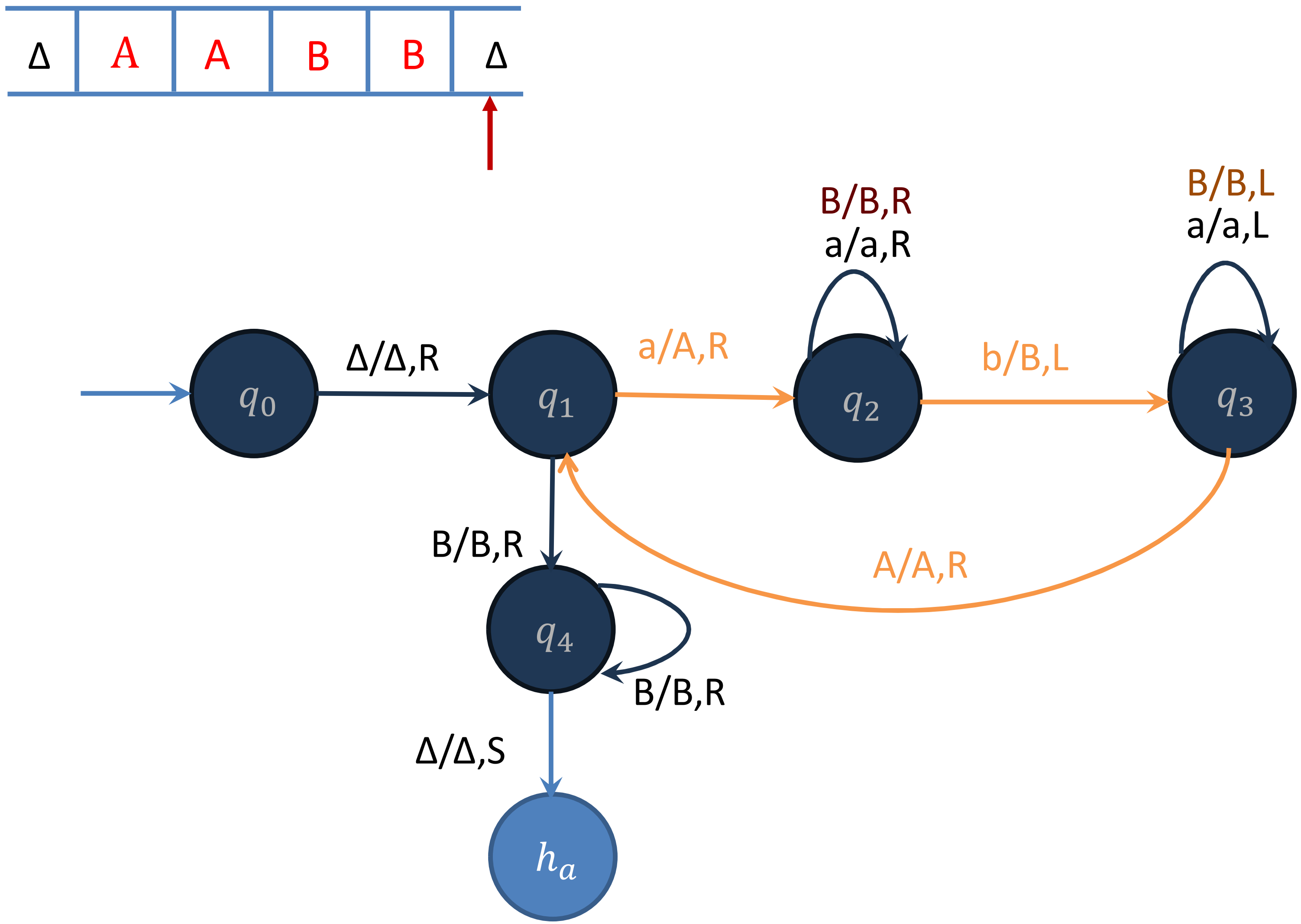
Design a Turing machine for accepting $\{a^n b^n |\ n \geq 1\}$



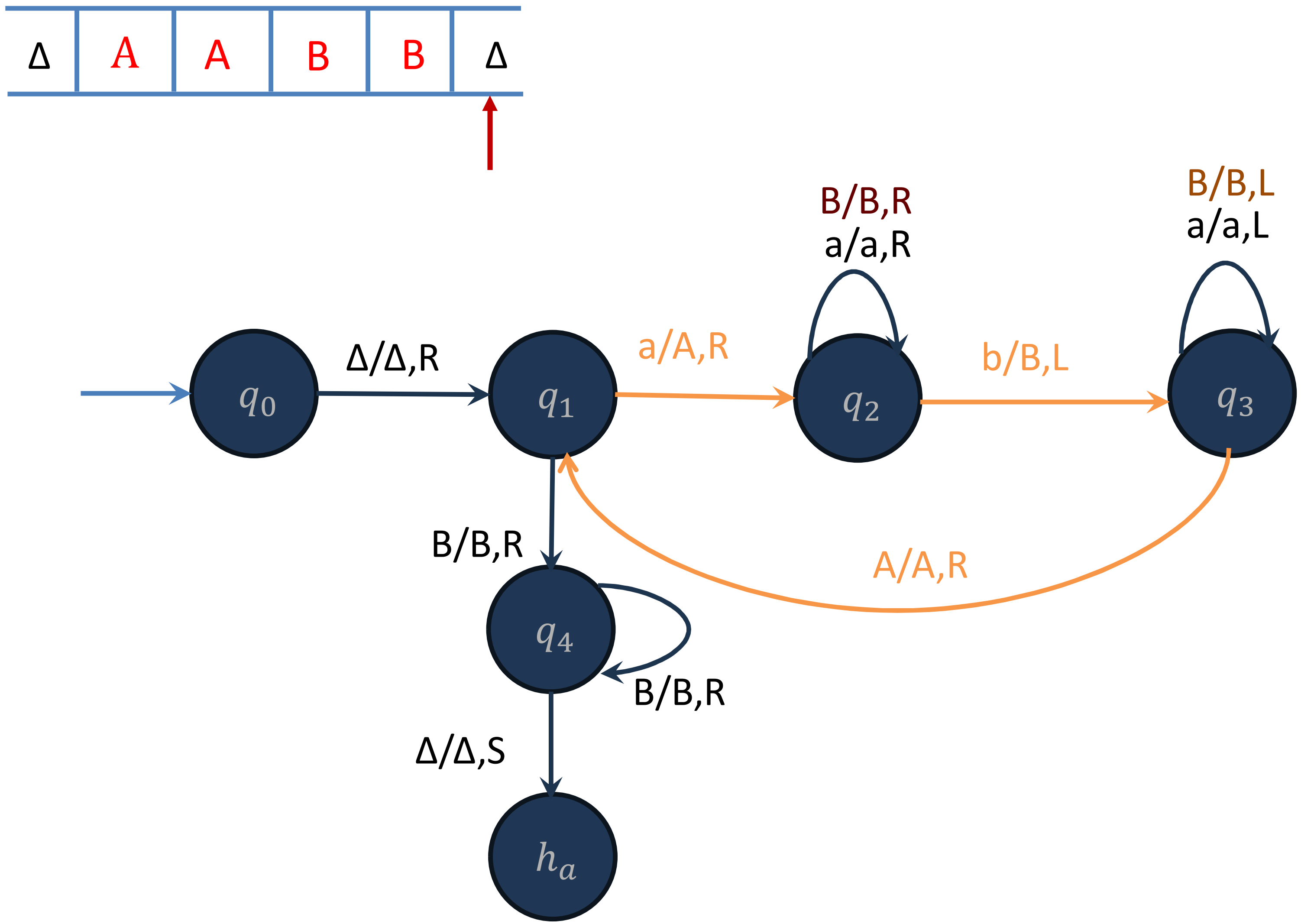
Δ	A	A	B	B	Δ



Design a Turing machine for accepting $\{a^n b^n | \, n \geq 1 \}$



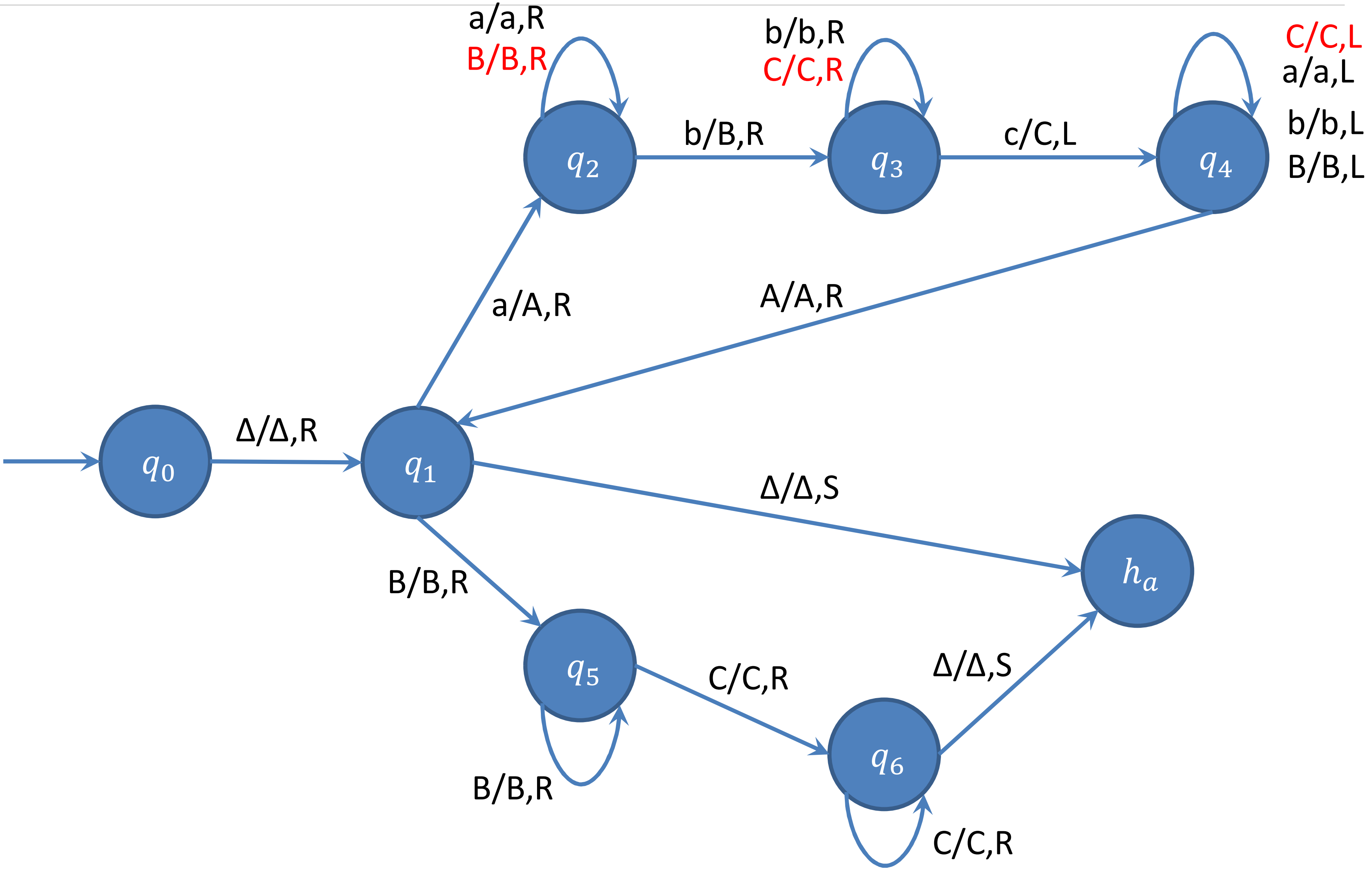
Design a Turing machine for accepting $\{a^n b^n \mid n \geq 1\}$



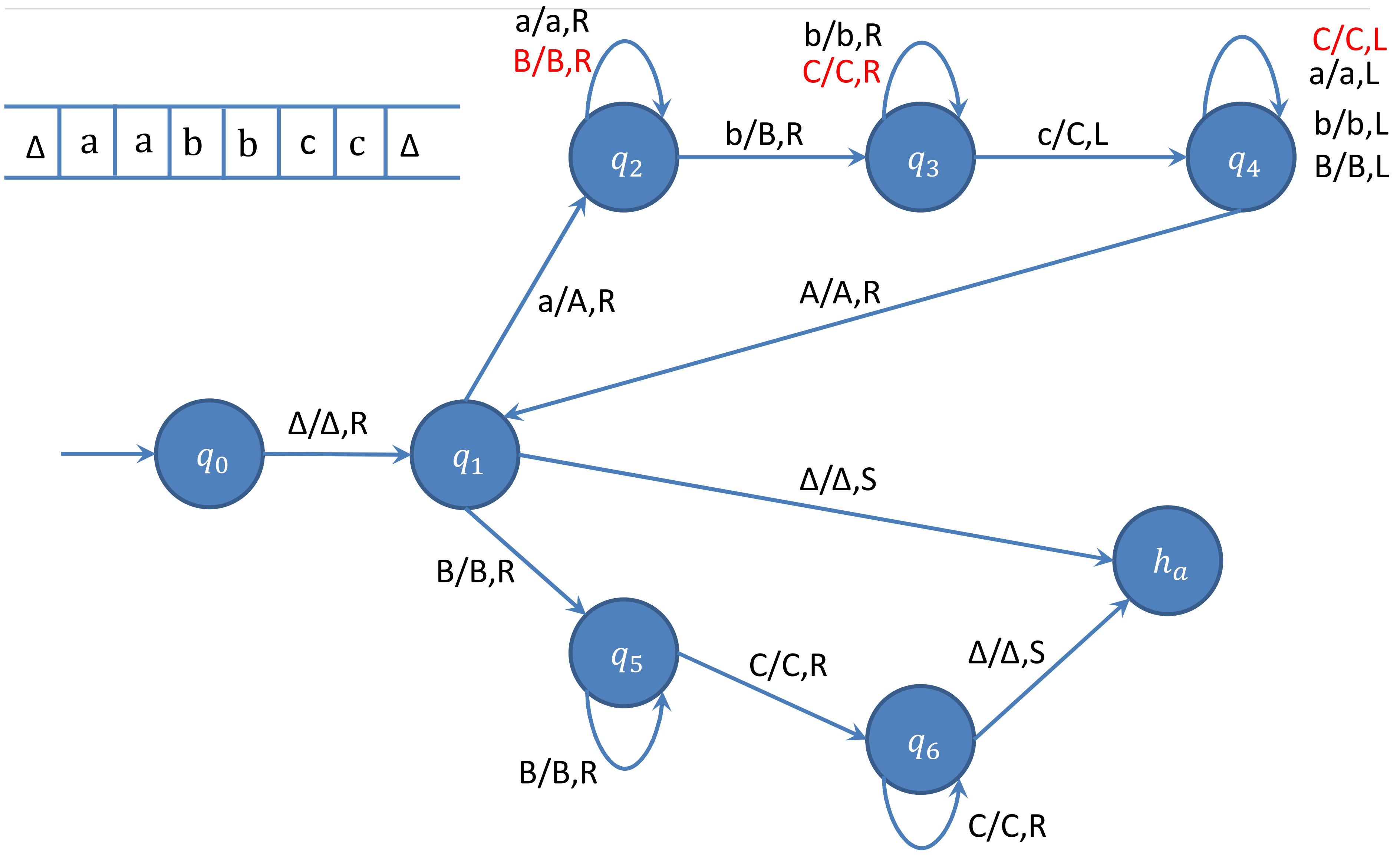
Turing Machine Accepting $\{a^n b^n c^n | \, n \geq 0 \}$

Design a Turing machine for accepting $\{a^n b^n c^n \mid n \geq 0\}$

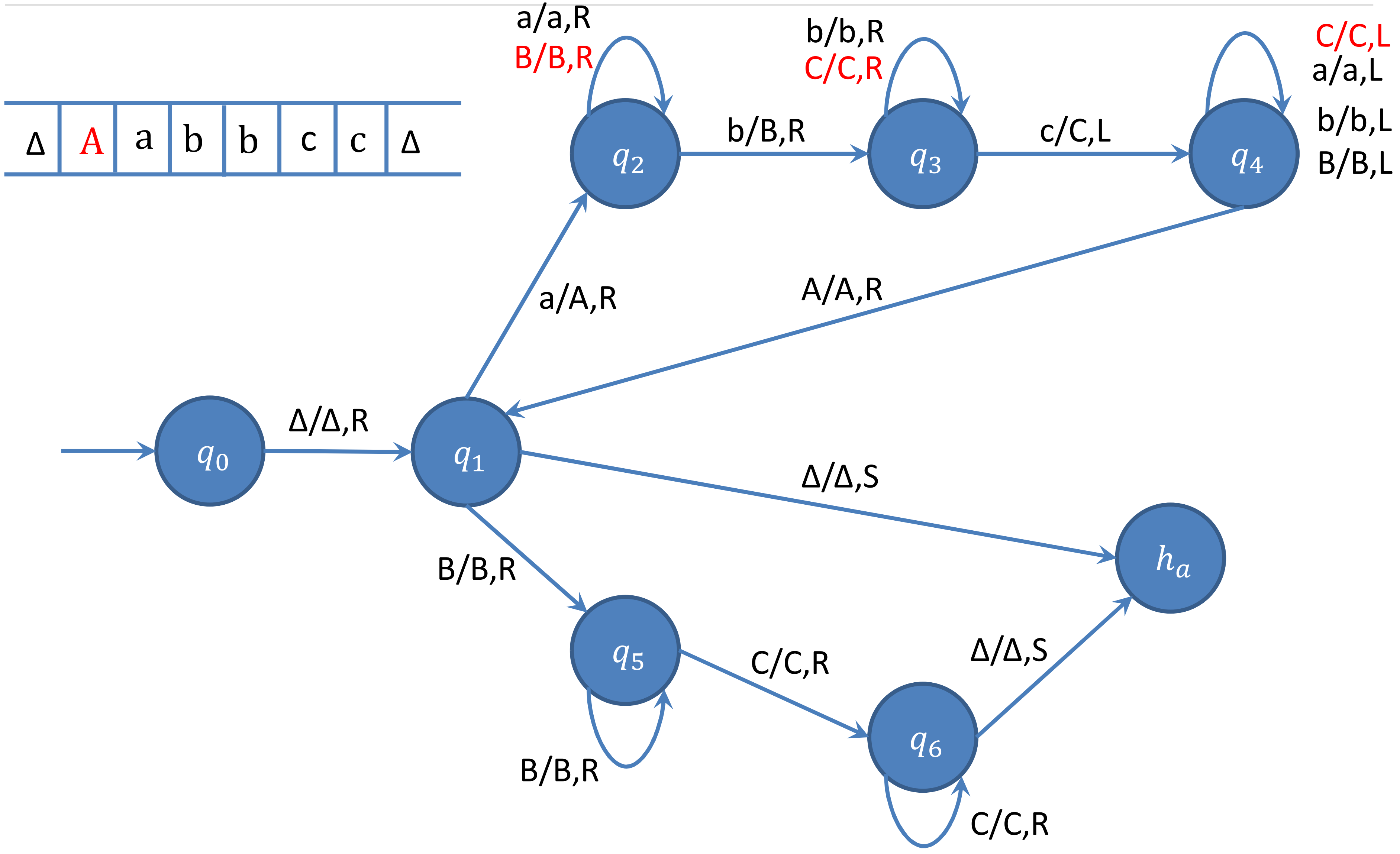
Design a Turing machine for accepting $\{a^n b^n c^n \mid n \geq 0\}$



Design a Turing machine for accepting $\{a^n b^n c^n \mid n \geq 0\}$

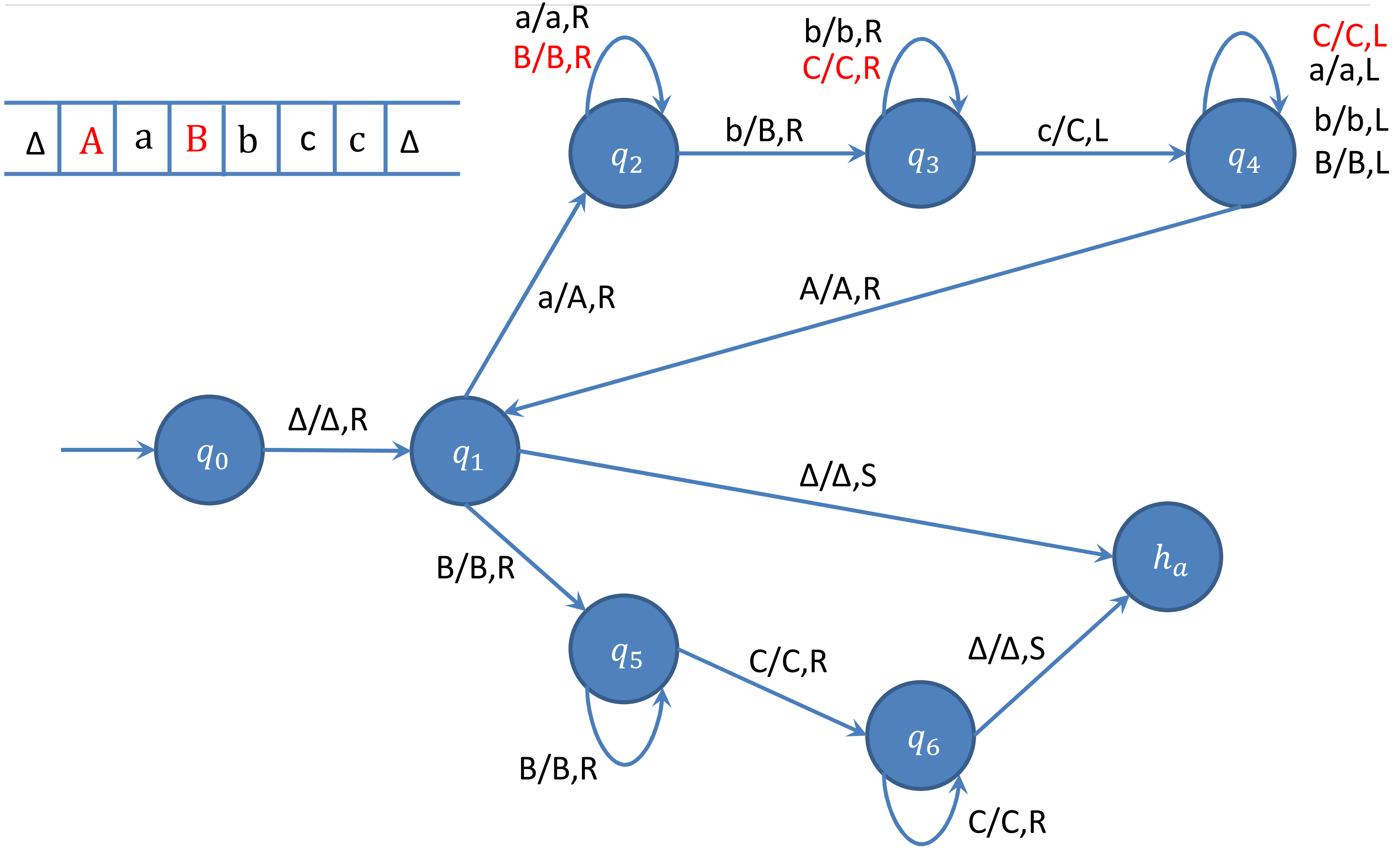


Design a Turing machine for accepting $\{a^n b^n c^n \mid n \geq 0\}$

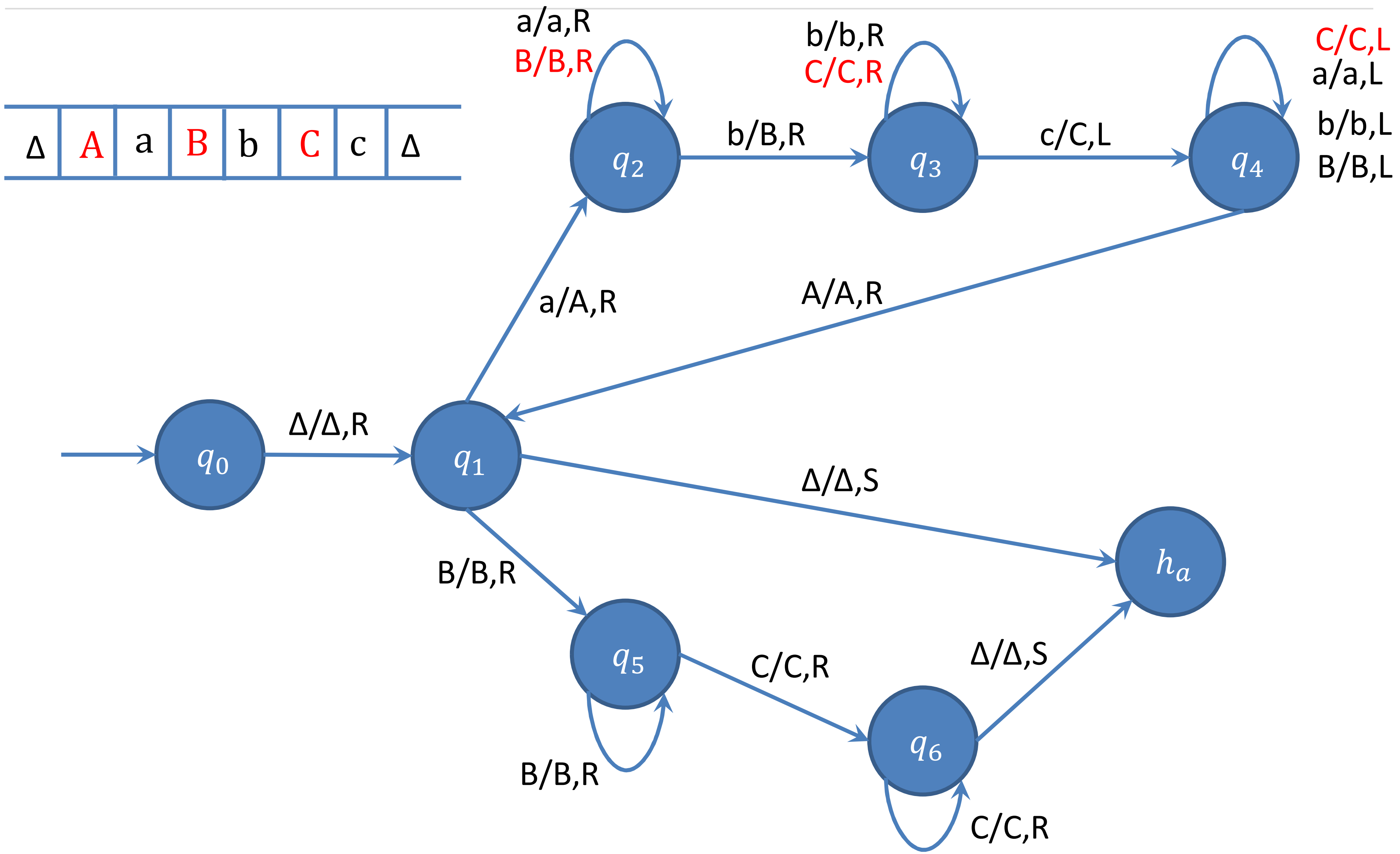


Δ	A	a	b	b	c	c	Δ

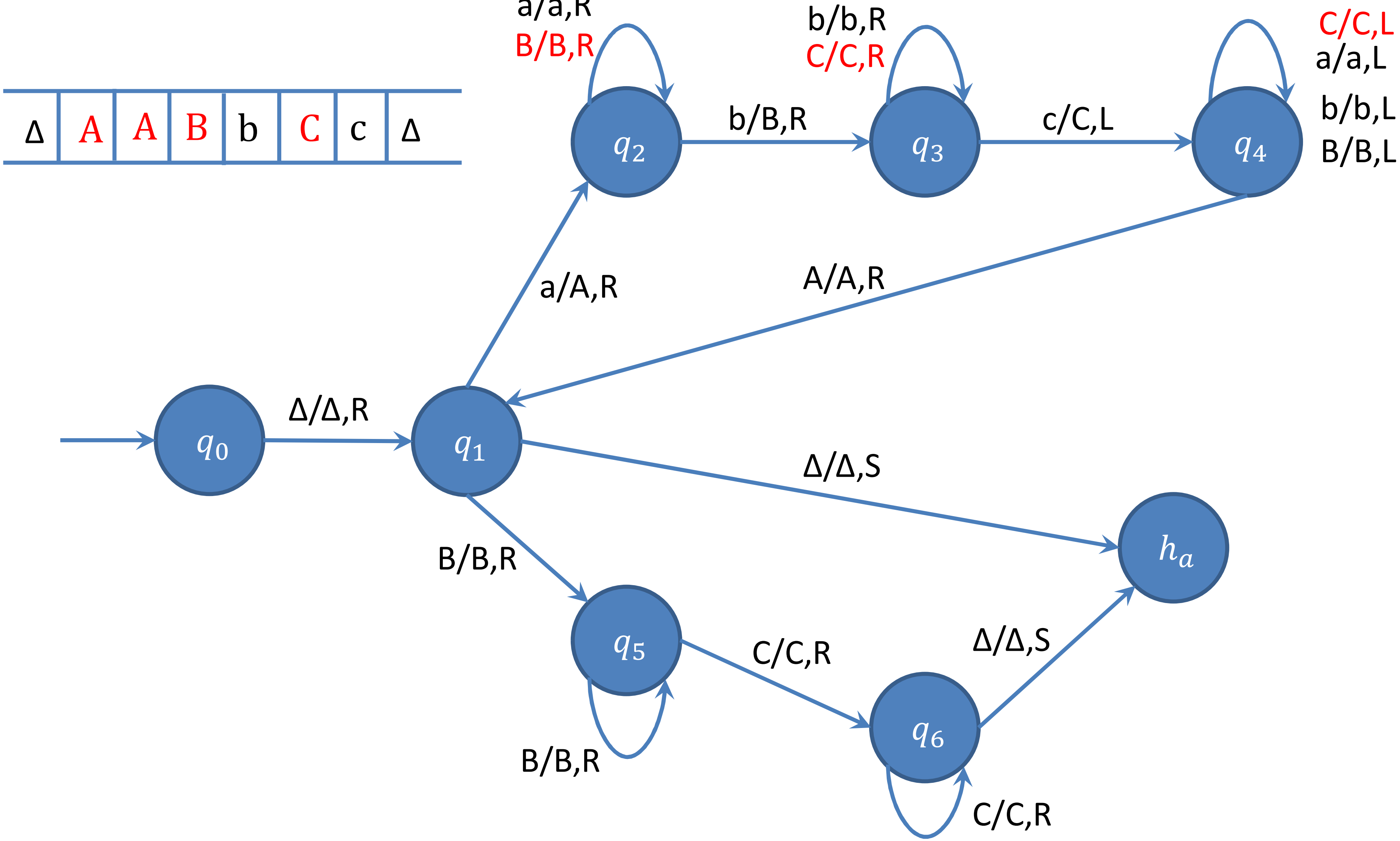
Design a Turing machine for accepting $\{a^n b^n c^n \mid n \geq 0\}$



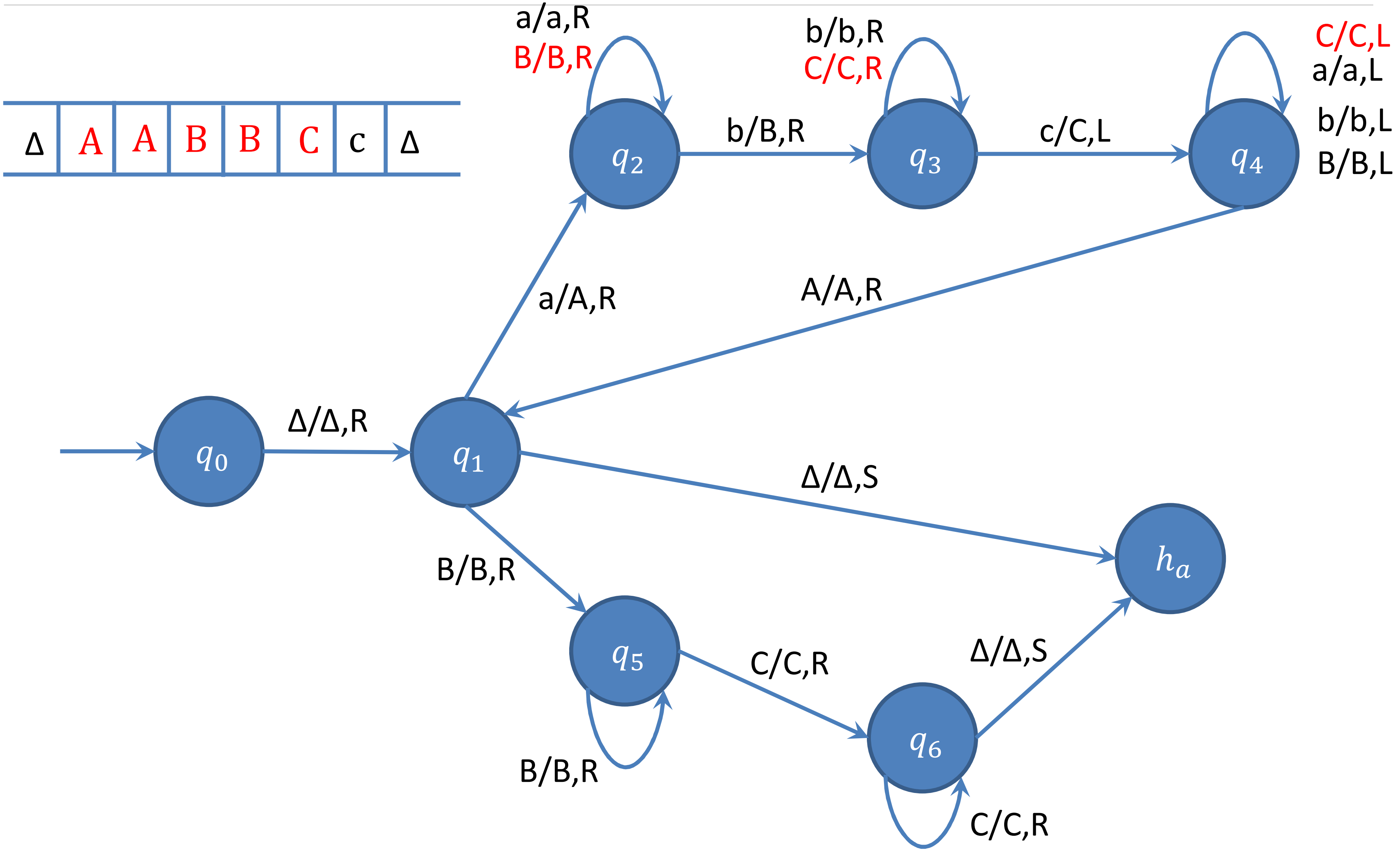
Design a Turing machine for accepting $\{a^n b^n c^n \mid n \geq 0\}$



Design a Turing machine for accepting $\{a^n b^n c^n \mid n \geq 0\}$

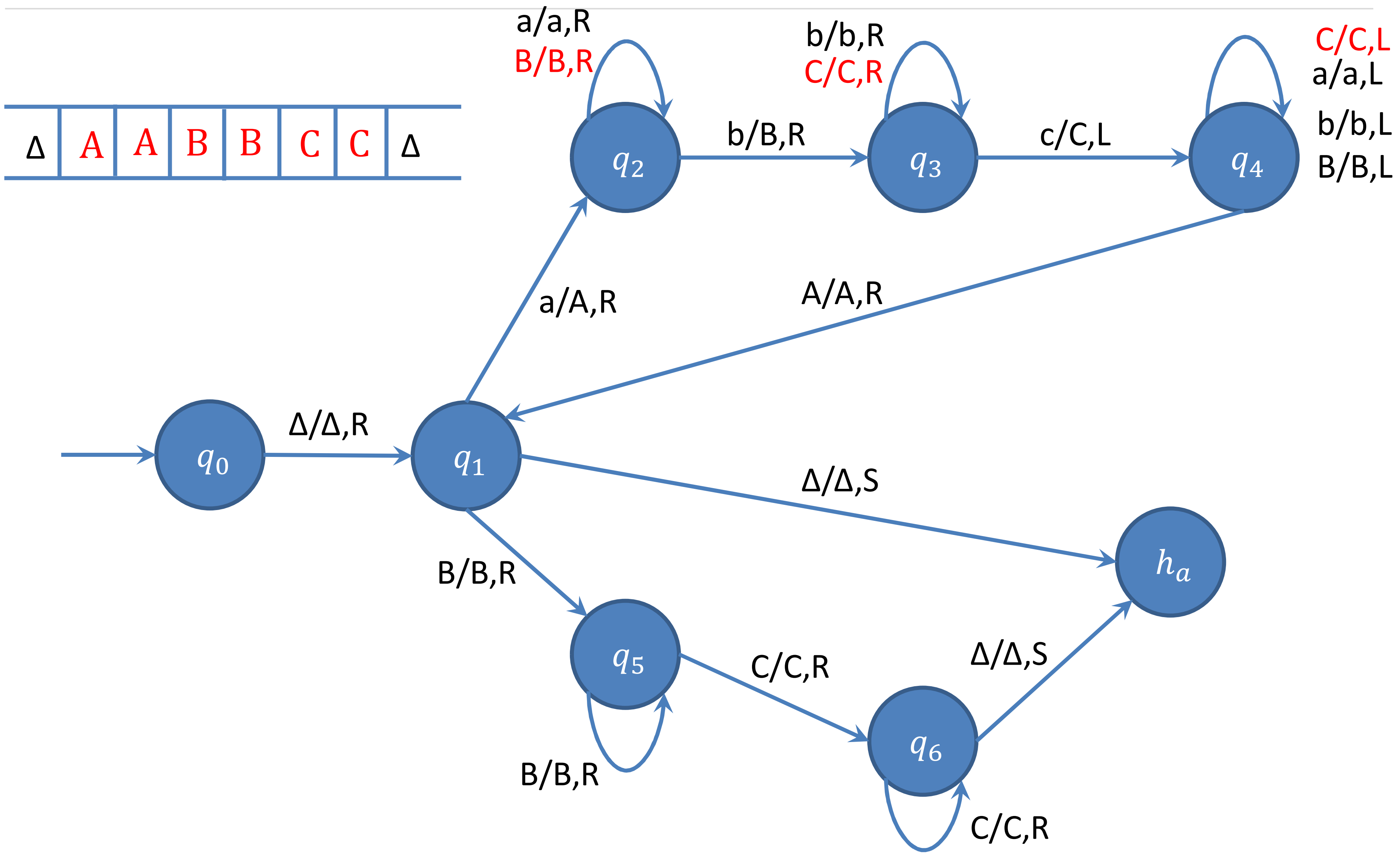


Design a Turing machine for accepting $\{a^n b^n c^n \mid n \geq 0\}$



Δ	A	A	B	B	C	c	Δ

Design a Turing machine for accepting $\{a^n b^n c^n \mid n \geq 0\}$



Turing Machine Accepting Palindrome strings of Even & Odd length

Design a TM for accepting Palindrome strings of even & odd length.

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Logic:

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Logic:

Tape

Δ	a	b	b	a	Δ

Even Length

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Tape

Δ	a	b	b	a	Δ

Even Length

Logic:

1. Read leftmost symbol , replace it with Δ

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Tape

Δ	Δ	b	b	a	Δ

Even Length

Logic:

1. Read leftmost symbol , replace it with Δ

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Tape

Δ	Δ	b	b	a	Δ

Even Length

Logic:

1. Read leftmost symbol , replace it with Δ
2. Keep moving to the right until the rightmost symbol,

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Tape

Δ	Δ	b	b	a	Δ

Even Length

Logic:

1. Read leftmost symbol , replace it with Δ
2. Keep moving to the right until the rightmost symbol,
   If matched with leftmost symbol then replace it with Δ

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Tape

Δ	Δ	b	b	a	Δ

Even Length

Logic:

1. Read leftmost symbol , replace it with Δ
2. Keep moving to the right until the rightmost symbol,
   If matched with leftmost symbol then replace it with Δ
   Otherwise : REJECT the String

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Tape

Δ	Δ	b	b	Δ	Δ

Even Length

Logic:

1. Read leftmost symbol , replace it with Δ
2. Keep moving to the right until the rightmost symbol,

   If matched with leftmost symbol then replace it with Δ

   Otherwise : REJECT the String

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Tape

Δ	Δ	b	b	Δ	Δ

Even Length

Logic:

1. Read leftmost symbol , replace it with Δ
2. Keep moving to the right until the rightmost symbol,
   If matched with leftmost symbol then replace it with Δ
   Otherwise : REJECT the String
3. Repeat the above steps until no more symbol left in string.

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Tape

Δ	Δ	Δ	b	Δ	Δ

Even Length

Logic:

1. Read leftmost symbol , replace it with Δ
2. Keep moving to the right until the rightmost symbol,

    If matched with leftmost symbol then replace it with Δ

    Otherwise : REJECT the String
3. Repeat the above steps until no more symbol left in string.

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

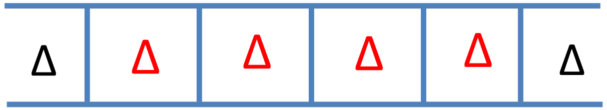
a a

b a b

a b b a

b a b a b

Tape



Δ	Δ	Δ	Δ	Δ	Δ

Even Length

Logic:

1. Read leftmost symbol , replace it with Δ
2. Keep moving to the right until the rightmost symbol,
   If matched with leftmost symbol then replace it with Δ
   Otherwise : REJECT the String
3. Repeat the above steps until no more symbol left in string.

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Tape

Δ	Δ	Δ	Δ	Δ	Δ

Even Length

Δ	b	a	b	a	b	Δ

Odd Length

Logic:

1. Read leftmost symbol , replace it with Δ
2. Keep moving to the right until the rightmost symbol,

   If matched with leftmost symbol then replace it with Δ

   Otherwise : REJECT the String
3. Repeat the above steps until no more symbol left in string.

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Tape

Δ	Δ	Δ	Δ	Δ	Δ

Even Length

Δ	Δ	a	b	a	b	Δ

Odd Length

Logic:

1. Read leftmost symbol , replace it with Δ
2. Keep moving to the right until the rightmost symbol,

   If matched with leftmost symbol then replace it with Δ

   Otherwise : REJECT the String
3. Repeat the above steps until no more symbol left in string.

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Tape

Δ	Δ	Δ	Δ	Δ	Δ

Even Length

Δ	Δ	a	b	a	Δ	Δ

Odd Length

Logic:

1. Read leftmost symbol , replace it with Δ
2. Keep moving to the right until the rightmost symbol,

   If matched with leftmost symbol then replace it with Δ

   Otherwise : REJECT the String
3. Repeat the above steps until no more symbol left in string.

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Tape

Δ	Δ	Δ	Δ	Δ	Δ

Even Length

Δ	Δ	Δ	b	a	Δ	Δ

Odd Length

Logic:

1. Read leftmost symbol , replace it with Δ
2. Keep moving to the right until the rightmost symbol,
   If matched with leftmost symbol then replace it with Δ
   Otherwise : REJECT the String
3. Repeat the above steps until no more symbol left in string.

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Tape

Δ	Δ	Δ	Δ	Δ	Δ

Even Length

Δ	Δ	Δ	b	Δ	Δ	Δ

Odd Length

Logic:

1. Read leftmost symbol , replace it with Δ
2. Keep moving to the right until the rightmost symbol,

   If matched with leftmost symbol then replace it with Δ

   Otherwise : REJECT the String
3. Repeat the above steps until no more symbol left in string.

Design a TM for accepting Palindrome strings of even & odd length.

L = Palindrome string

Strings accepting in given L are:

a a

b a b

a b b a

b a b a b

Tape

Δ	Δ	Δ	Δ	Δ	Δ

Even Length

Δ	Δ	Δ	Δ	Δ	Δ	Δ

Odd Length

Logic:

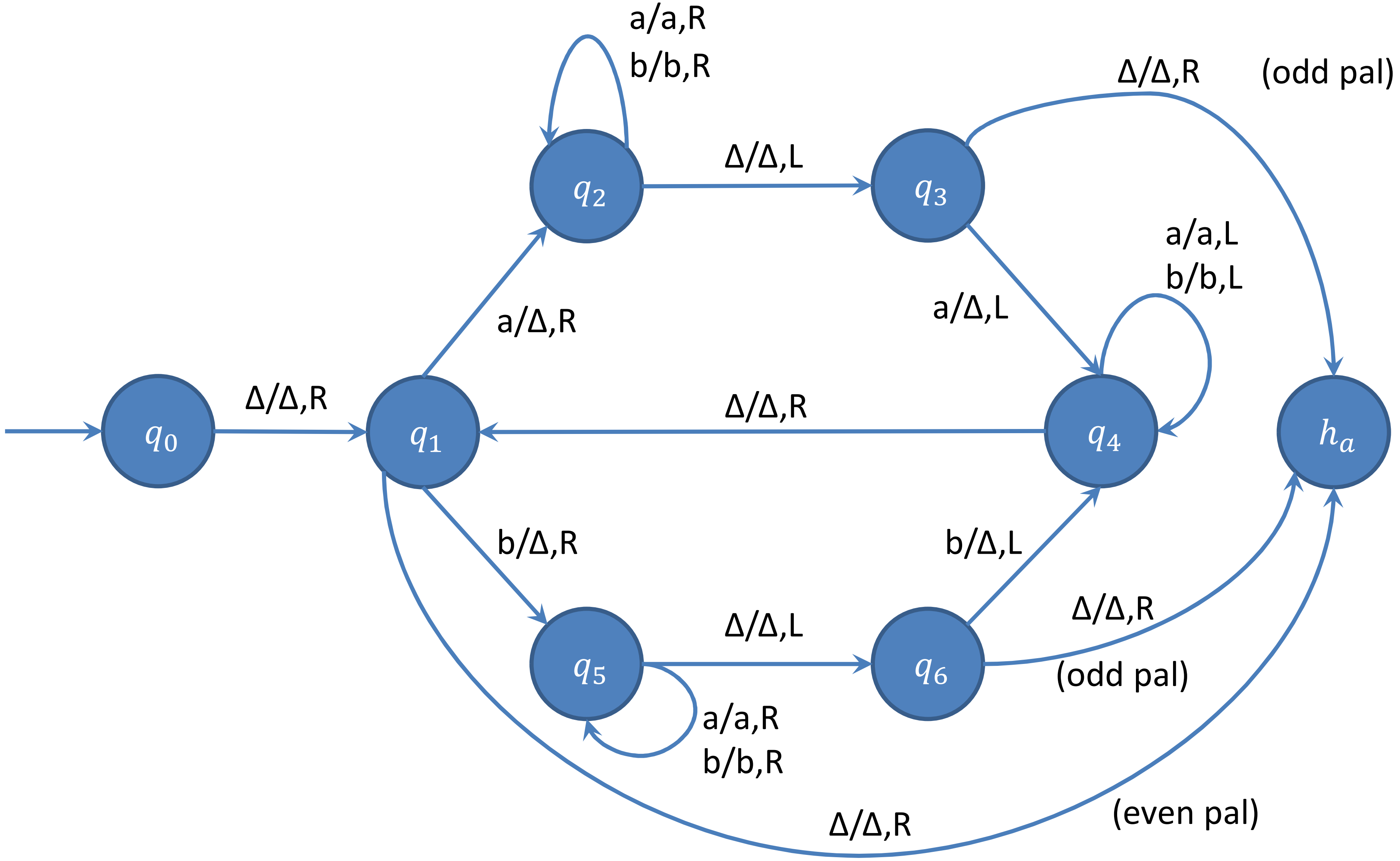
1. Read leftmost symbol , replace it with Δ
2. Keep moving to the right until the rightmost symbol,

   If matched with leftmost symbol then replace it with Δ

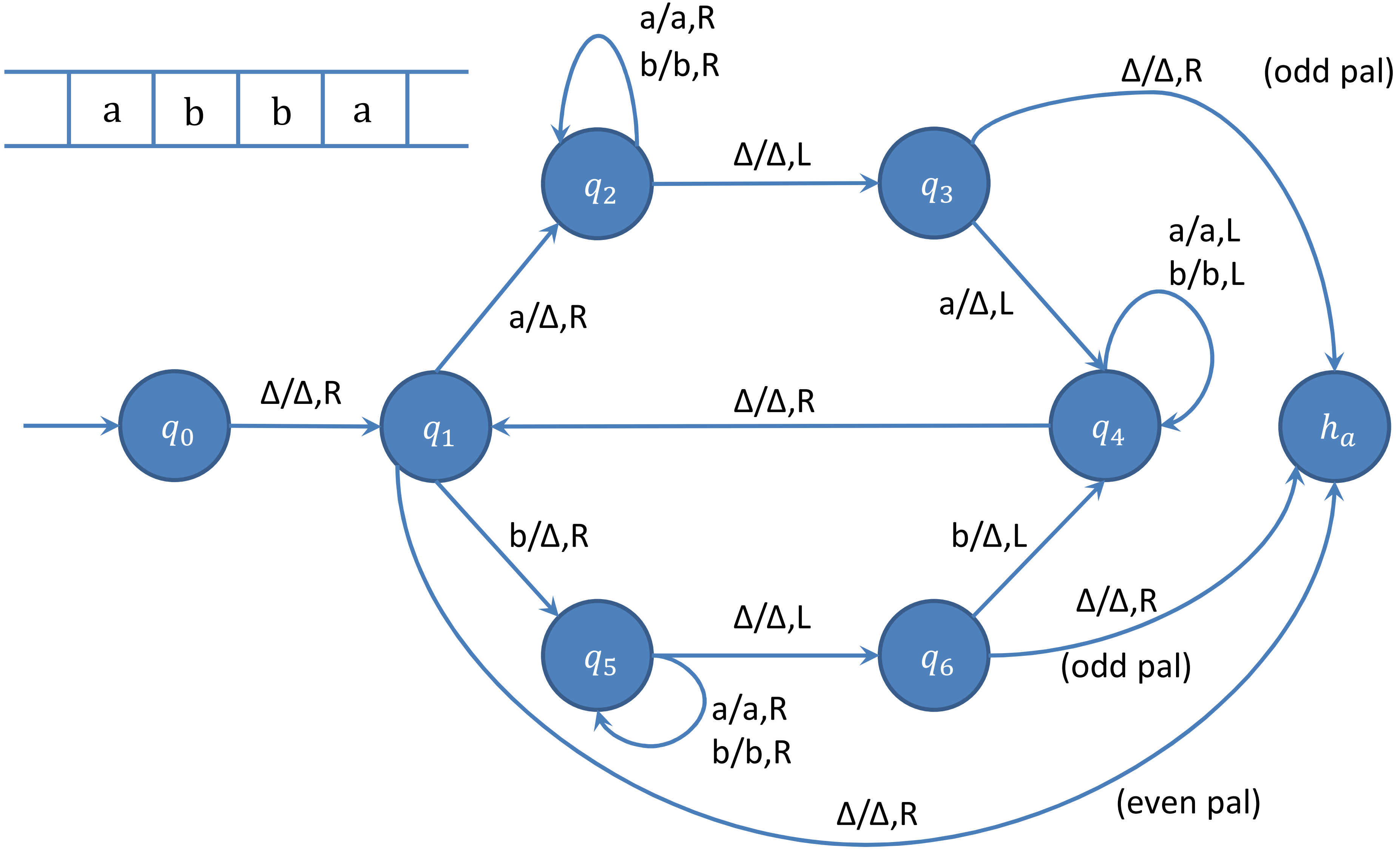
   Otherwise : REJECT the String
3. Repeat the above steps until no more symbol left in string.

Design a TM for accepting Palindrome strings of even & odd length.

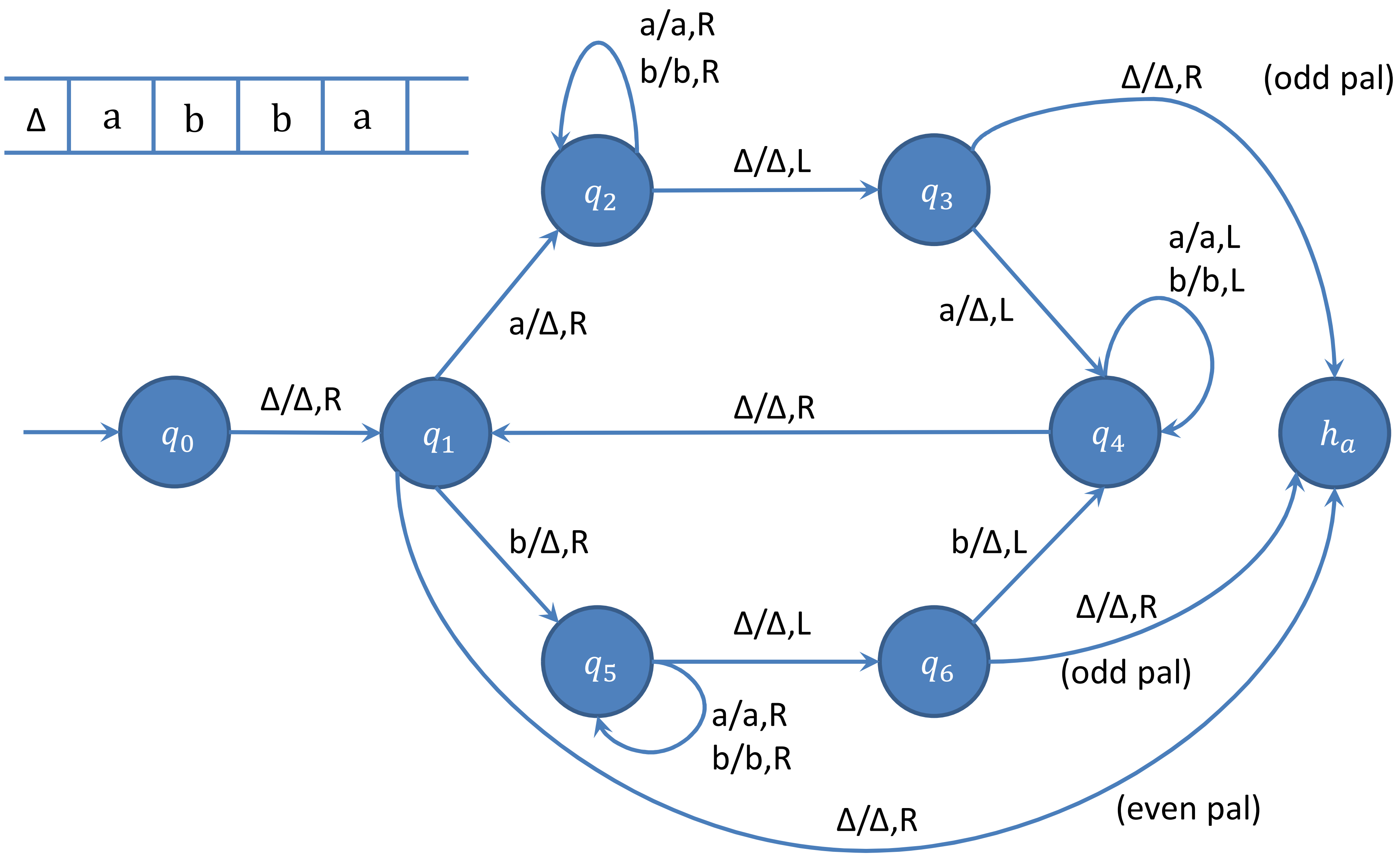
Design a TM for accepting Palindrome strings of even & odd length.



Design a TM for accepting Palindrome strings of even & odd length.

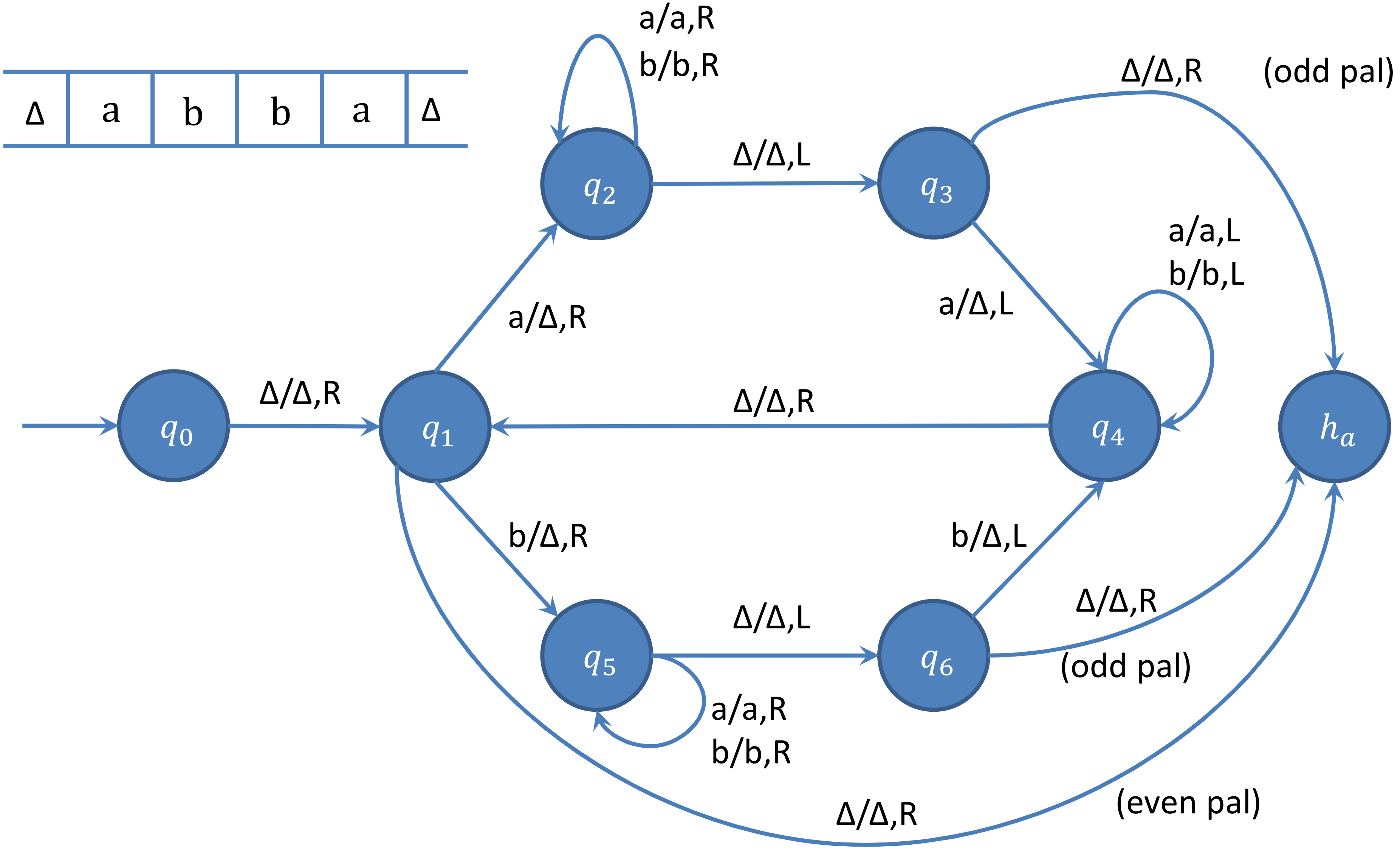


Design a TM for accepting Palindrome strings of even & odd length.

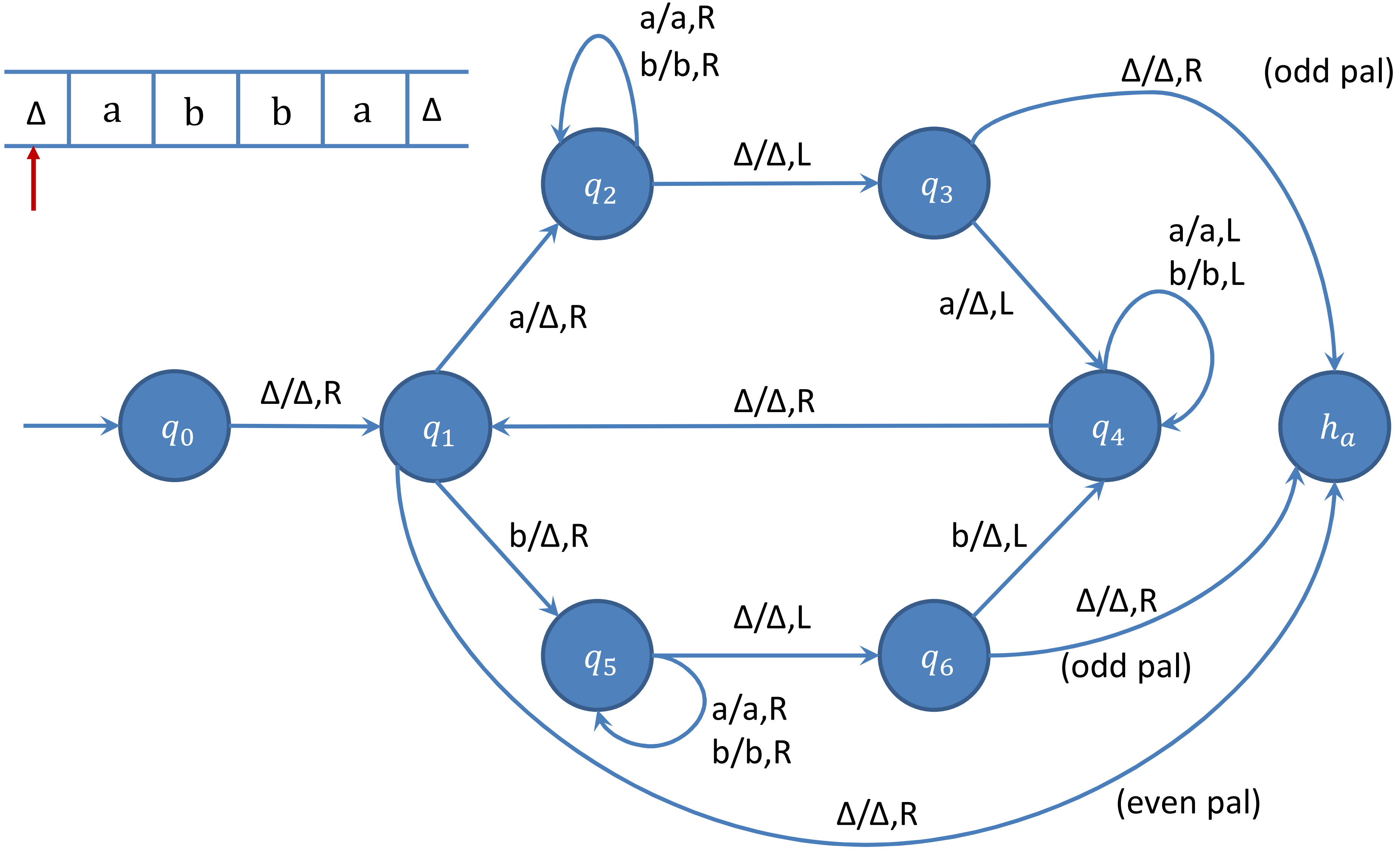


Design a TM for accepting Palindrome strings of even & odd length.

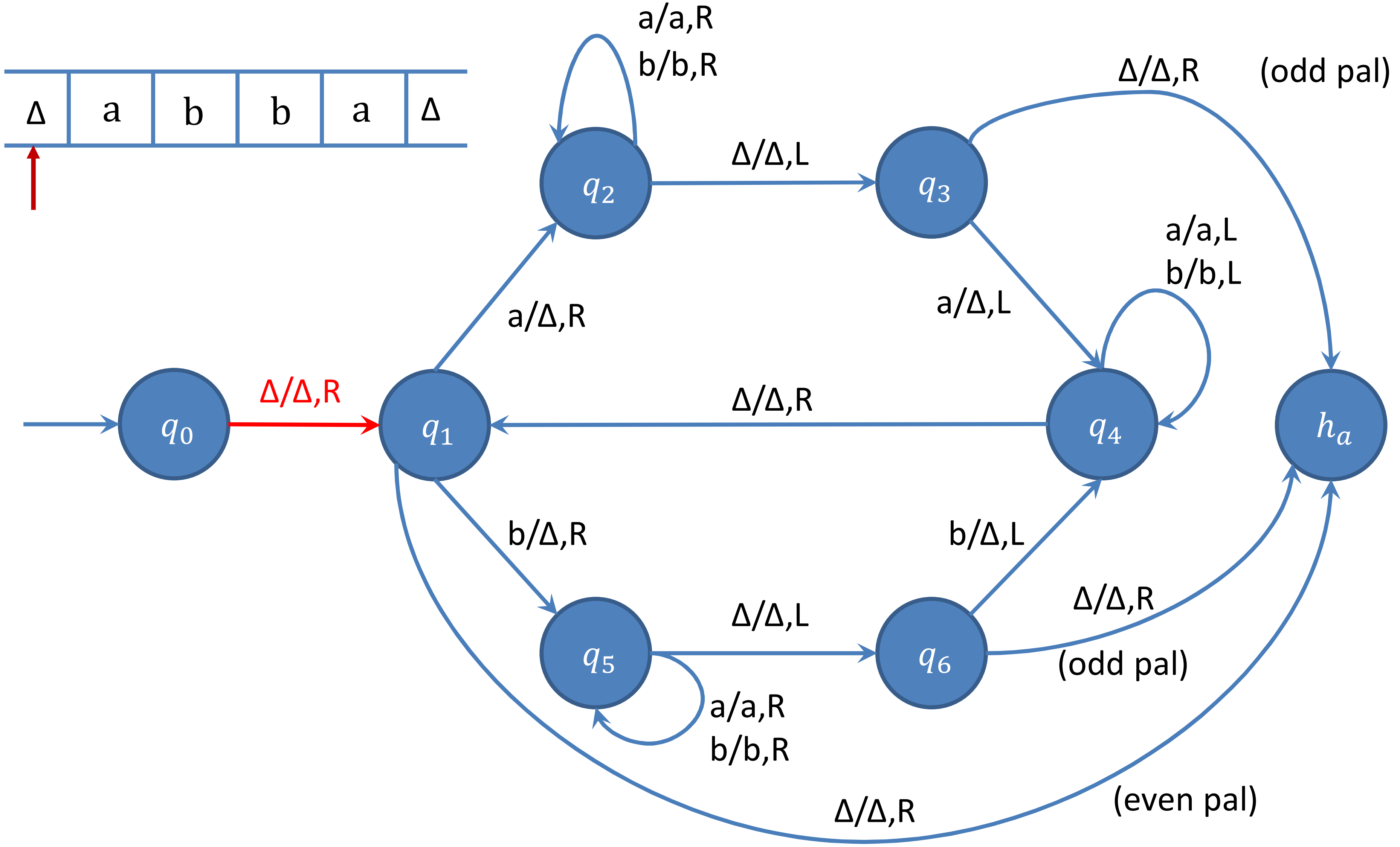
Δ	a	b	b	a	Δ



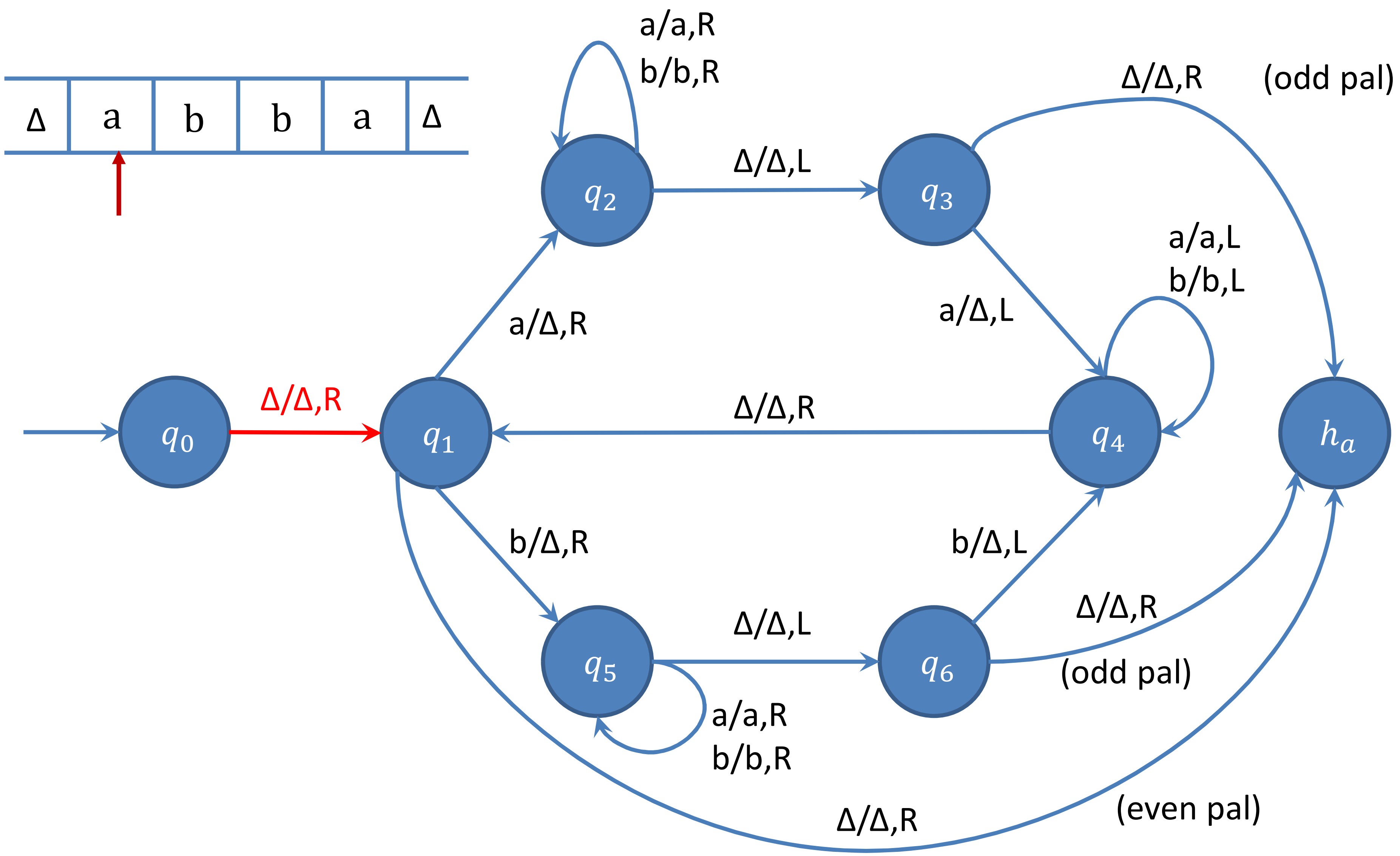
Design a TM for accepting Palindrome strings of even & odd length.



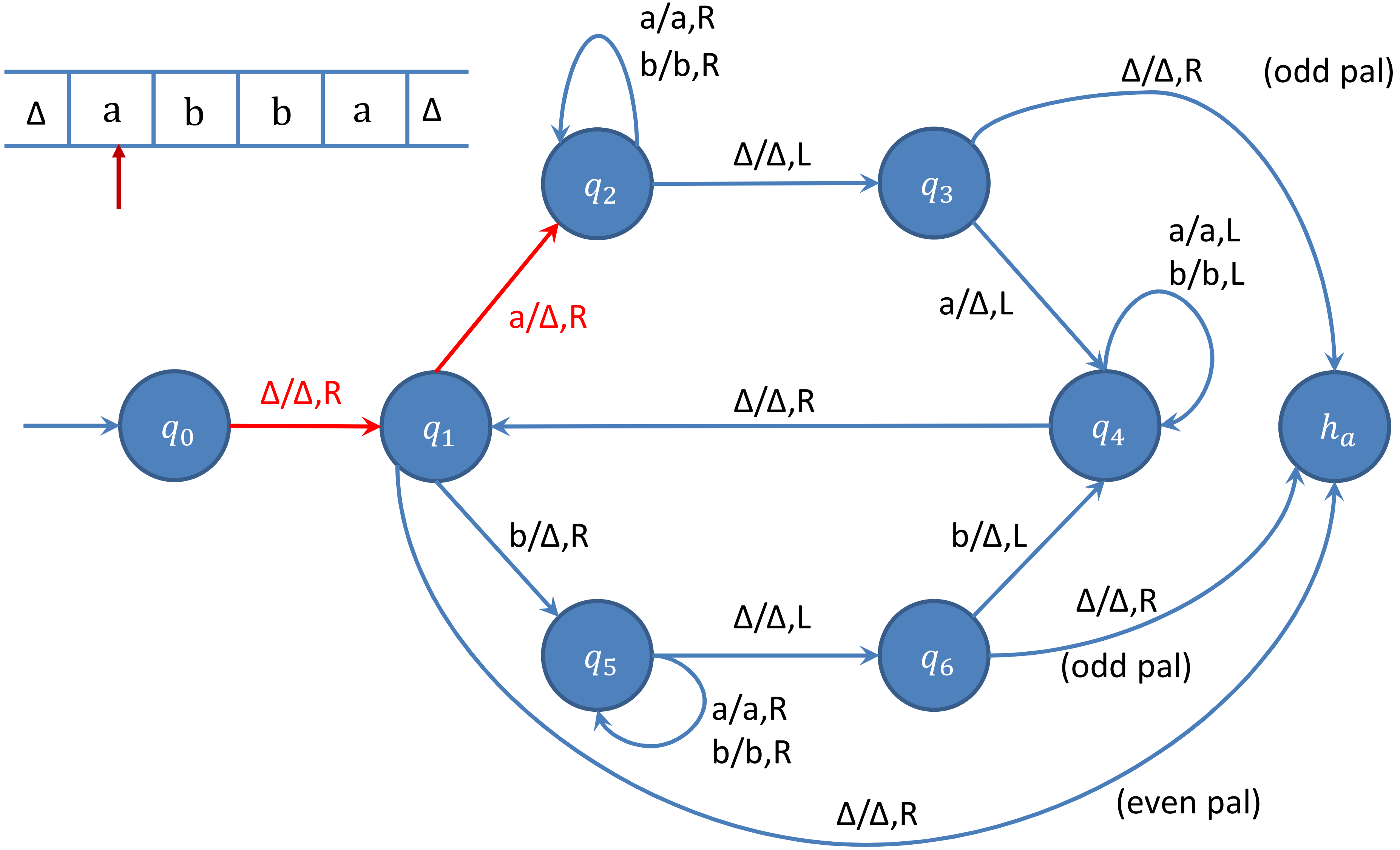
Design a TM for accepting Palindrome strings of even & odd length.



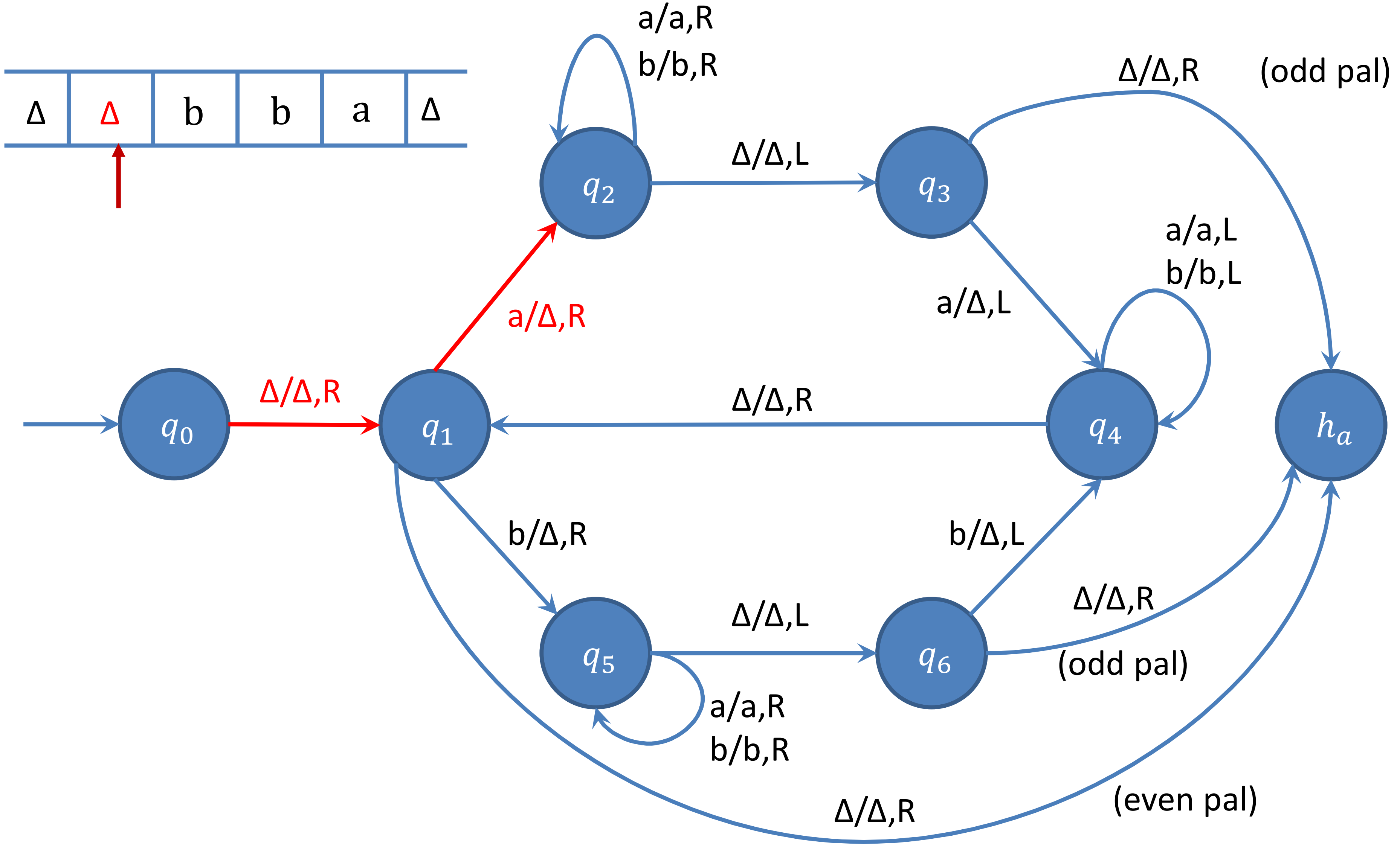
Design a TM for accepting Palindrome strings of even & odd length.



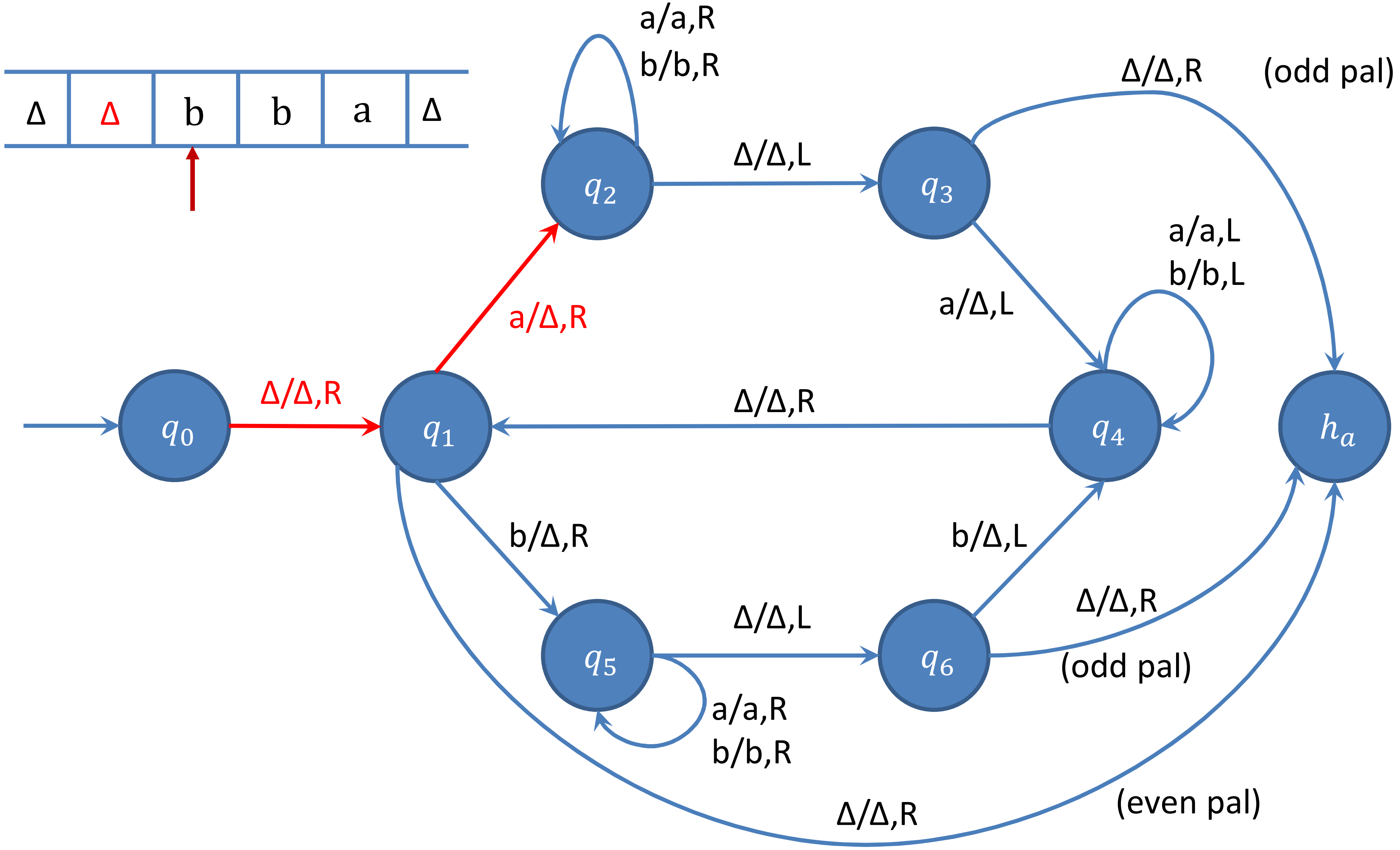
Design a TM for accepting Palindrome strings of even & odd length.



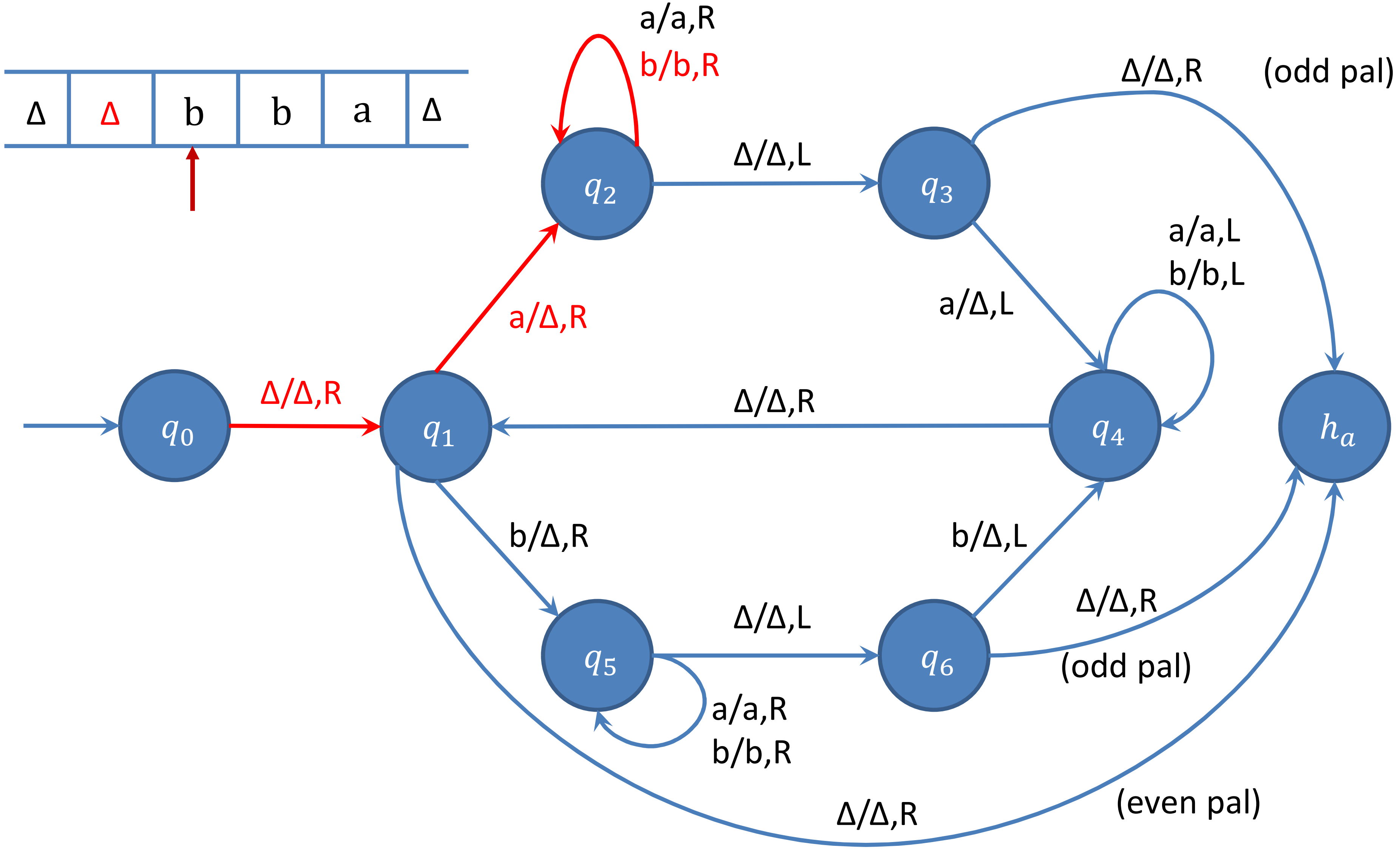
Design a TM for accepting Palindrome strings of even & odd length.



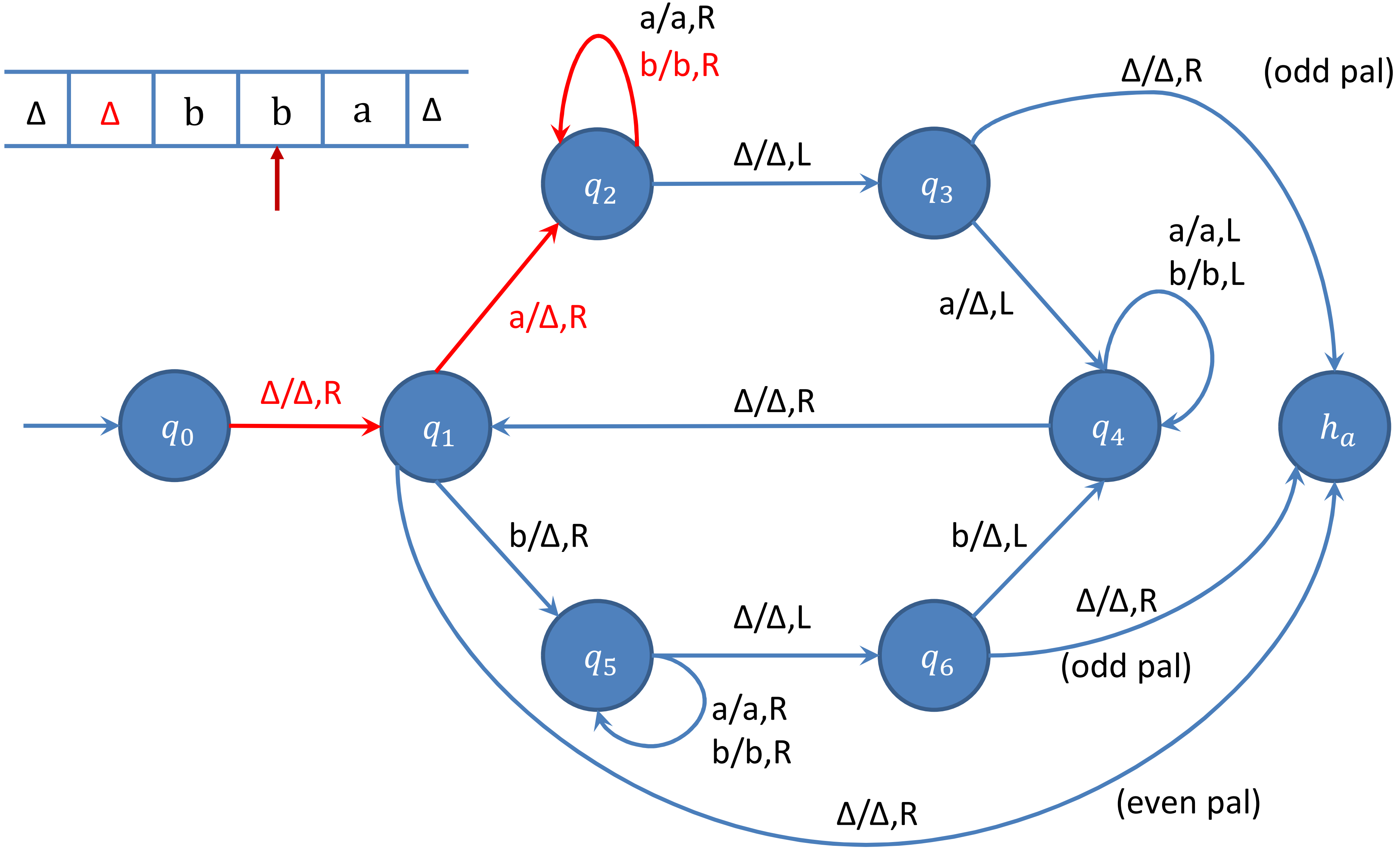
Design a TM for accepting Palindrome strings of even & odd length.



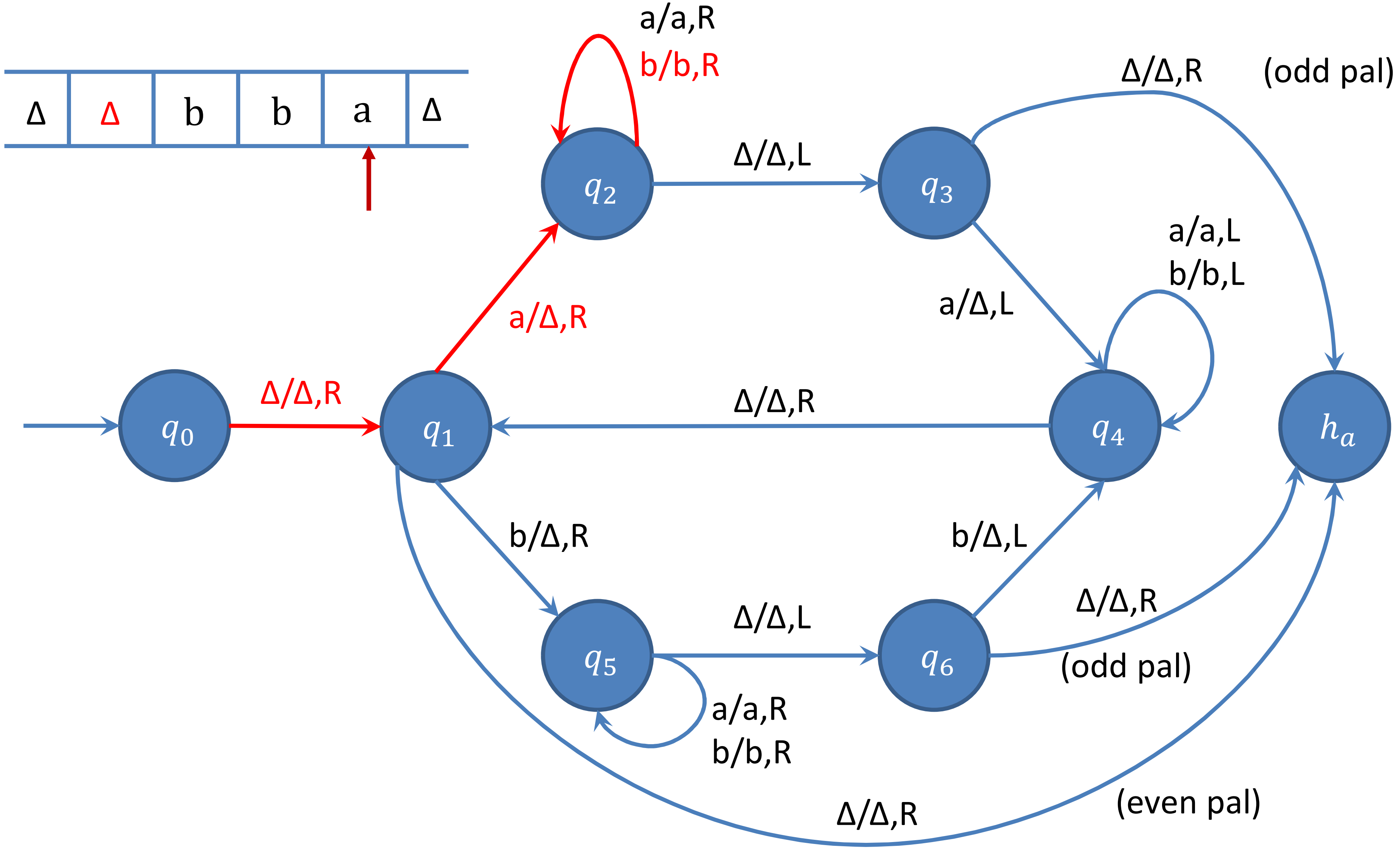
Design a TM for accepting Palindrome strings of even & odd length.



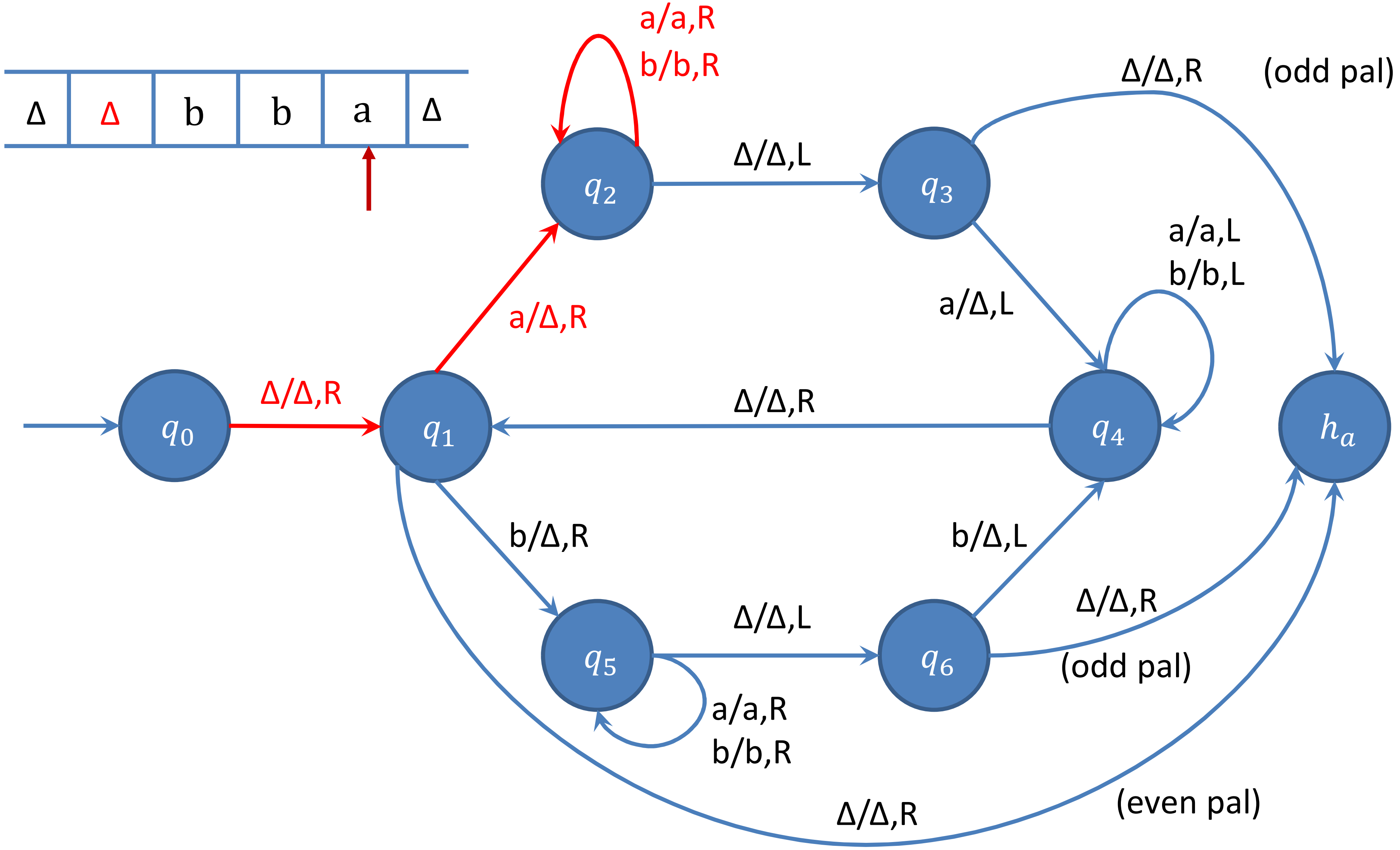
Design a TM for accepting Palindrome strings of even & odd length.



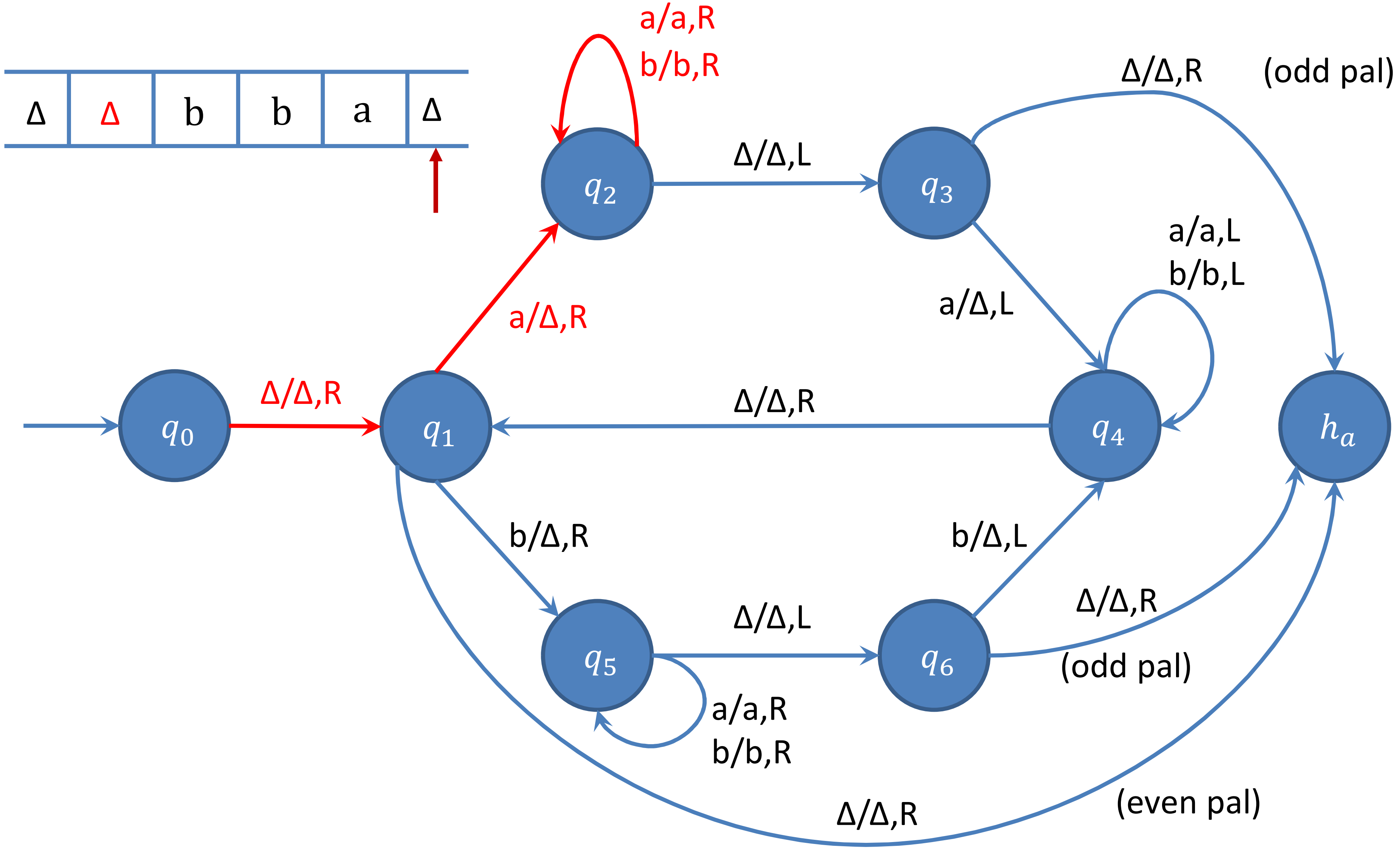
Design a TM for accepting Palindrome strings of even & odd length.



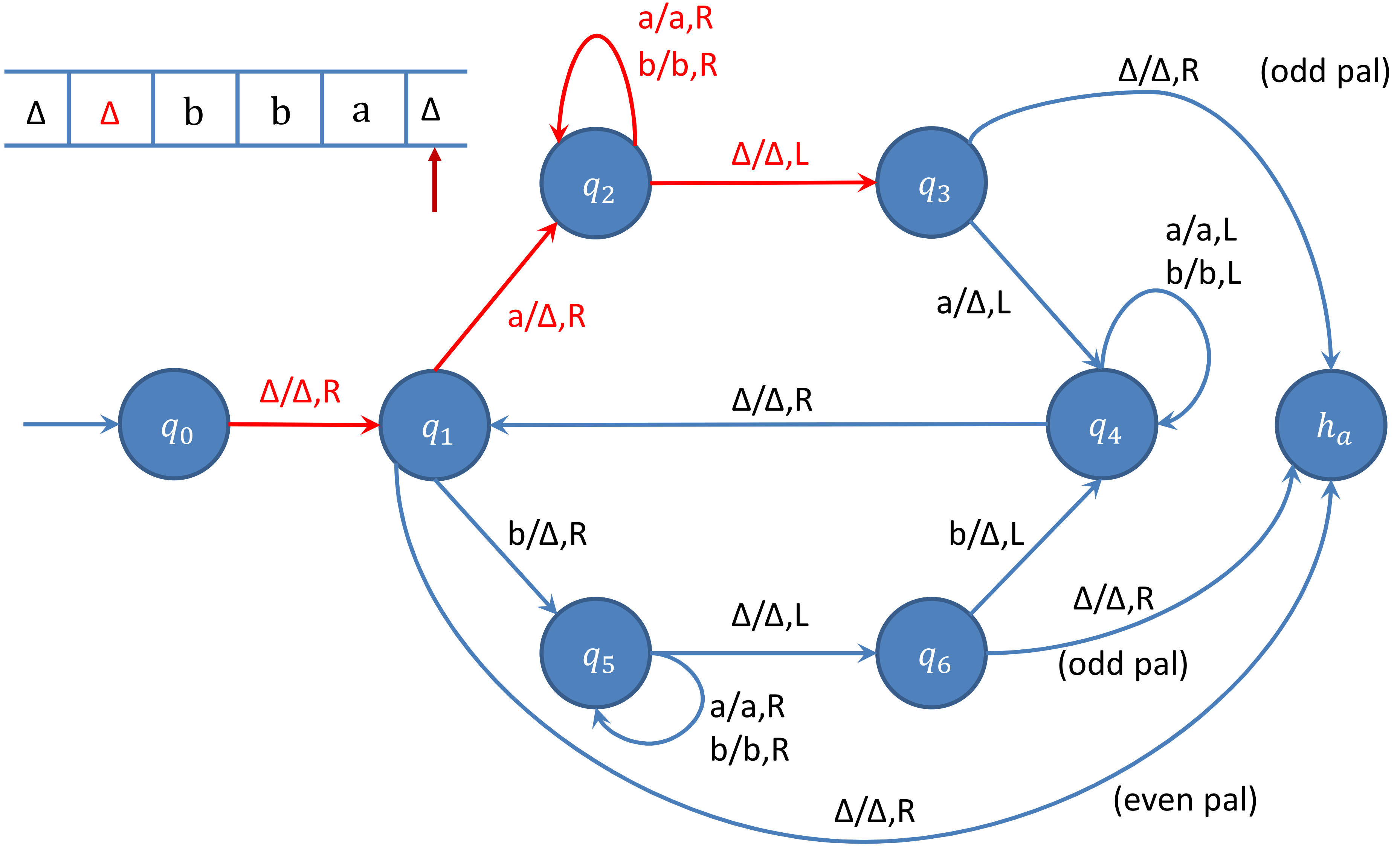
Design a TM for accepting Palindrome strings of even & odd length.



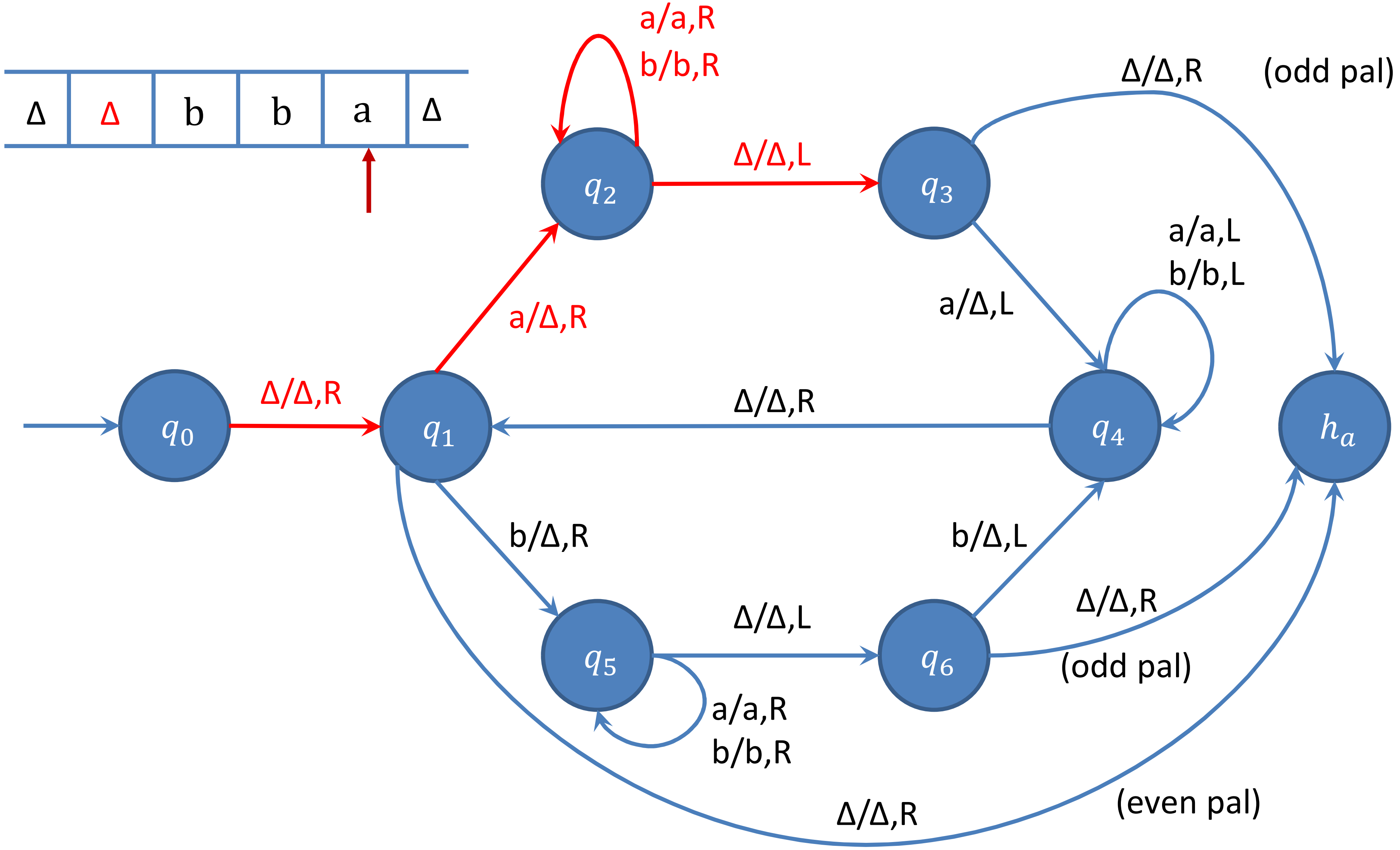
Design a TM for accepting Palindrome strings of even & odd length.



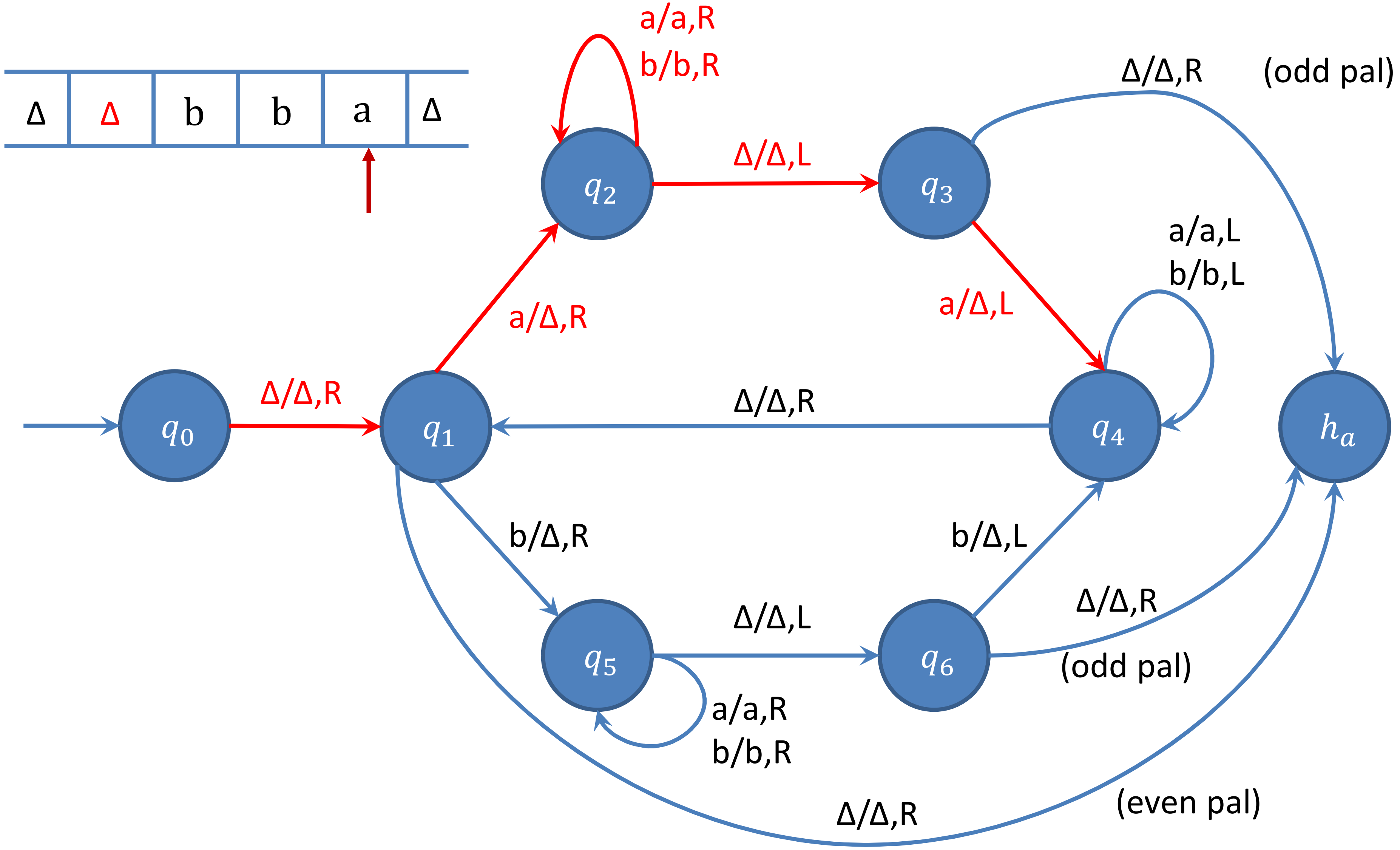
Design a TM for accepting Palindrome strings of even & odd length.



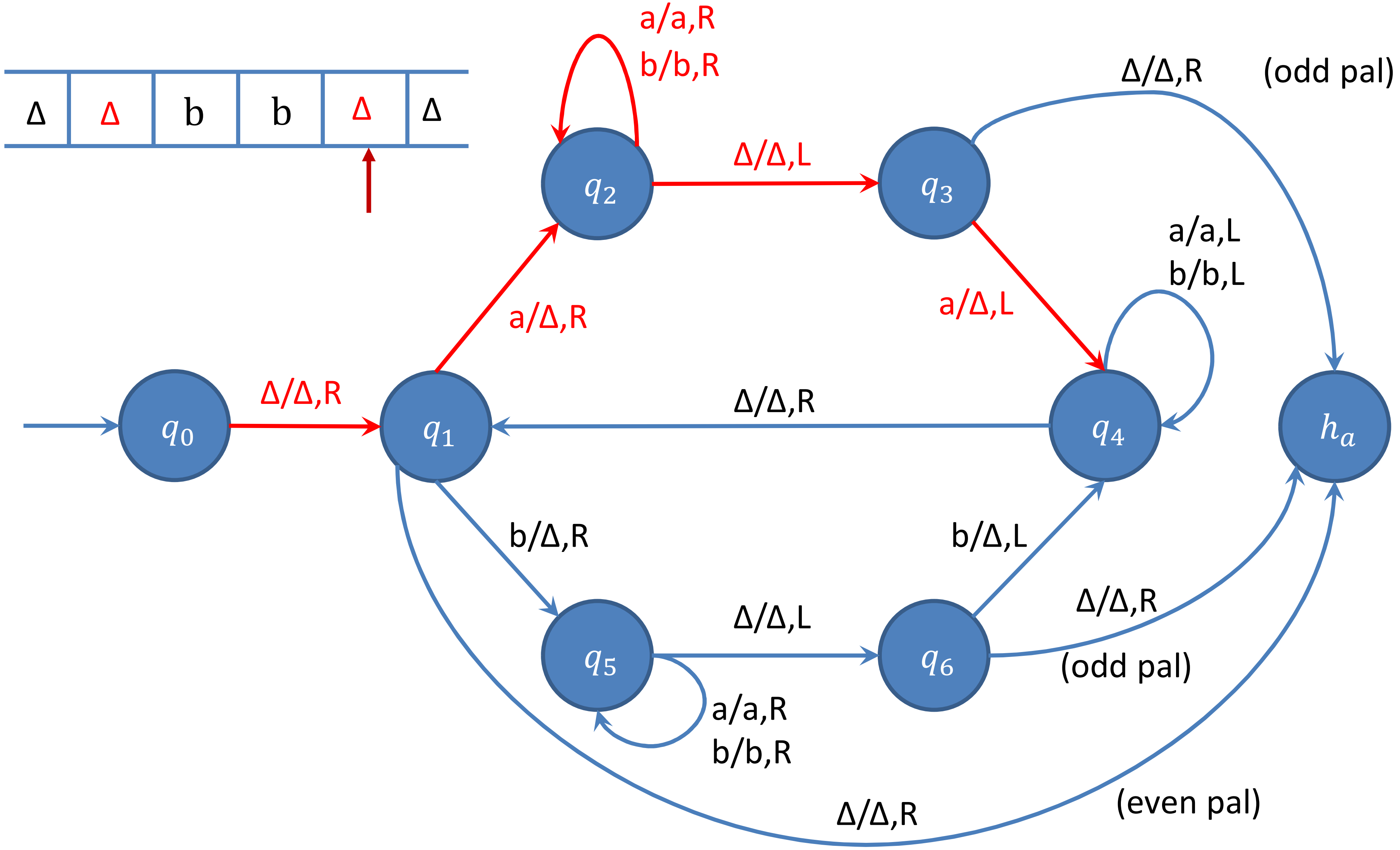
Design a TM for accepting Palindrome strings of even & odd length.



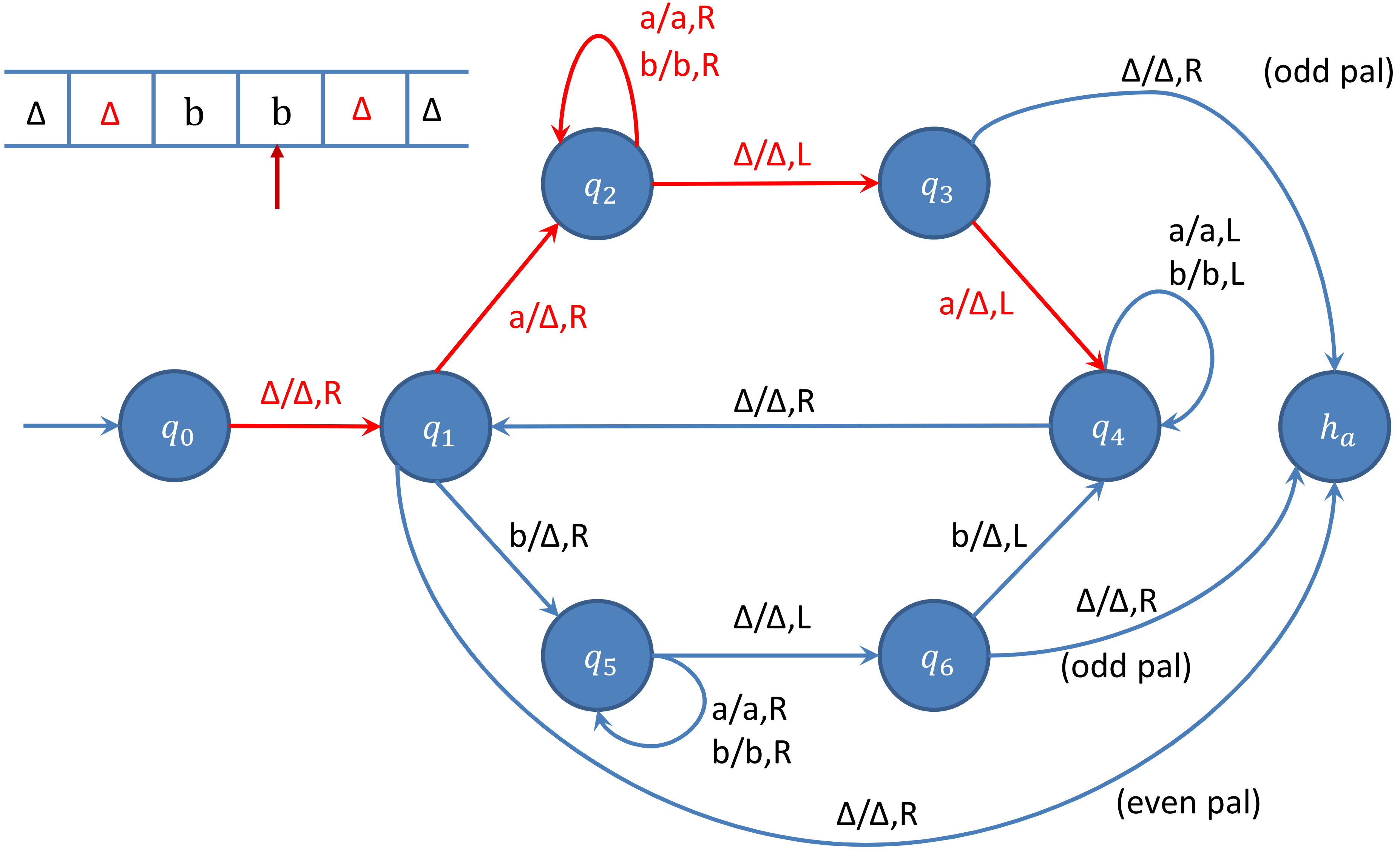
Design a TM for accepting Palindrome strings of even & odd length.



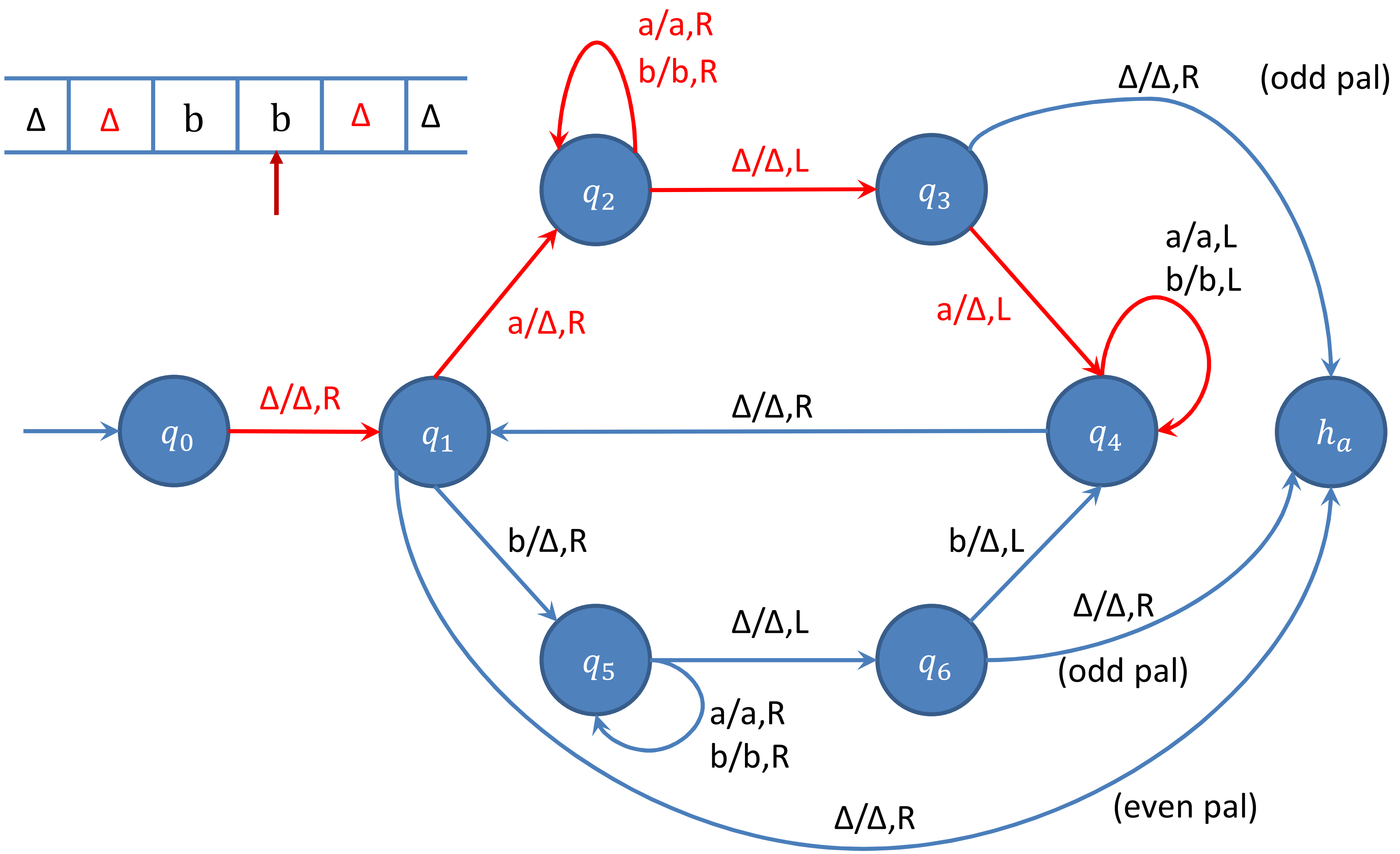
Design a TM for accepting Palindrome strings of even & odd length.



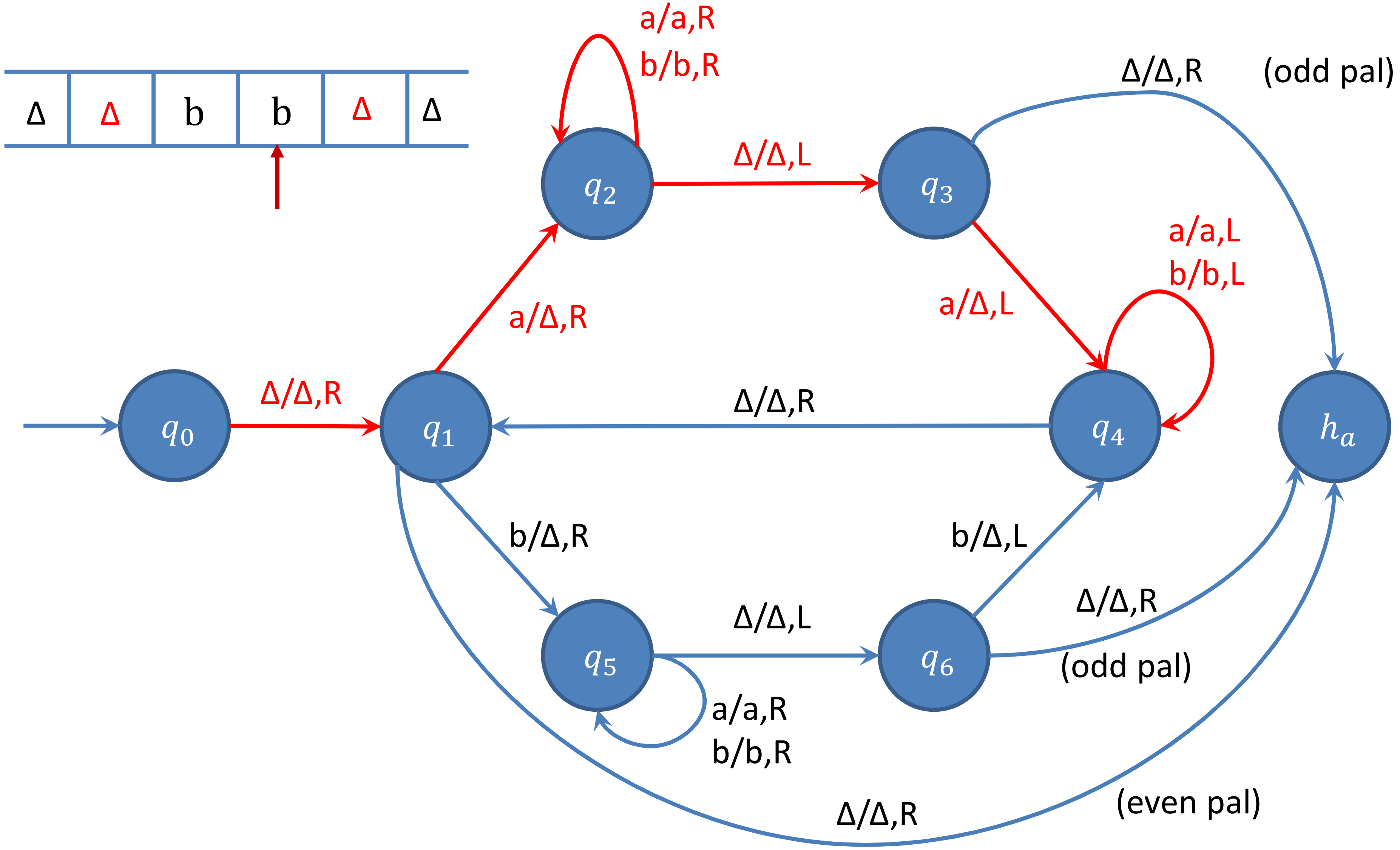
Design a TM for accepting Palindrome strings of even & odd length.



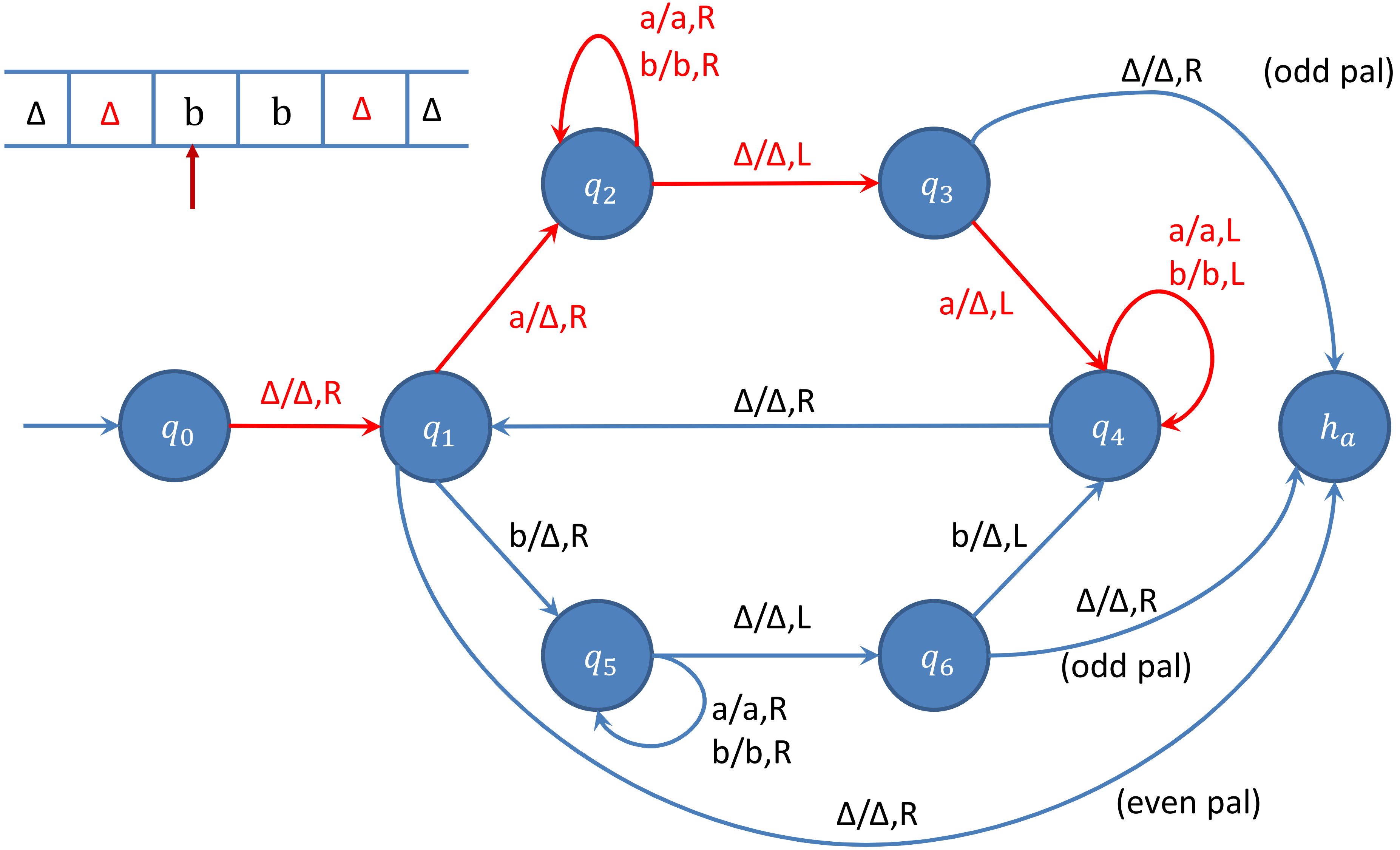
Design a TM for accepting Palindrome strings of even & odd length.



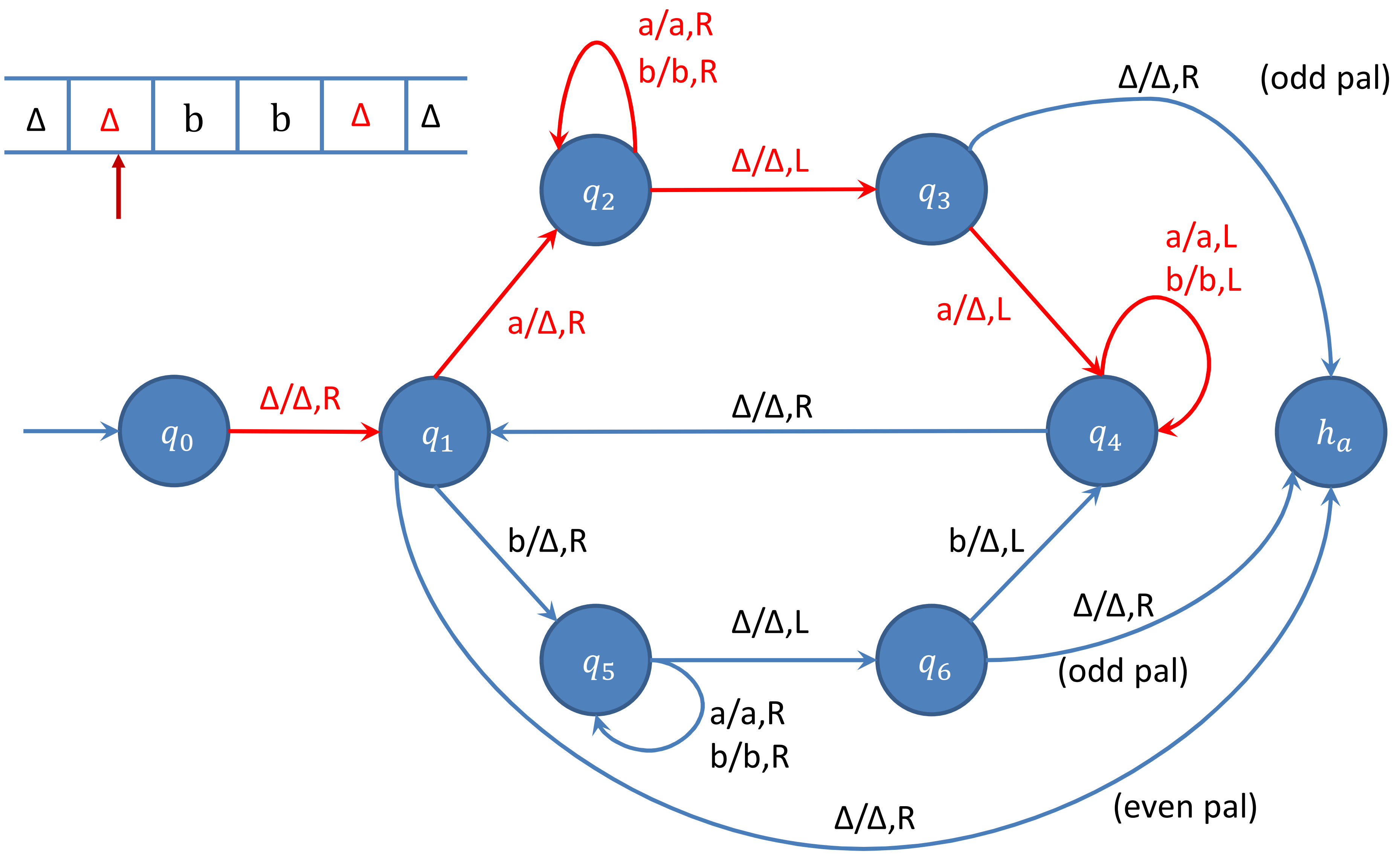
Design a TM for accepting Palindrome strings of even & odd length.



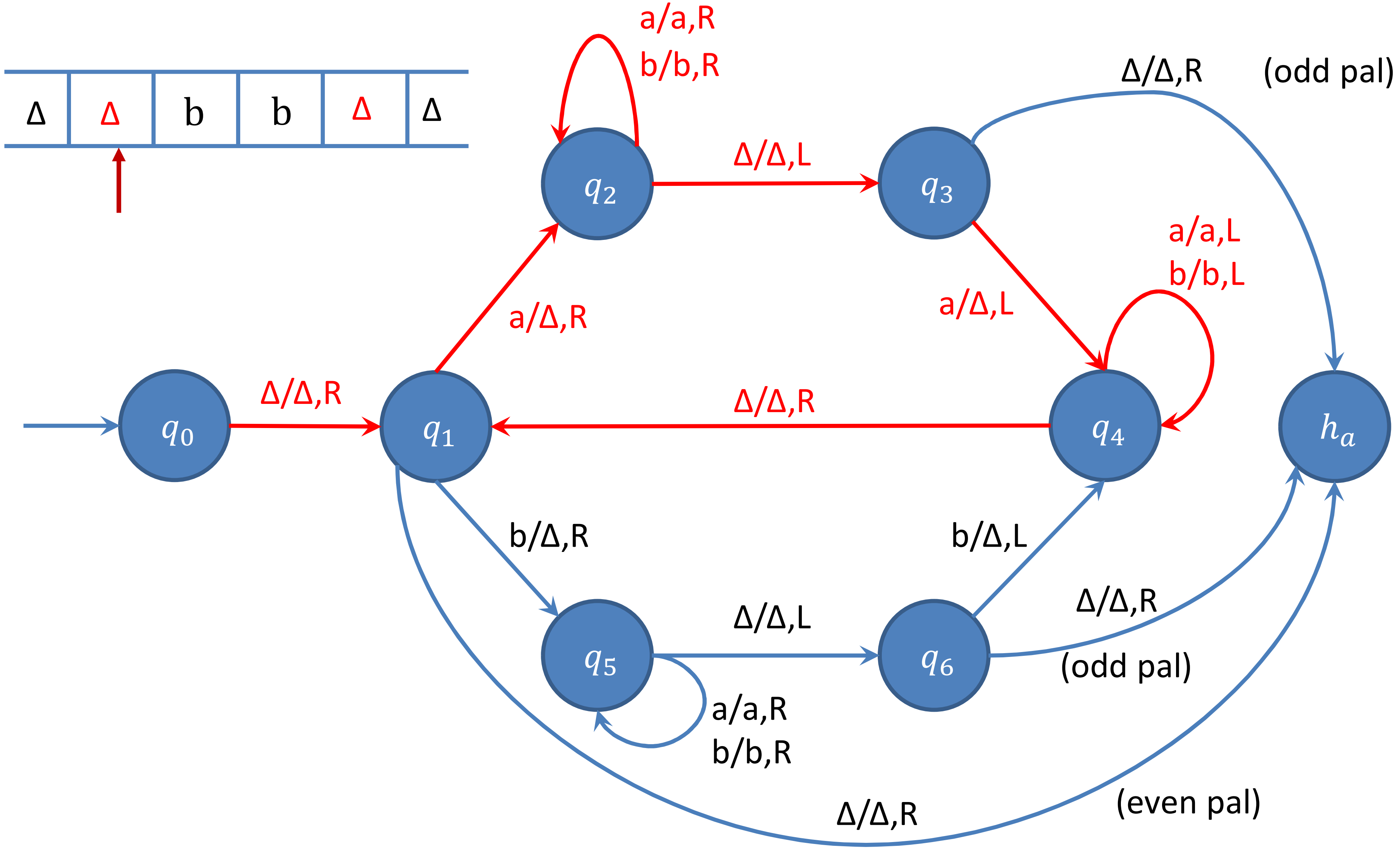
Design a TM for accepting Palindrome strings of even & odd length.



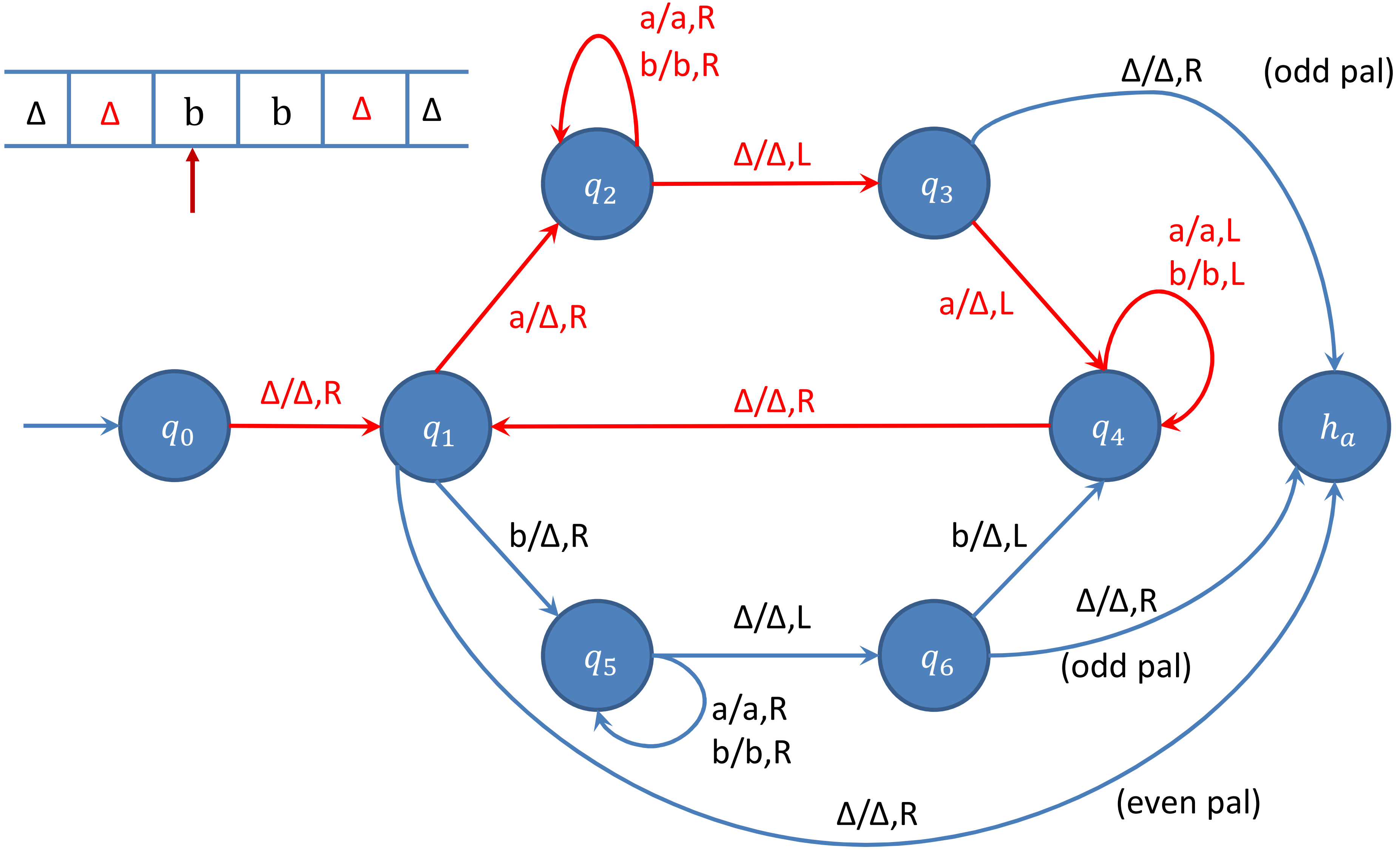
Design a TM for accepting Palindrome strings of even & odd length.



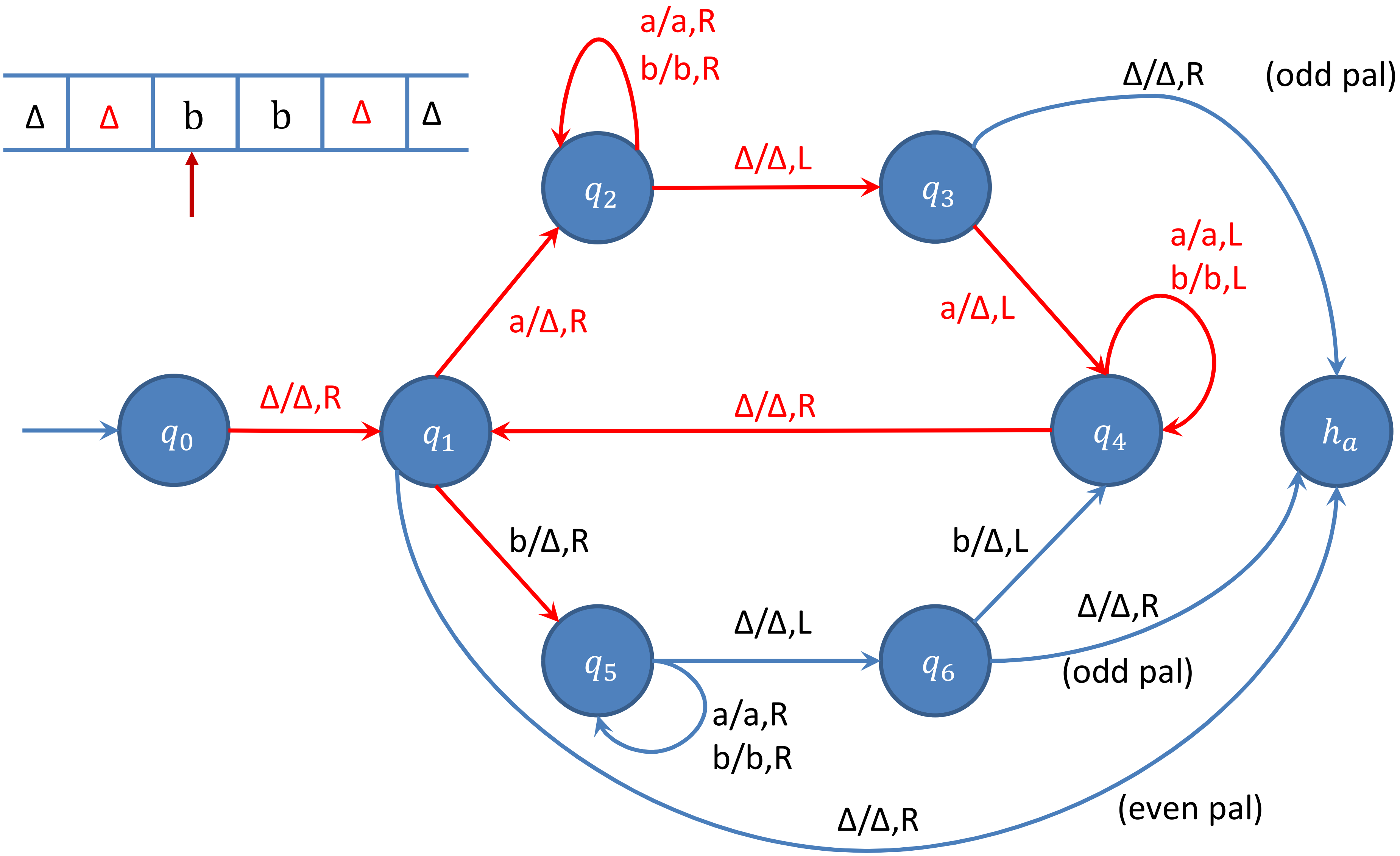
Design a TM for accepting Palindrome strings of even & odd length.



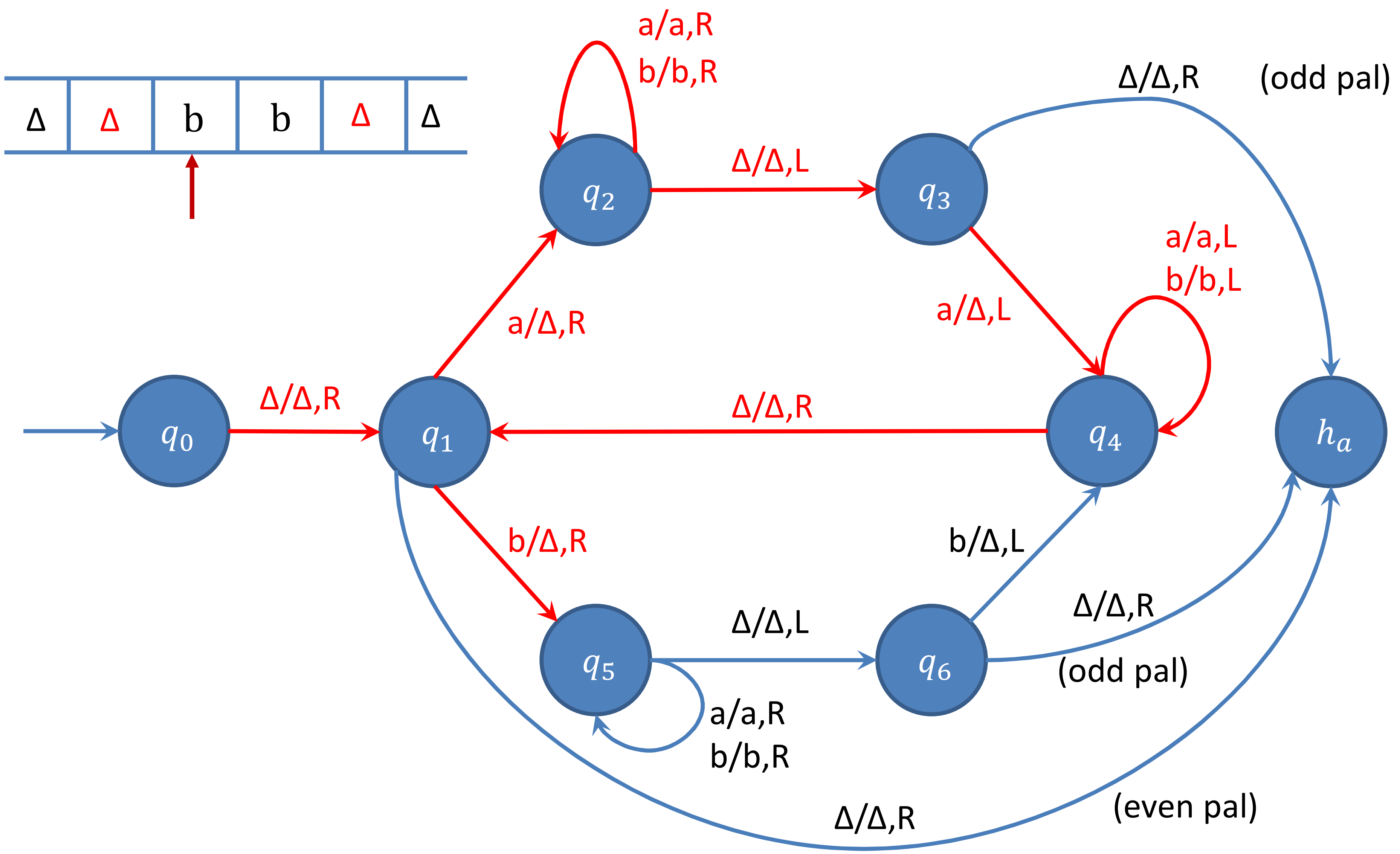
Design a TM for accepting Palindrome strings of even & odd length.



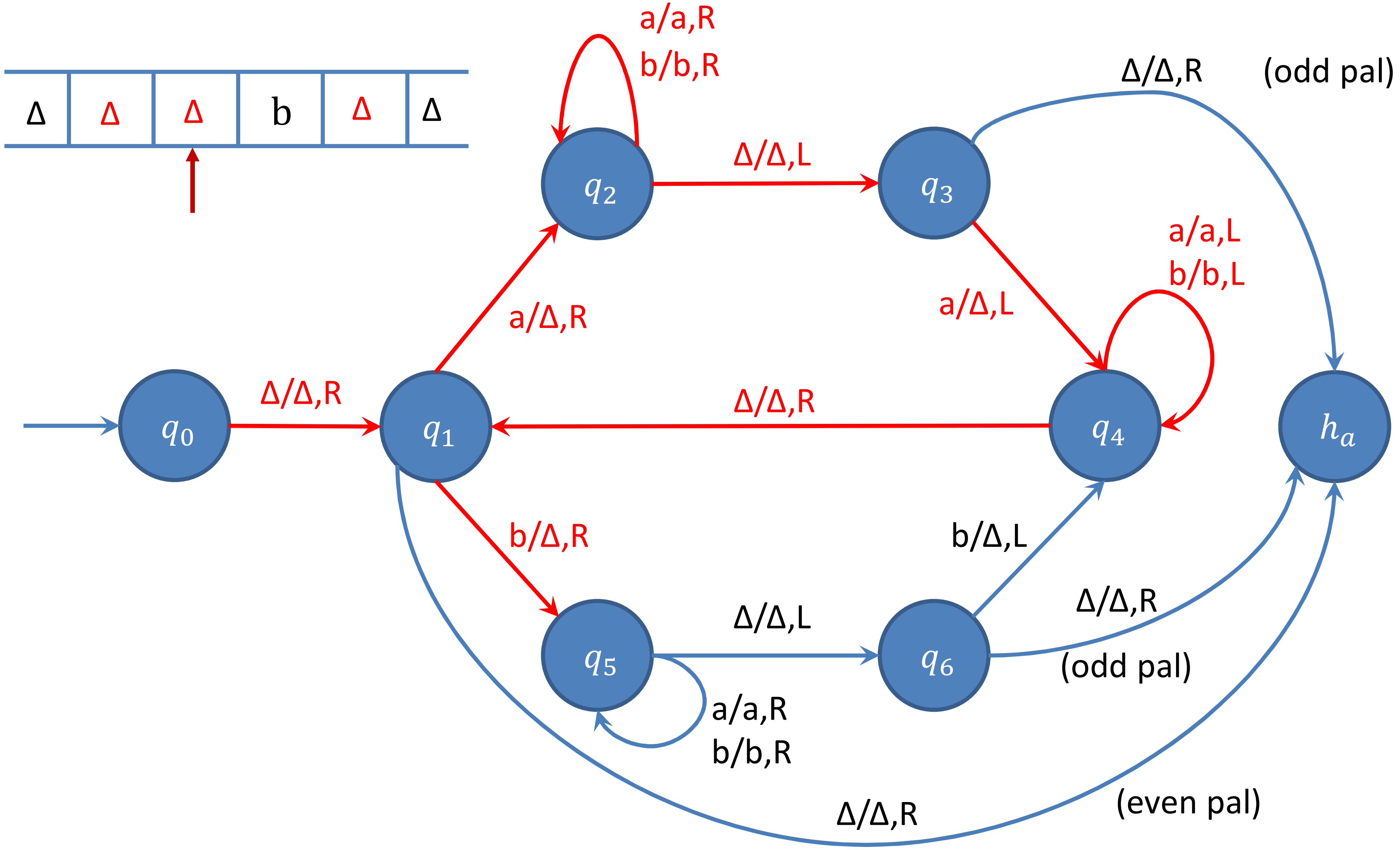
Design a TM for accepting Palindrome strings of even & odd length.



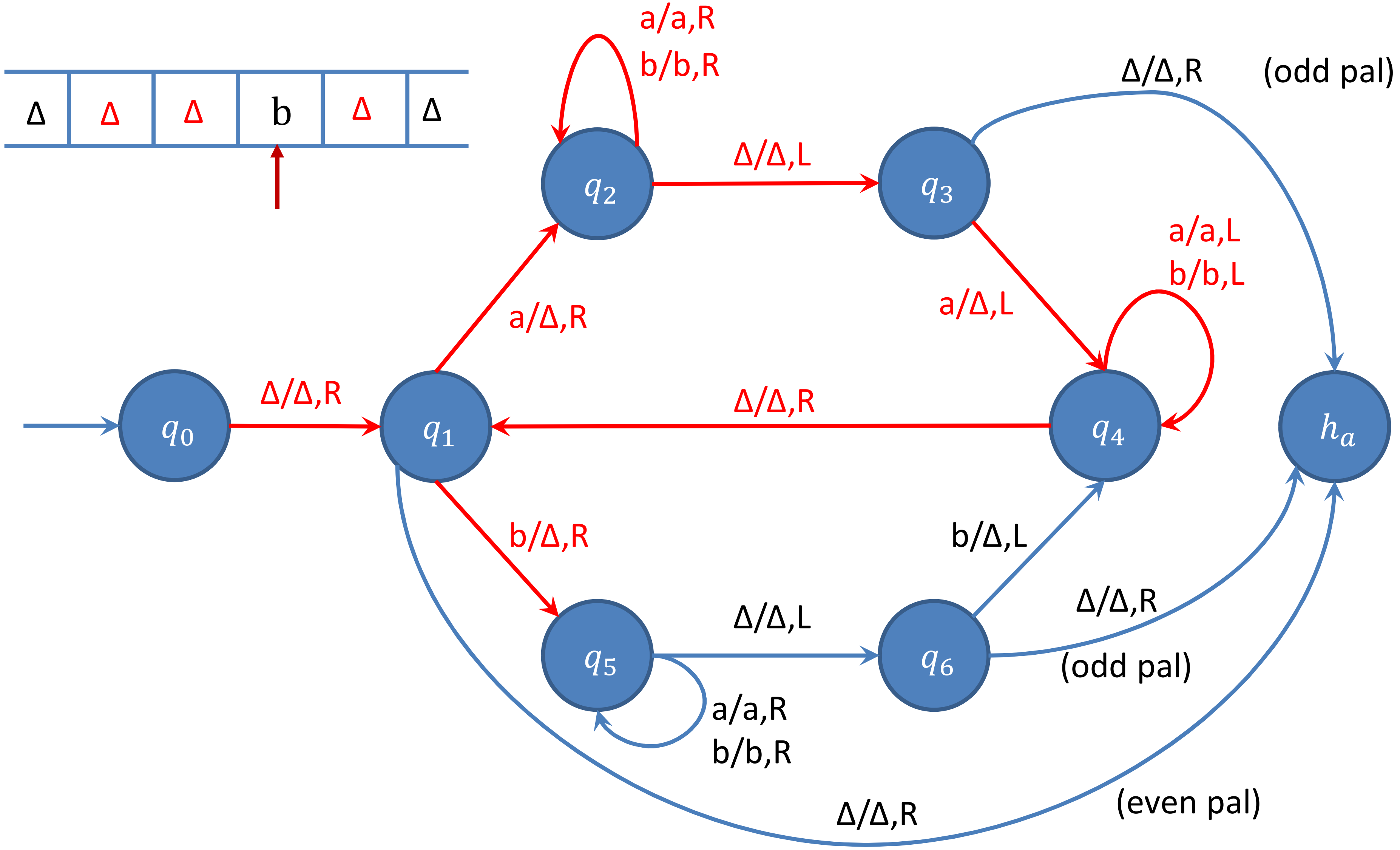
Design a TM for accepting Palindrome strings of even & odd length.



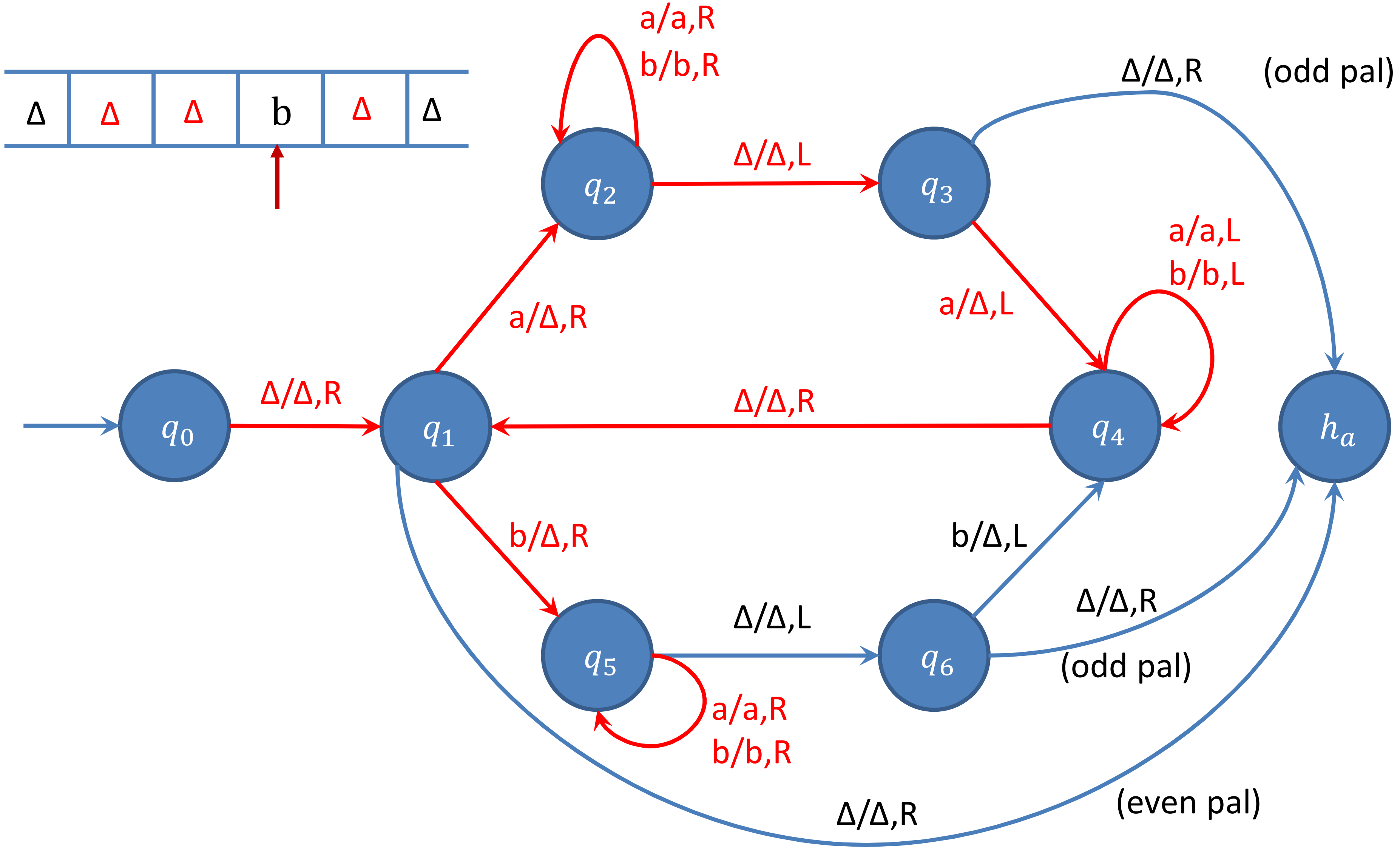
Design a TM for accepting Palindrome strings of even & odd length.



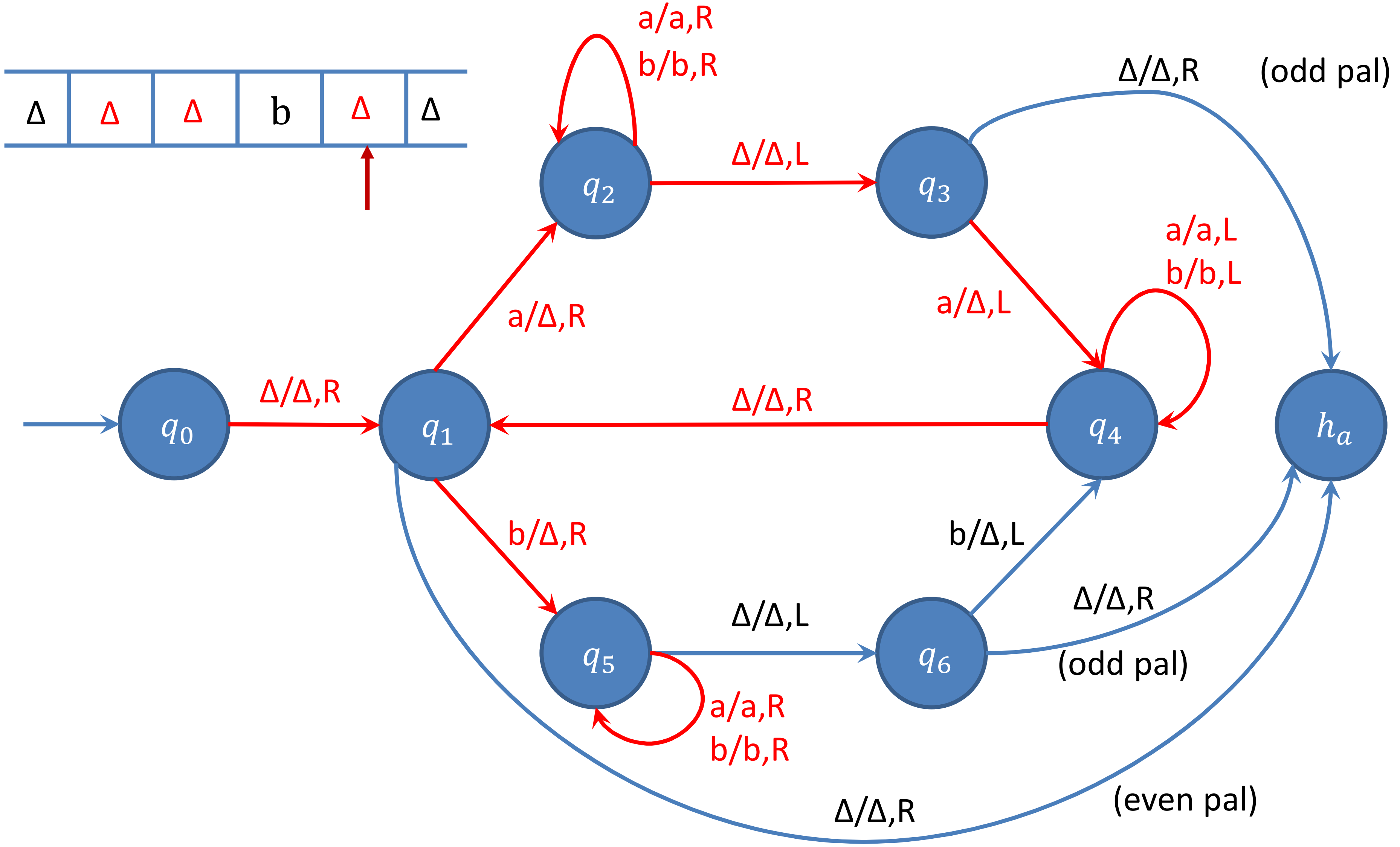
Design a TM for accepting Palindrome strings of even & odd length.



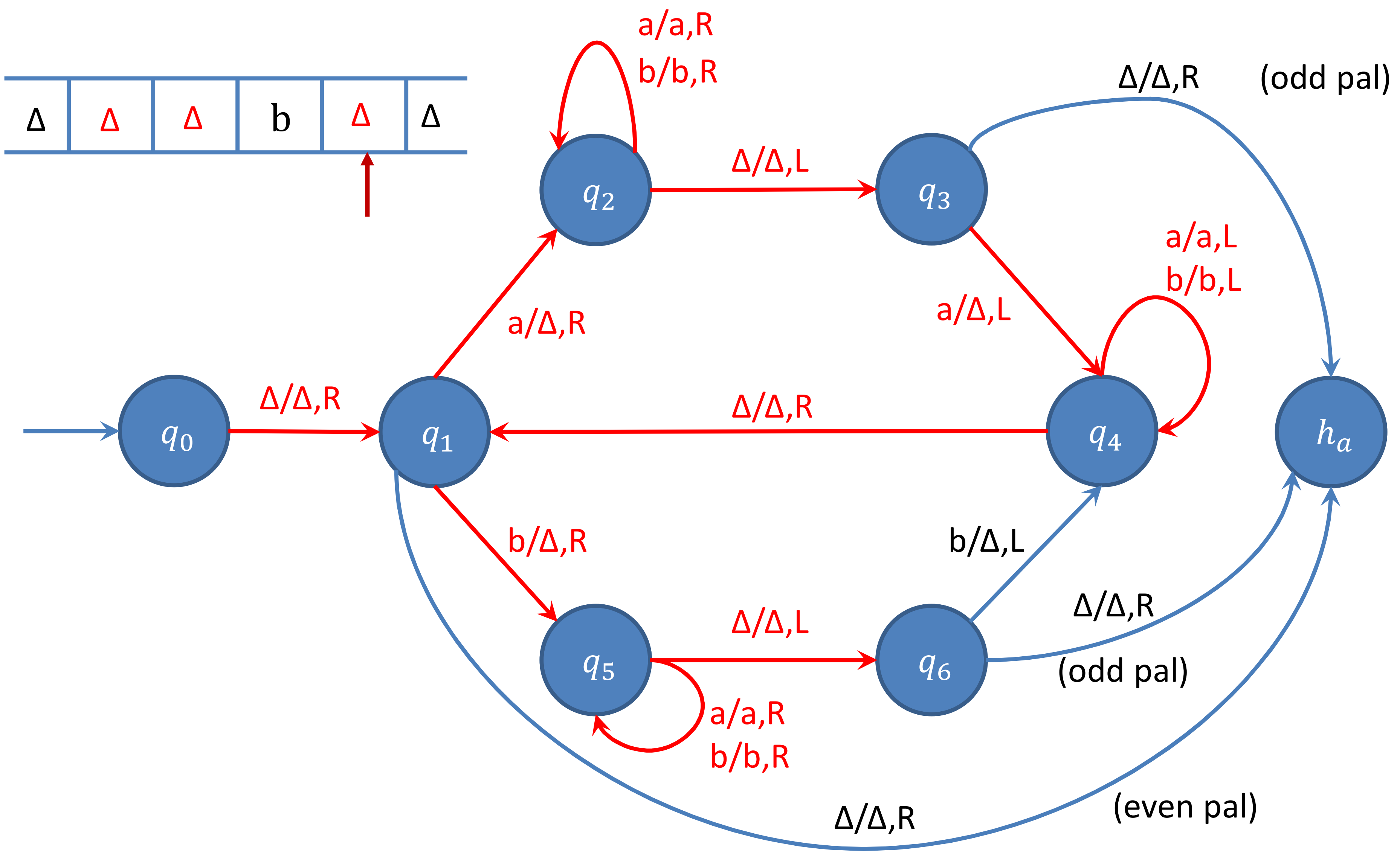
Design a TM for accepting Palindrome strings of even & odd length.



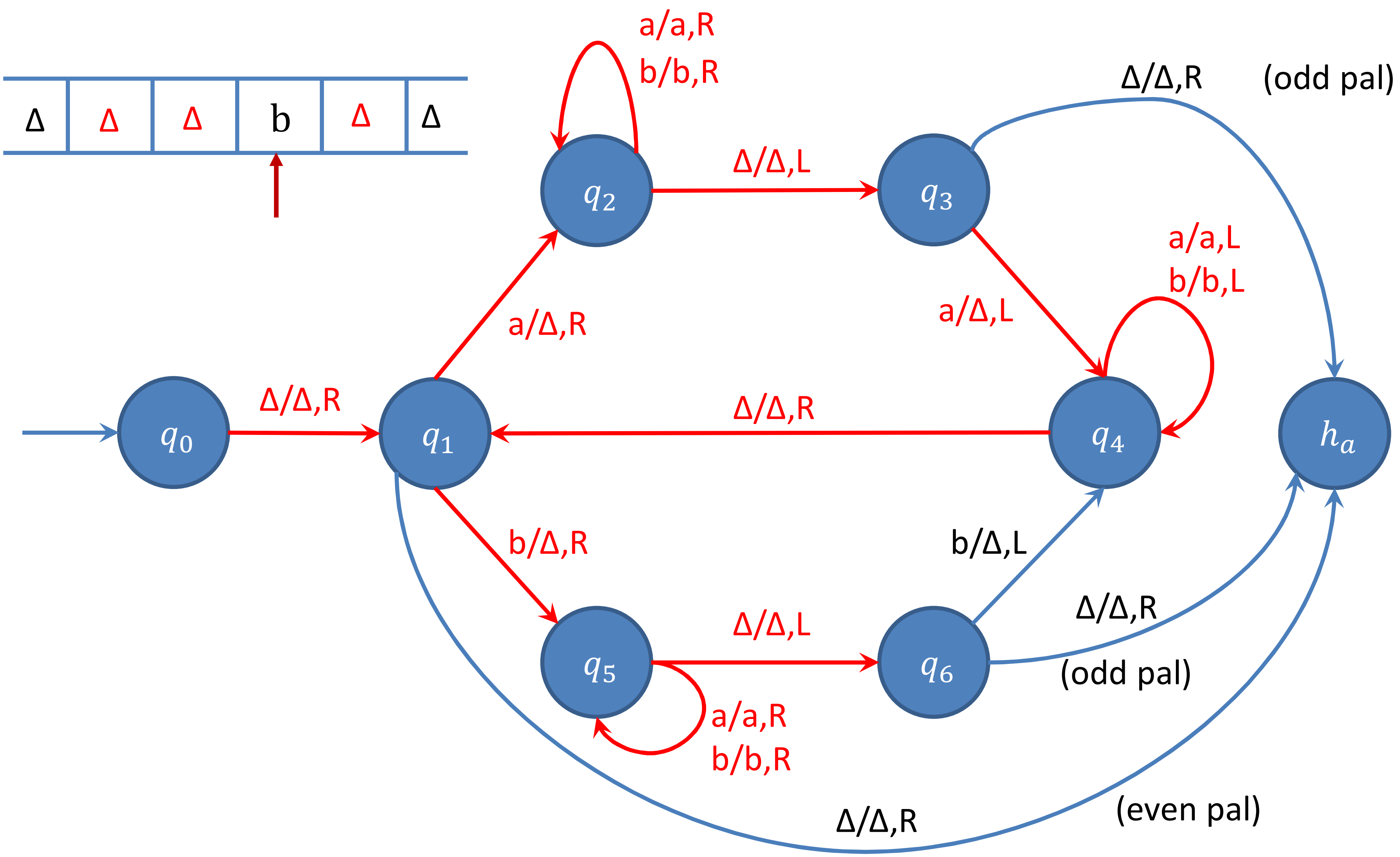
Design a TM for accepting Palindrome strings of even & odd length.



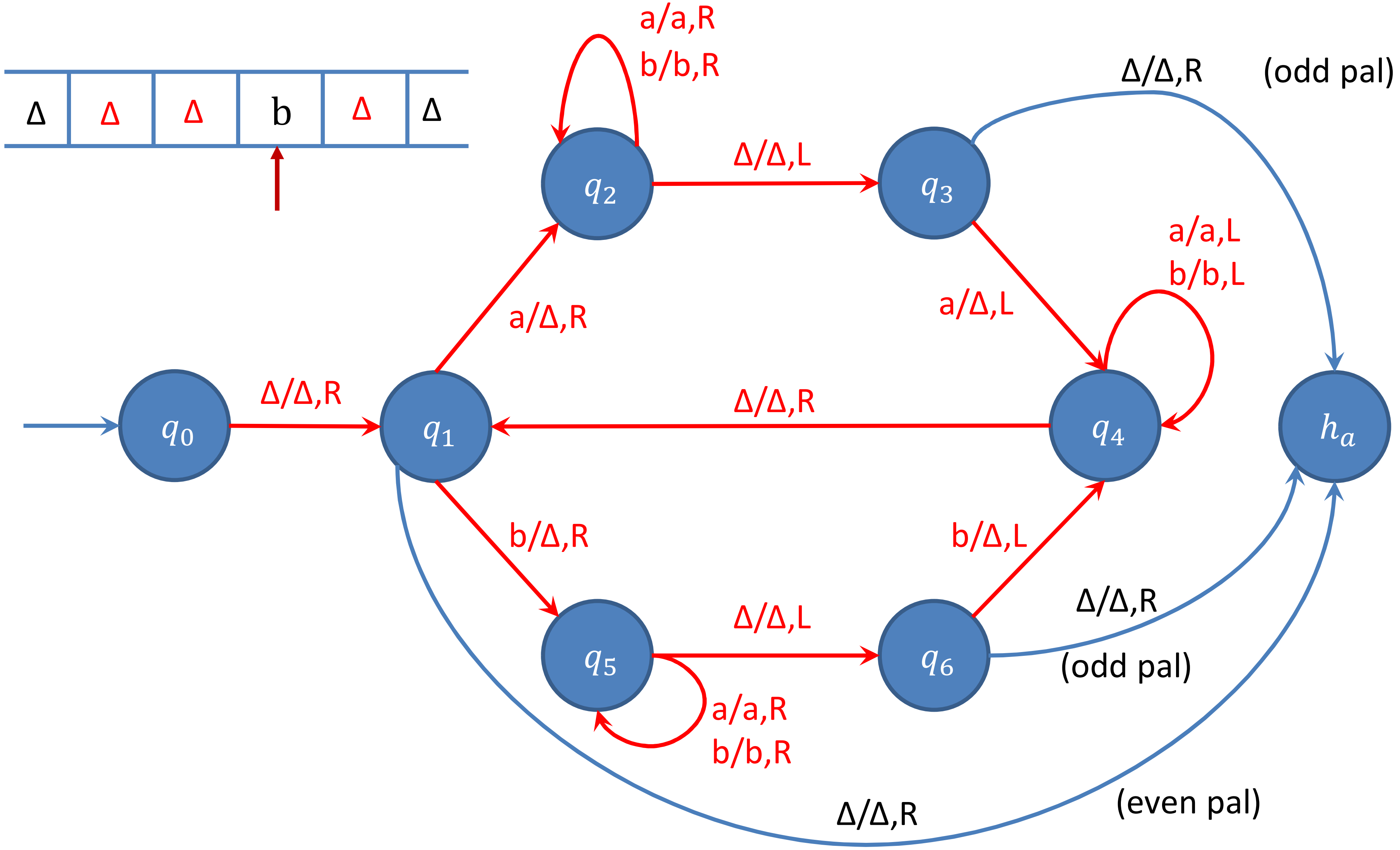
Design a TM for accepting Palindrome strings of even & odd length.



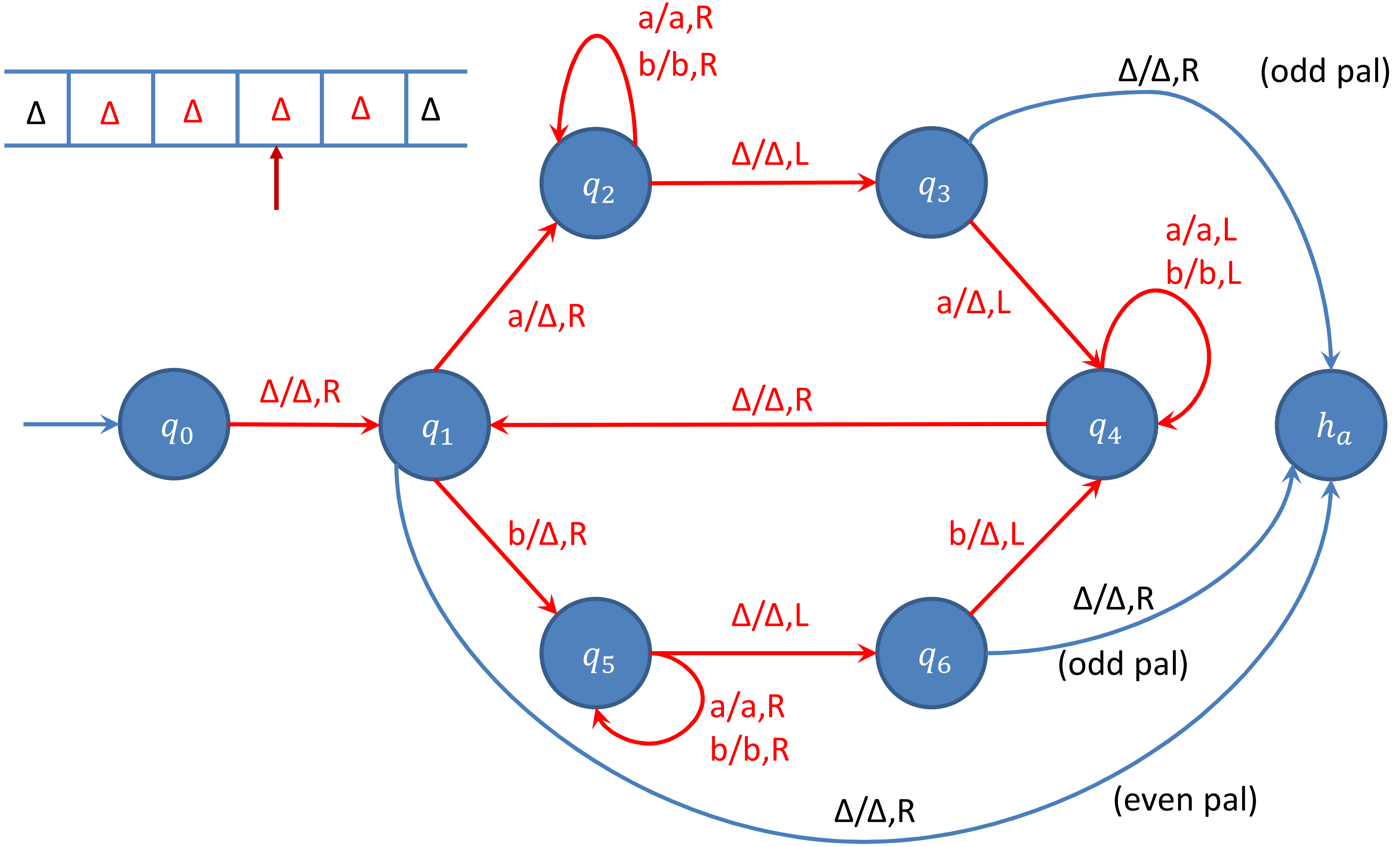
Design a TM for accepting Palindrome strings of even & odd length.



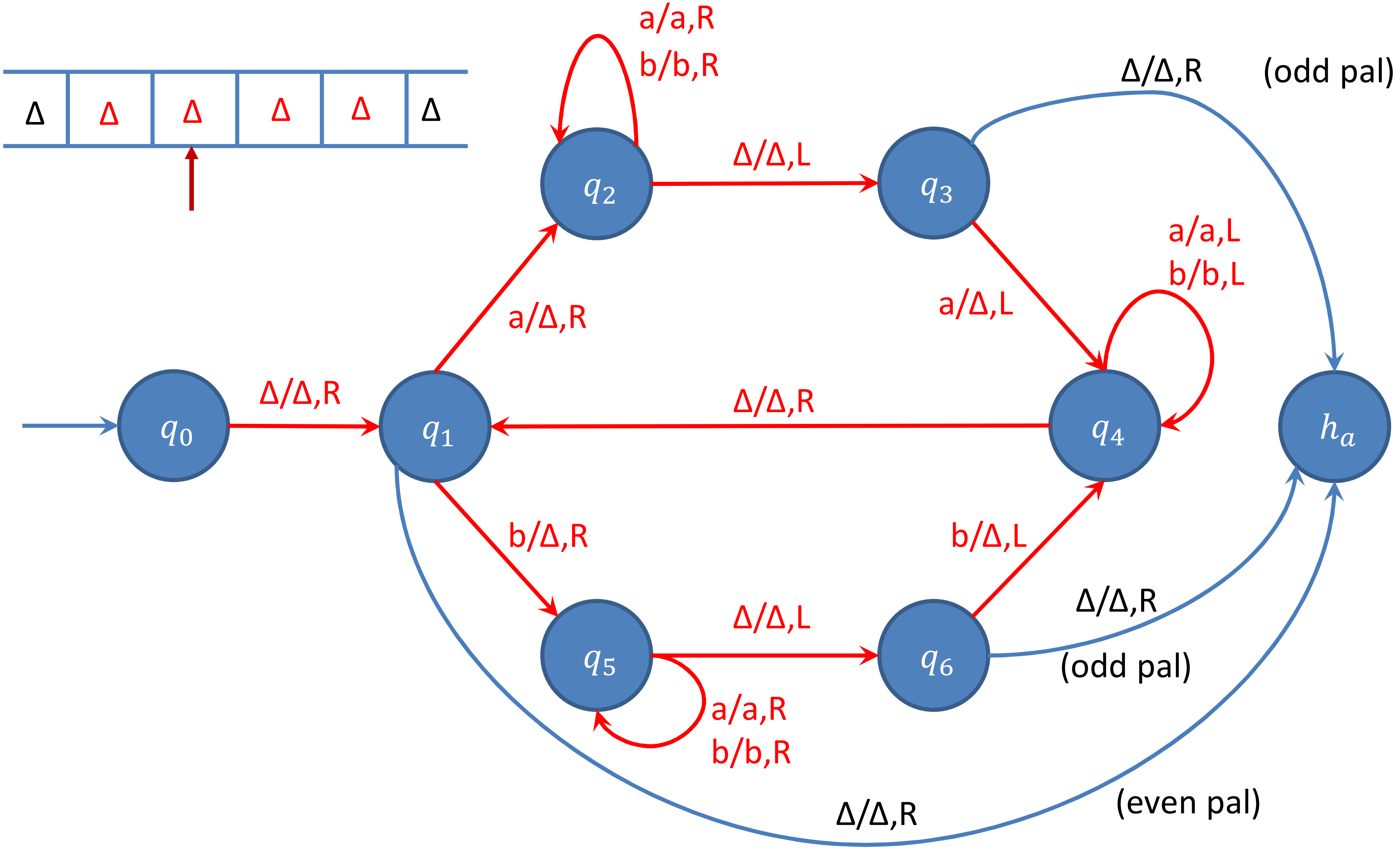
Design a TM for accepting Palindrome strings of even & odd length.



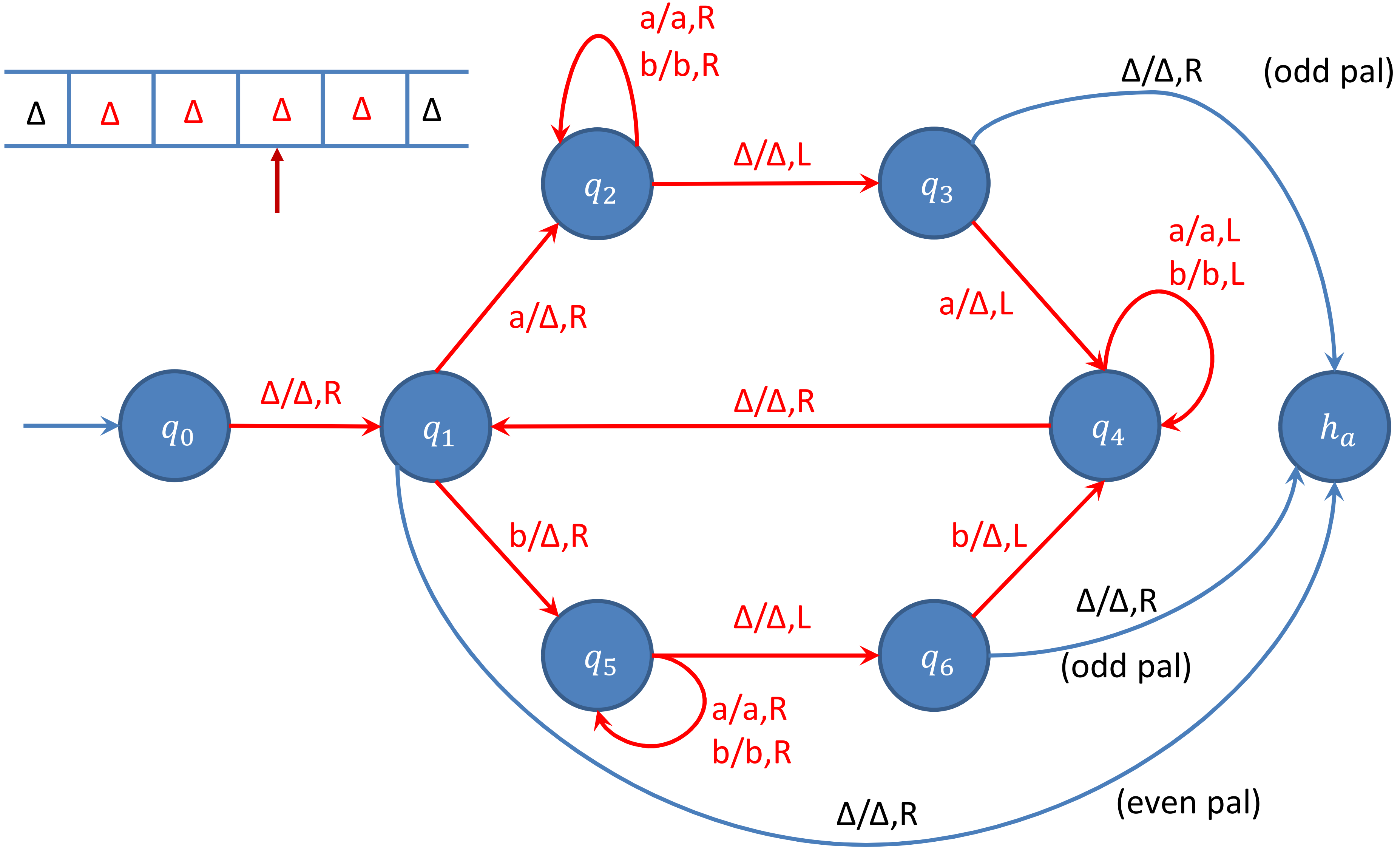
Design a TM for accepting Palindrome strings of even & odd length.



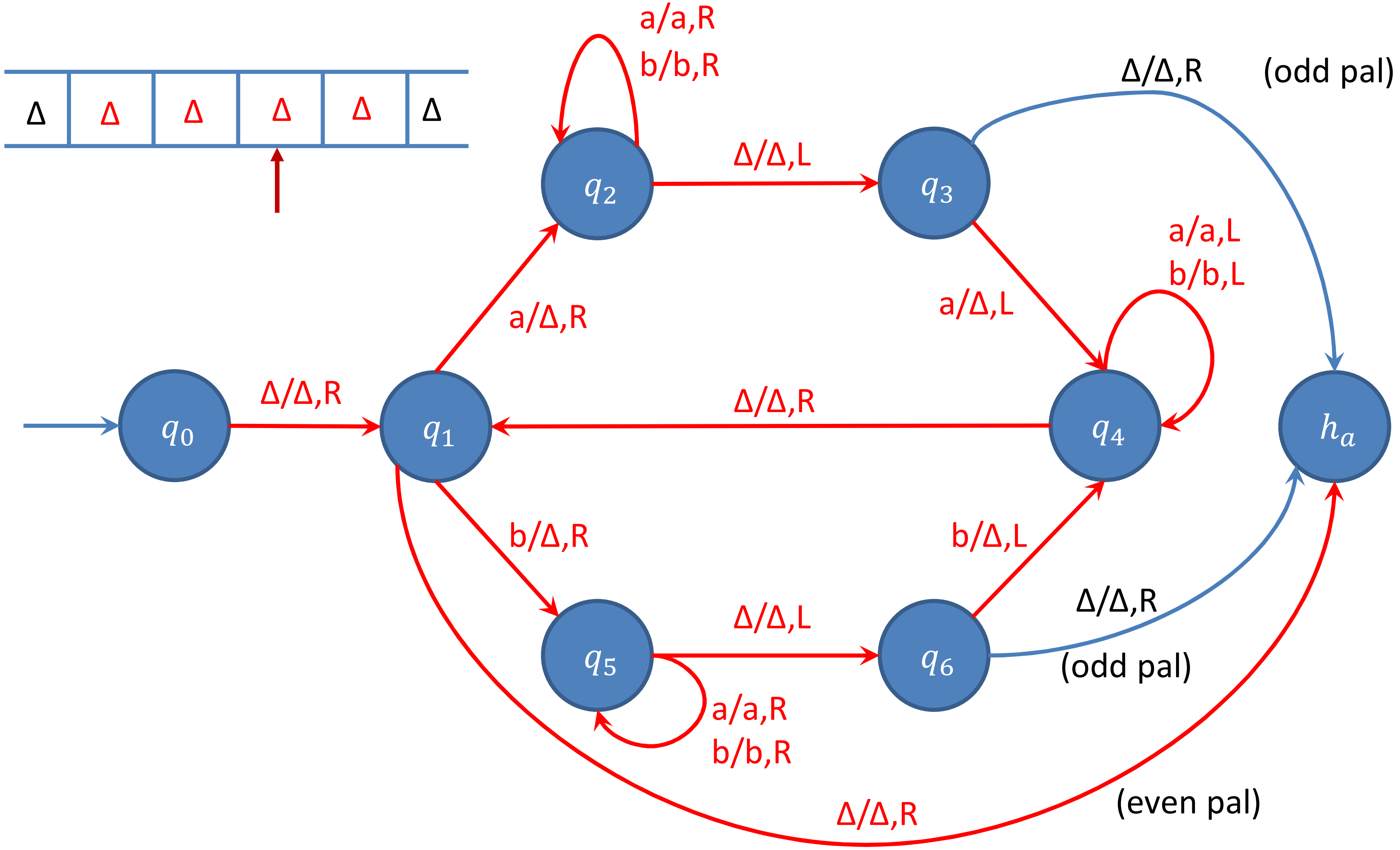
Design a TM for accepting Palindrome strings of even & odd length.



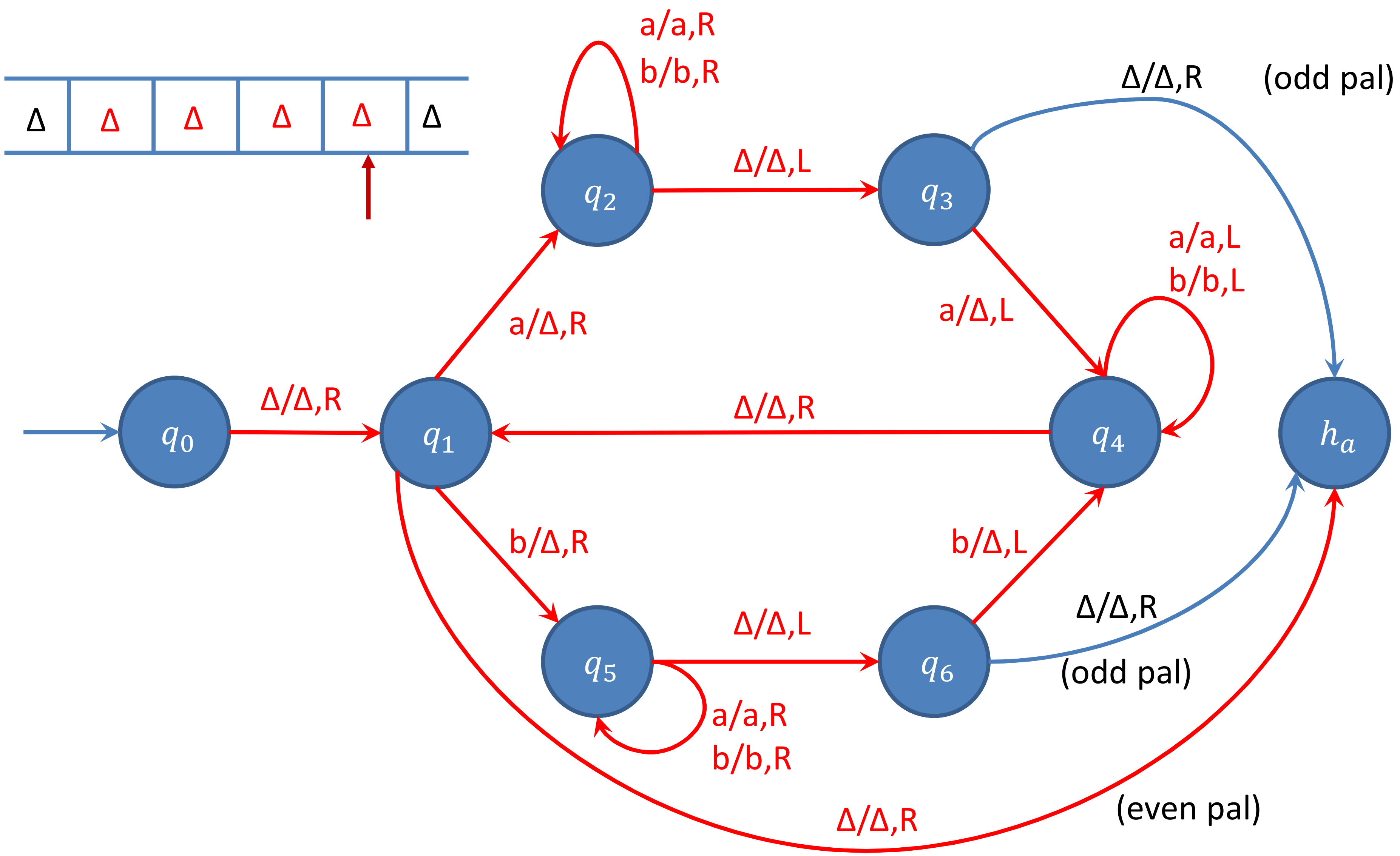
Design a TM for accepting Palindrome strings of even & odd length.



Design a TM for accepting Palindrome strings of even & odd length.



Design a TM for accepting Palindrome strings of even & odd length.



TM Accepting same no of 0’s and 1’s

Design a TM for accepting same no of 0’s and 1’s

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

01

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

01

10

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

01

10

0101

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

01

10

0101

1100

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

01

10

0101

1100

100011

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

01

10

0101

1100

100011

Δ	1	0	0	1	Δ

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

Δ	1	0	0	1	Δ

01

10

0101

1100

100011

Logic:

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

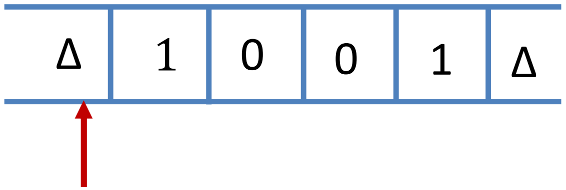
01

10

0101

1100

100011



Logic:

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

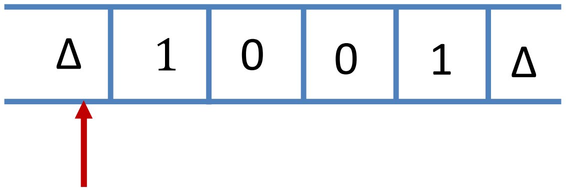
01

10

0101

1100

100011



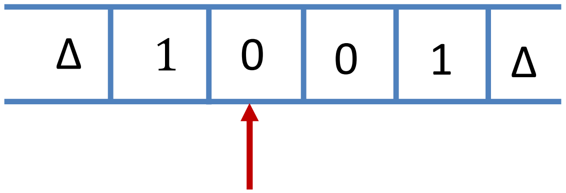
Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:



01

10

0101

1100

100011

Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

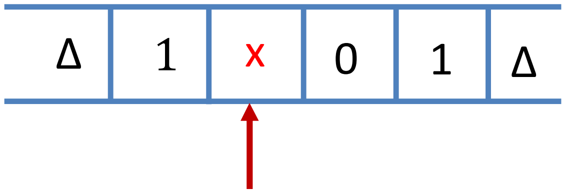
01

10

0101

1100

100011



Δ	1	x	0	1	Δ

Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

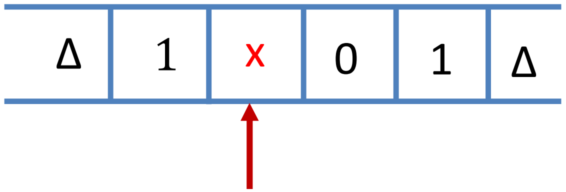
01

10

0101

1100

100011



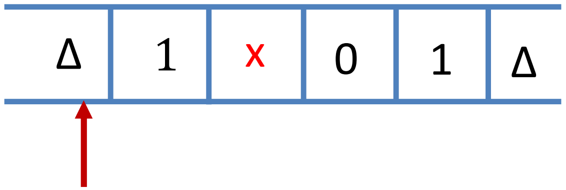
Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:



Δ	1	x	0	1	Δ

01

10

0101

1100

100011

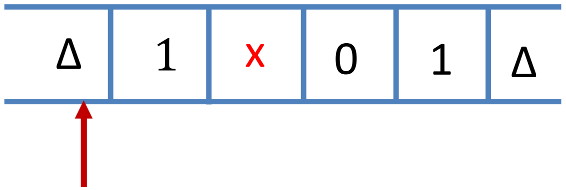
Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:



01

10

0101

1100

100011

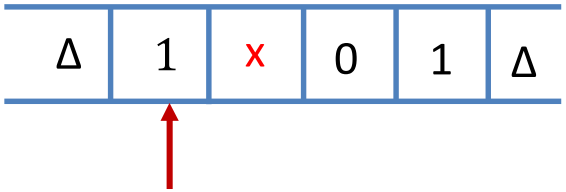
Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:



Δ	1	x	0	1	Δ

01

10

0101

1100

100011

Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

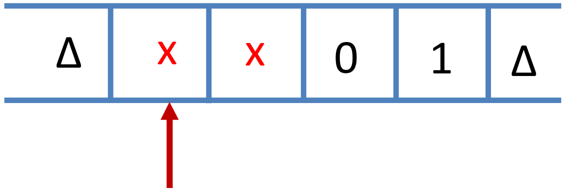
01

10

0101

1100

100011



Δ	x	x	0	1	Δ

Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

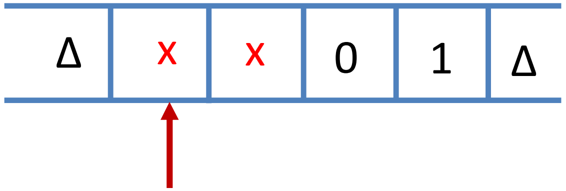
01

10

0101

1100

100011



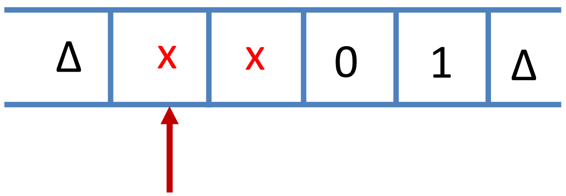
Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:



01

10

0101

1100

100011

Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ
5. Repeat the above steps until no more 0’s and 1’s left in string.

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

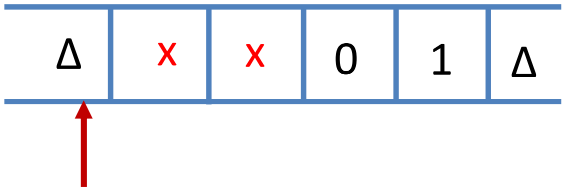
01

10

0101

1100

100011



Δ	x	x	0	1	Δ

Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ
5. Repeat the above steps until no more 0’s and 1’s left in string.

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

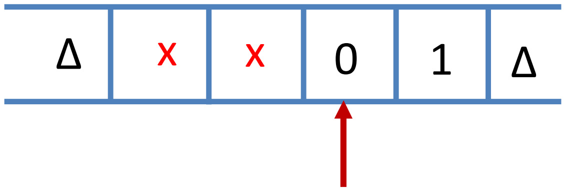
01

10

0101

1100

100011



Δ	x	x	0	1	Δ

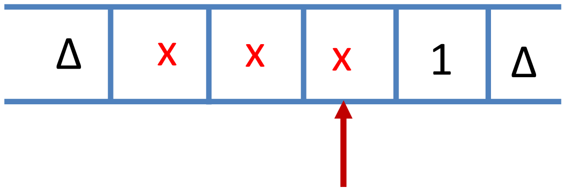
Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ
5. Repeat the above steps until no more 0’s and 1’s left in string.

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:



Δ	x	x	x	1	Δ

01

10

0101

1100

100011

Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ
5. Repeat the above steps until no more 0’s and 1’s left in string.

Design a TM for accepting same no of 0's and 1's

L = Same no. of 0's and 1's

Strings accepting in given L are:

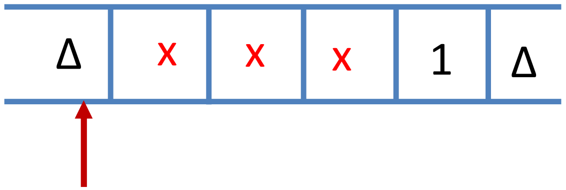
01

10

0101

1100

100011



Δ	x	x	x	1	Δ

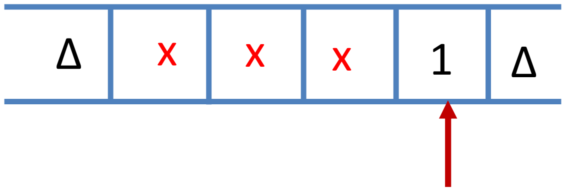
Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ
5. Repeat the above steps until no more 0's and 1's left in string.

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:



01

10

0101

1100

100011

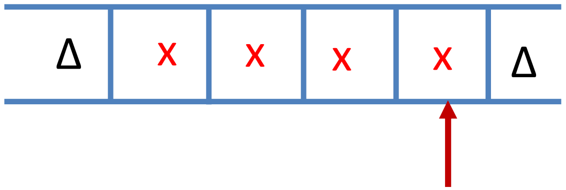
Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ
5. Repeat the above steps until no more 0’s and 1’s left in string.

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:



01

10

0101

1100

100011

Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ
5. Repeat the above steps until no more 0’s and 1’s left in string.

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

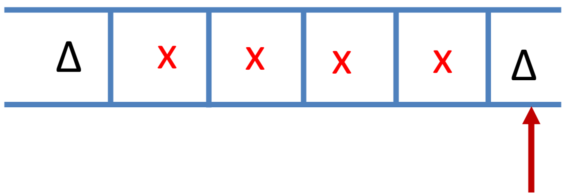
01

10

0101

1100

100011



Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ
5. Repeat the above steps until no more 0’s and 1’s left in string.

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

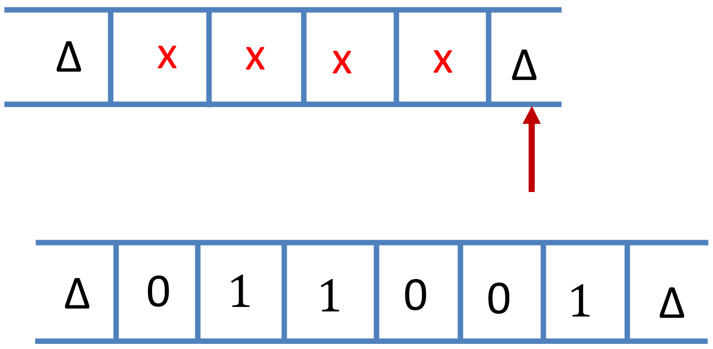
01

10

0101

1100

100011



Δ	x	x	x	x	Δ

Δ	0	1	1	0	0	1	Δ

Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ
5. Repeat the above steps until no more 0’s and 1’s left in string.

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

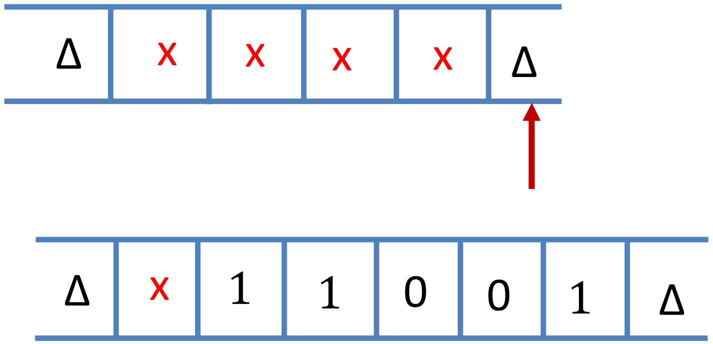
01

10

0101

1100

100011



Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ
5. Repeat the above steps until no more 0’s and 1’s left in string.

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

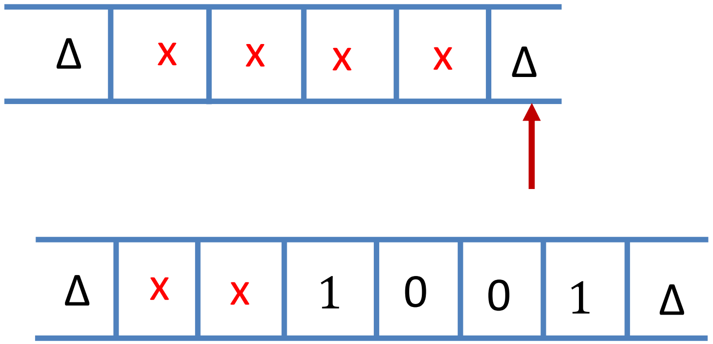
01

10

0101

1100

100011



Δ	x	x	x	x	Δ

Δ	x	x	1	0	0	1	Δ

Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ
5. Repeat the above steps until no more 0’s and 1’s left in string.

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

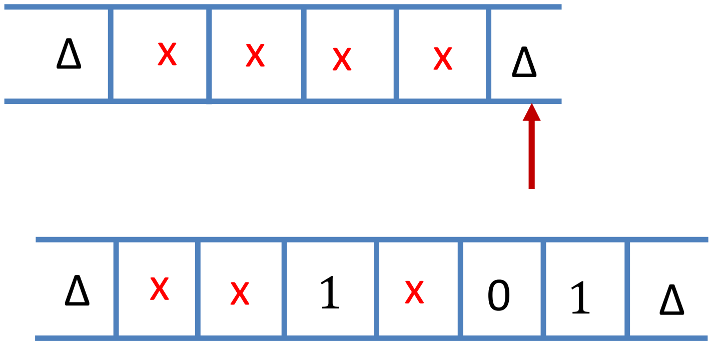
01

10

0101

1100

100011



Δ	x	x	x	x	Δ

Δ	x	x	1	x	0	1	Δ

Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ
5. Repeat the above steps until no more 0’s and 1’s left in string.

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

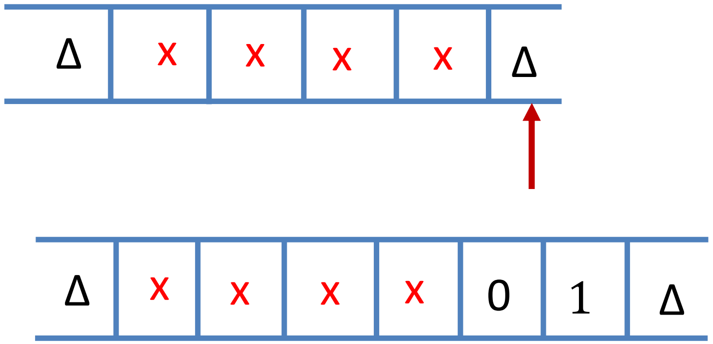
01

10

0101

1100

100011



Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ
5. Repeat the above steps until no more 0’s and 1’s left in string.

Design a TM for accepting same no of 0’s and 1’s

L = Same no. of 0’s and 1’s

Strings accepting in given L are:

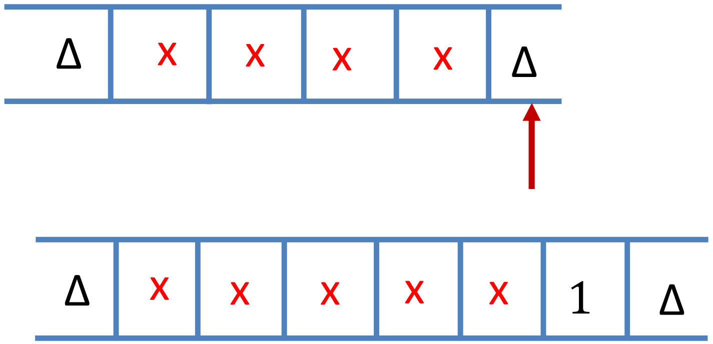
01

10

0101

1100

100011



Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ
5. Repeat the above steps until no more 0’s and 1’s left in string.

Design a TM for accepting same no of 0's and 1's

L = Same no. of 0's and 1's

Strings accepting in given L are:

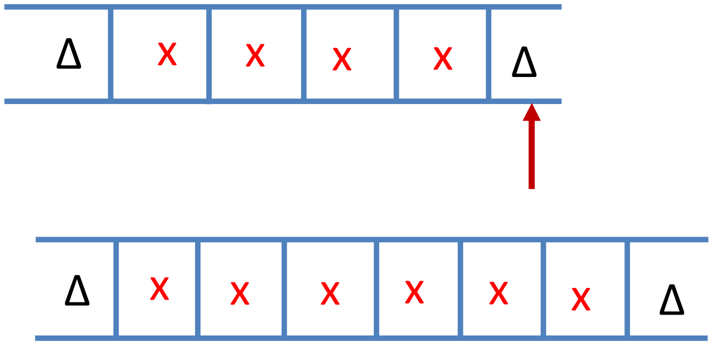
01

10

0101

1100

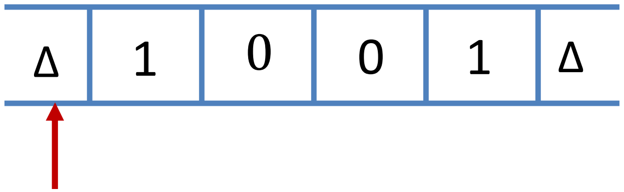
100011

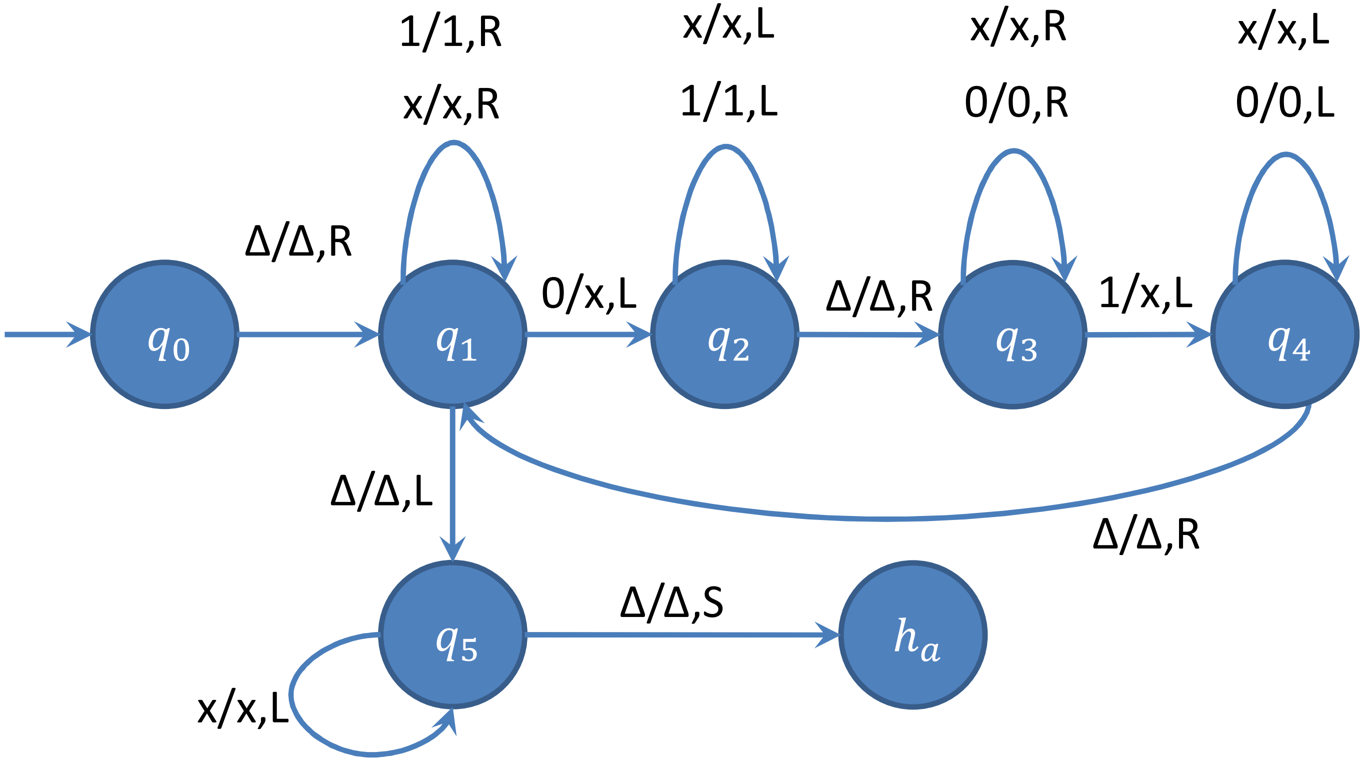


Logic:

1. Move right, Read first leftmost 0, replace 0 with X (0→X)
2. Keep moving to the left until you encounter Δ
3. Move right, Read first leftmost 1, replace 1 with X (1→X)
4. Keep moving to the left until you encounter Δ
5. Repeat the above steps until no more 0's and 1's left in string.

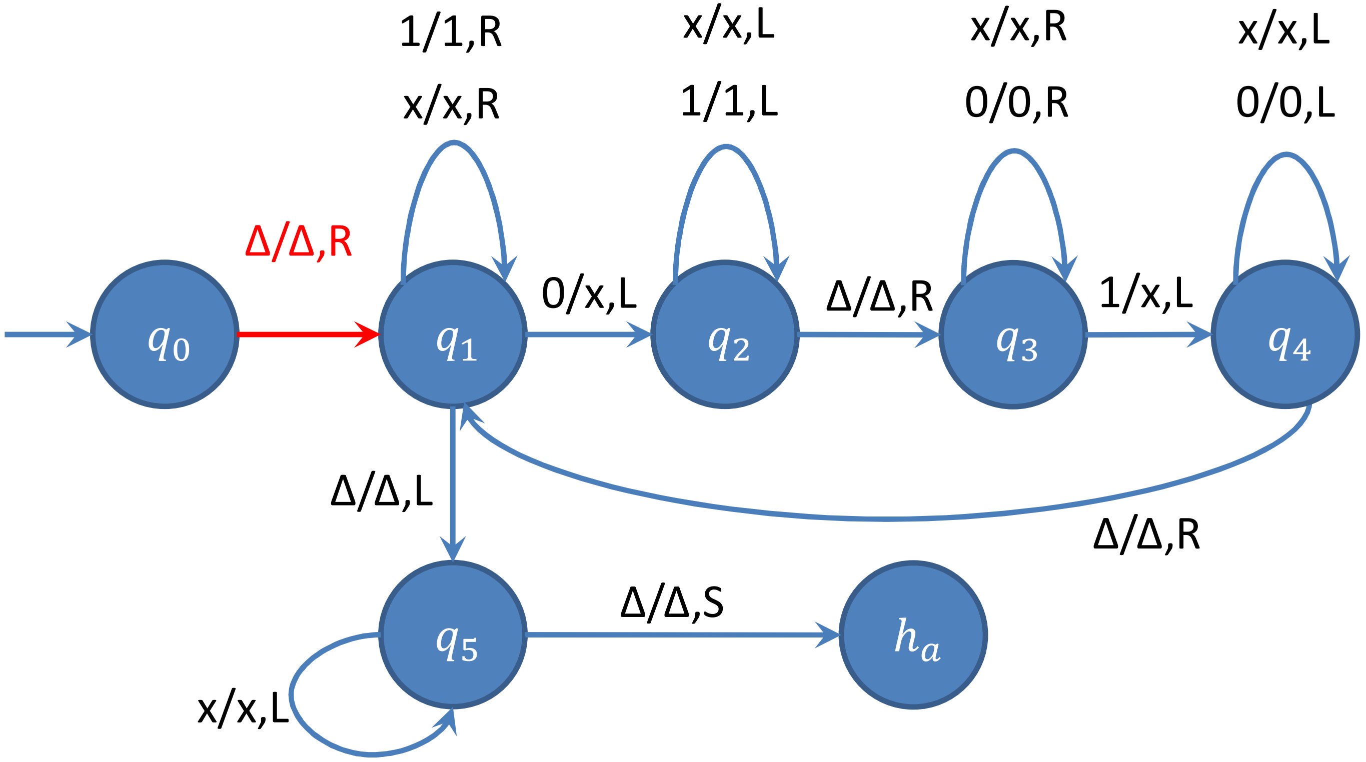
Design a TM for accepting same no of 0’s and 1’s



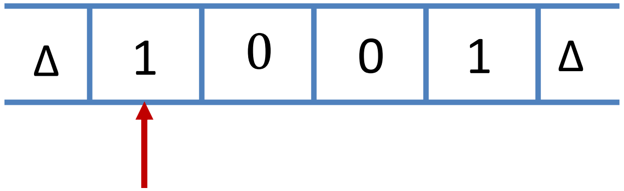


Design a TM for accepting same no of 0’s and 1’s

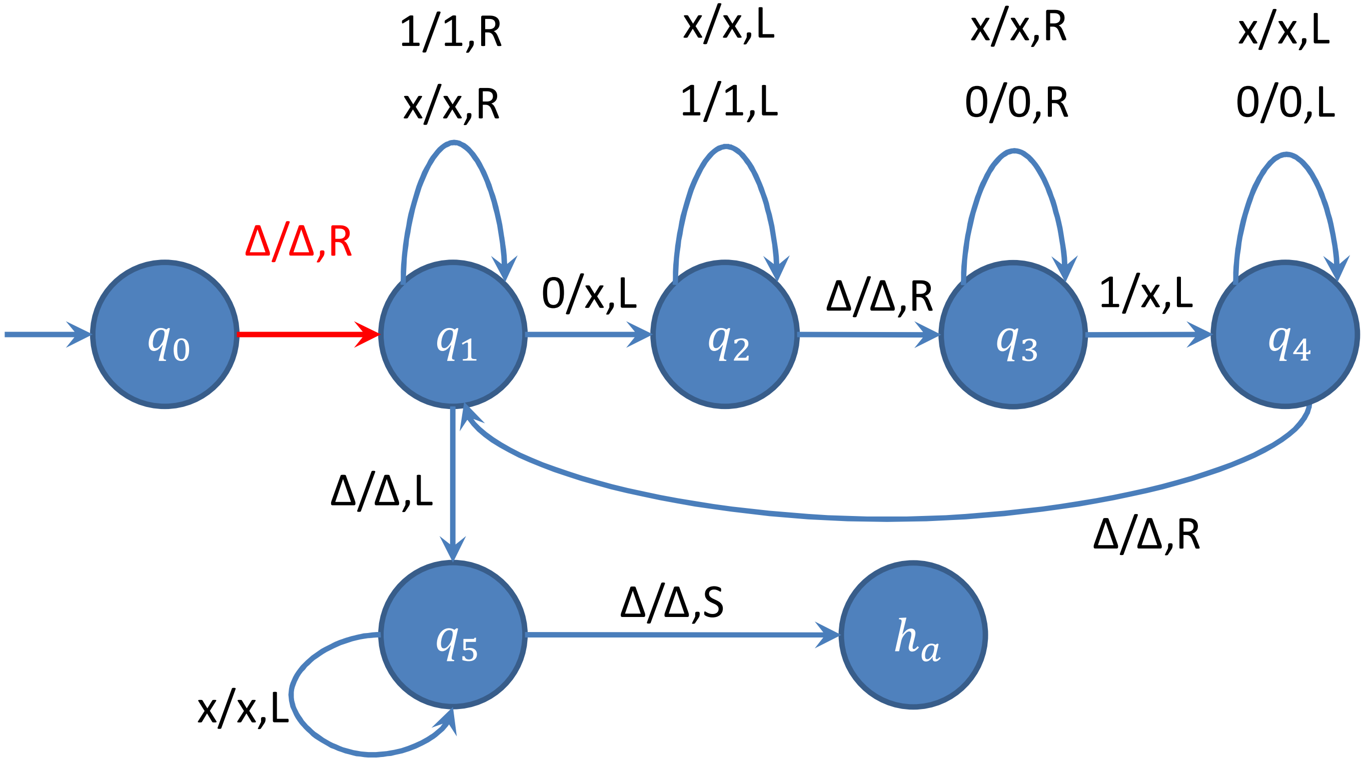
Δ	1	0	0	1	Δ



Design a TM for accepting same no of 0’s and 1’s

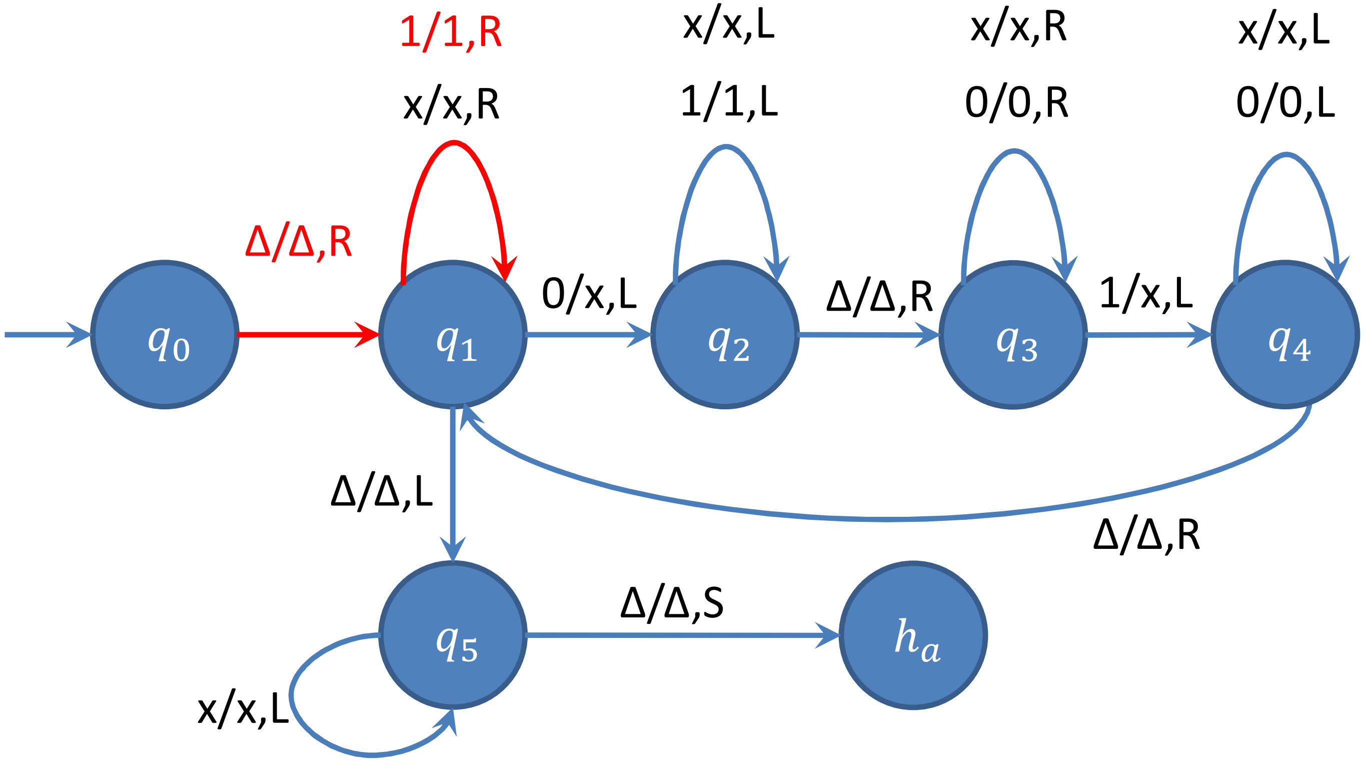


Δ	1	0	0	1	Δ

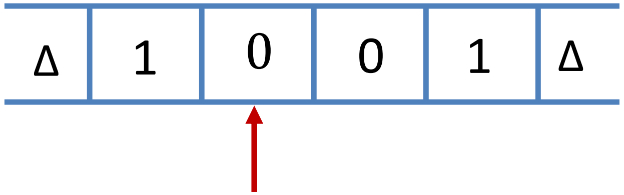


Design a TM for accepting same no of 0’s and 1’s

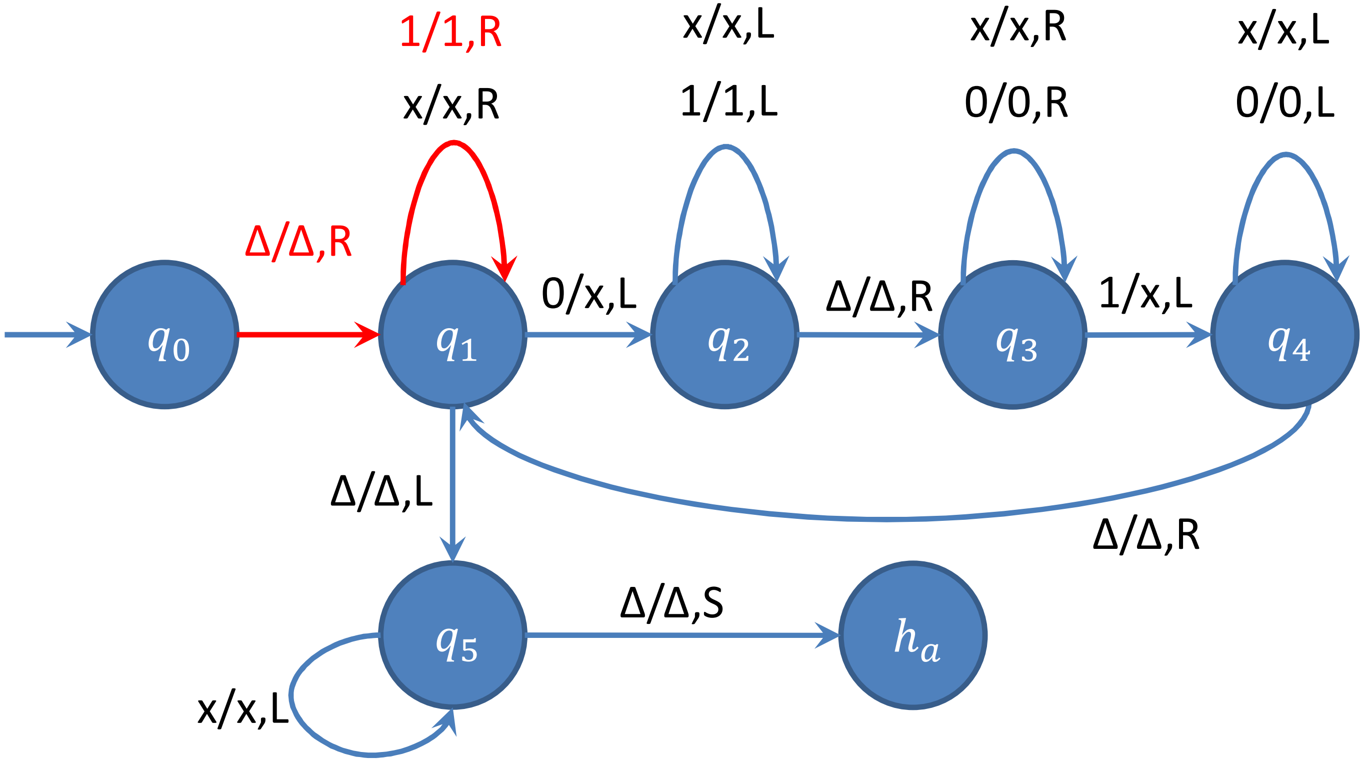
Δ	1	0	0	1	Δ



Design a TM for accepting same no of 0’s and 1’s

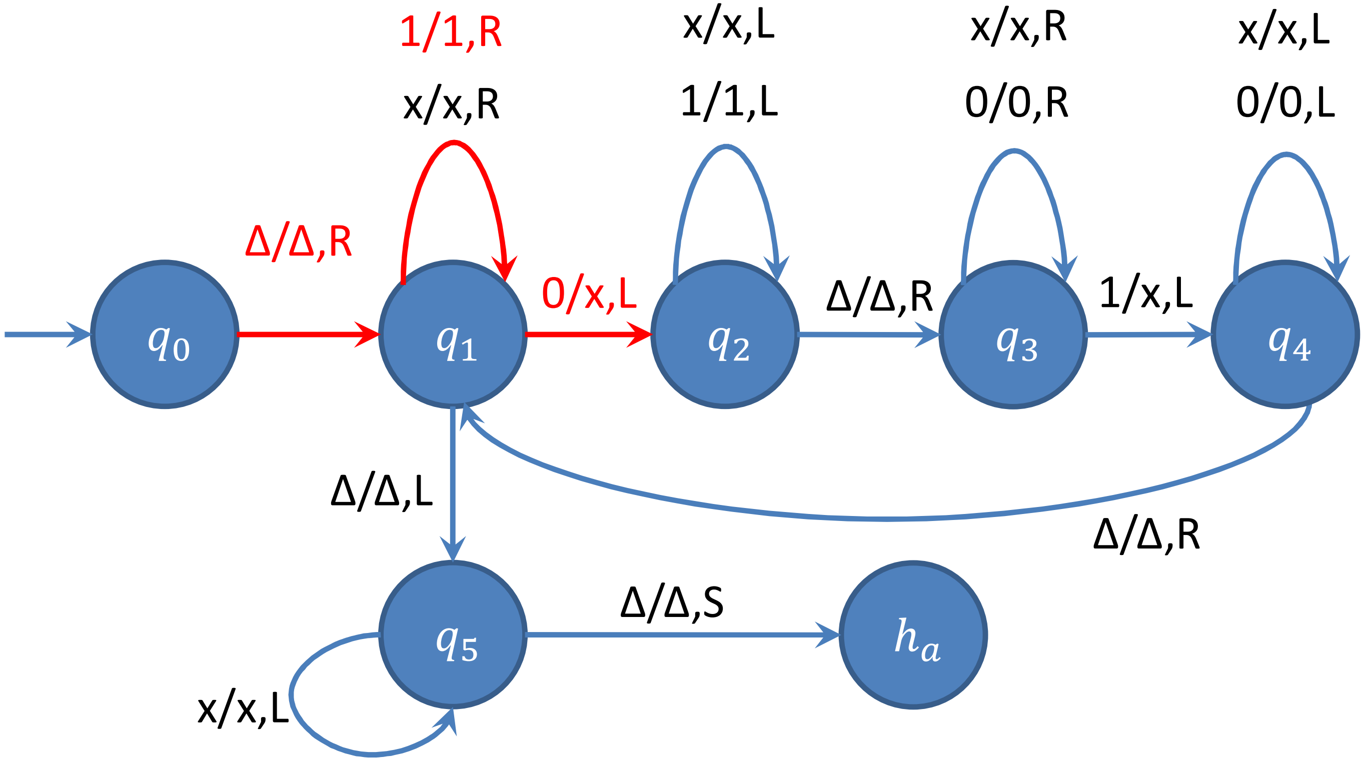


Δ	1	0	0	1	Δ



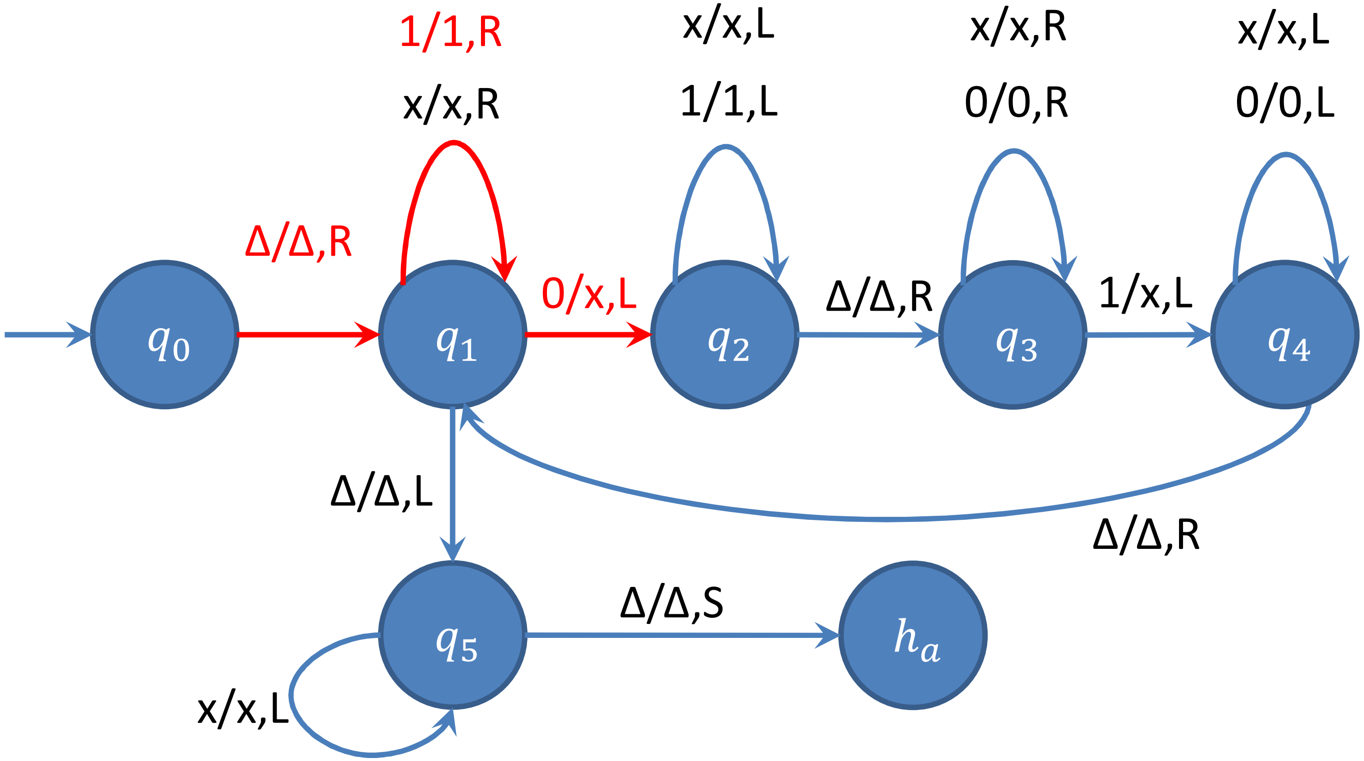
Design a TM for accepting same no of 0’s and 1’s

Δ	1	0	0	1	Δ

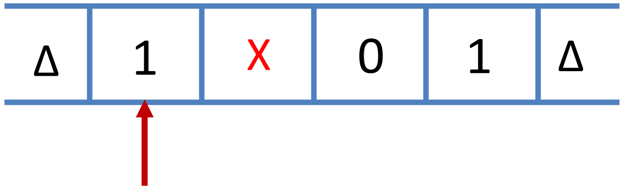


Design a TM for accepting same no of 0’s and 1’s

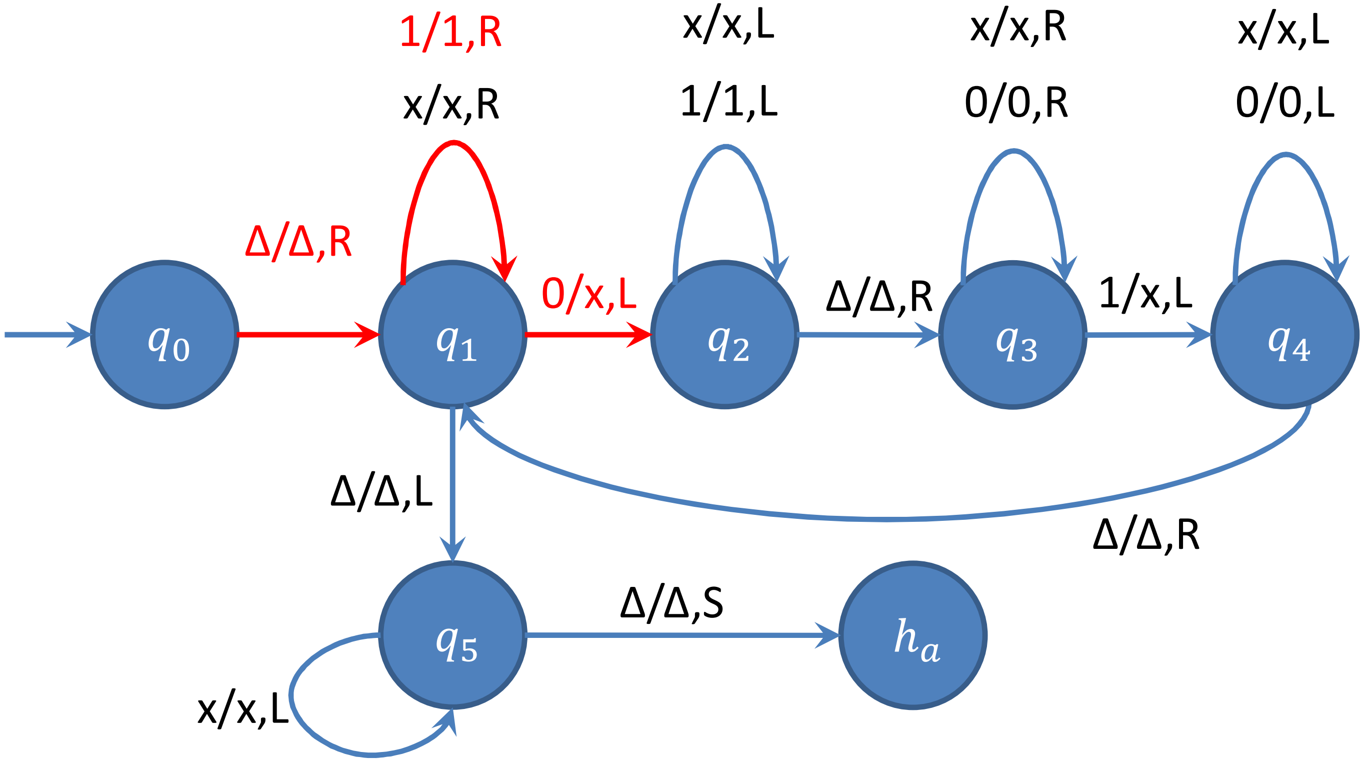
Δ	1	X	0	1	Δ



Design a TM for accepting same no of 0’s and 1’s

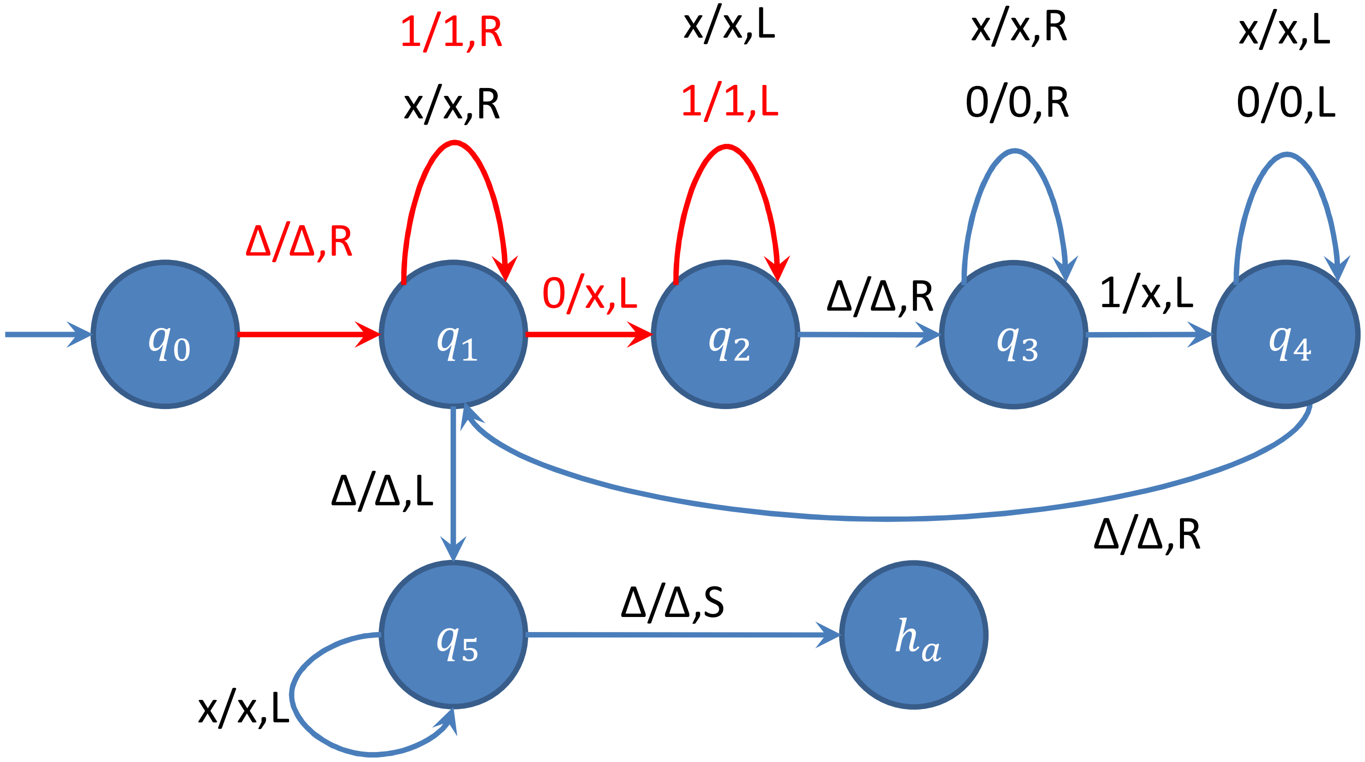


Δ	1	X	0	1	Δ



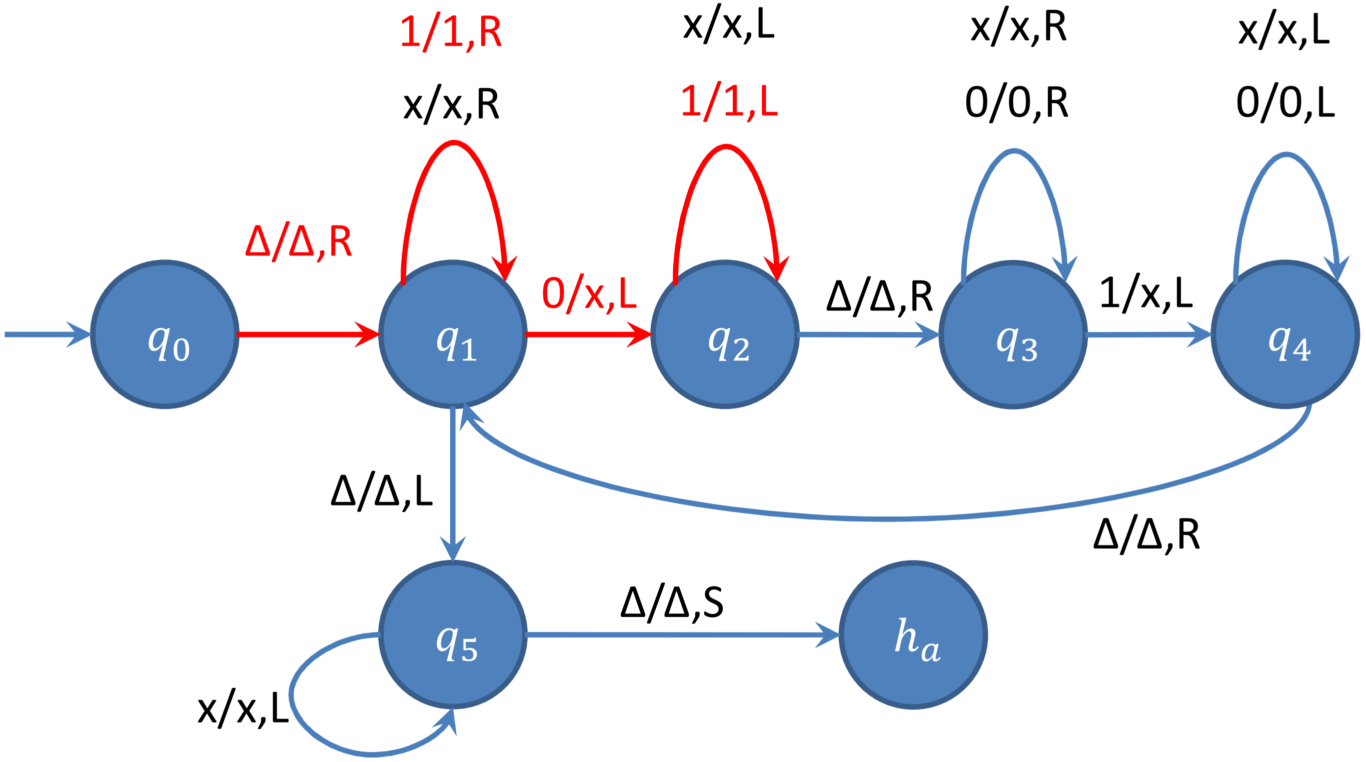
Design a TM for accepting same no of 0’s and 1’s

Δ	1	X	0	1	Δ



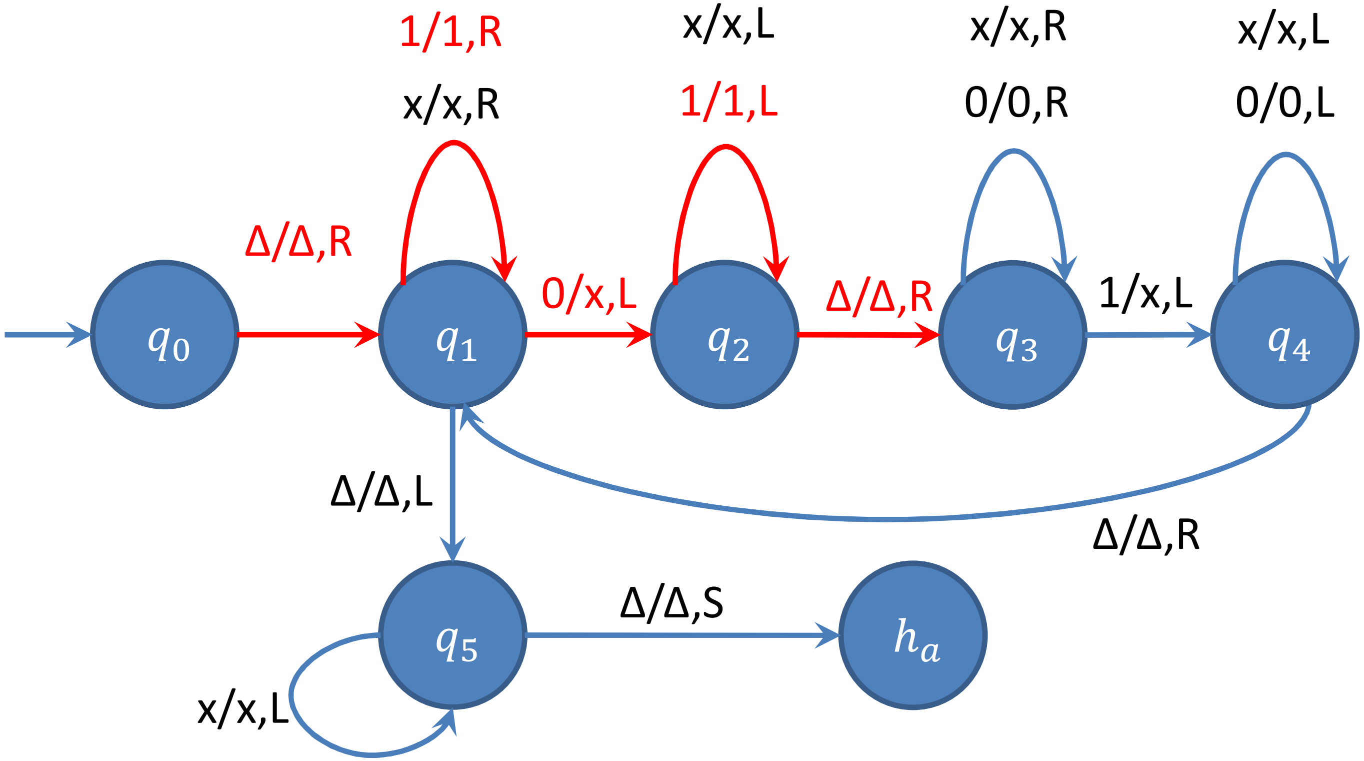
Design a TM for accepting same no of 0’s and 1’s

Δ	1	X	0	1	Δ



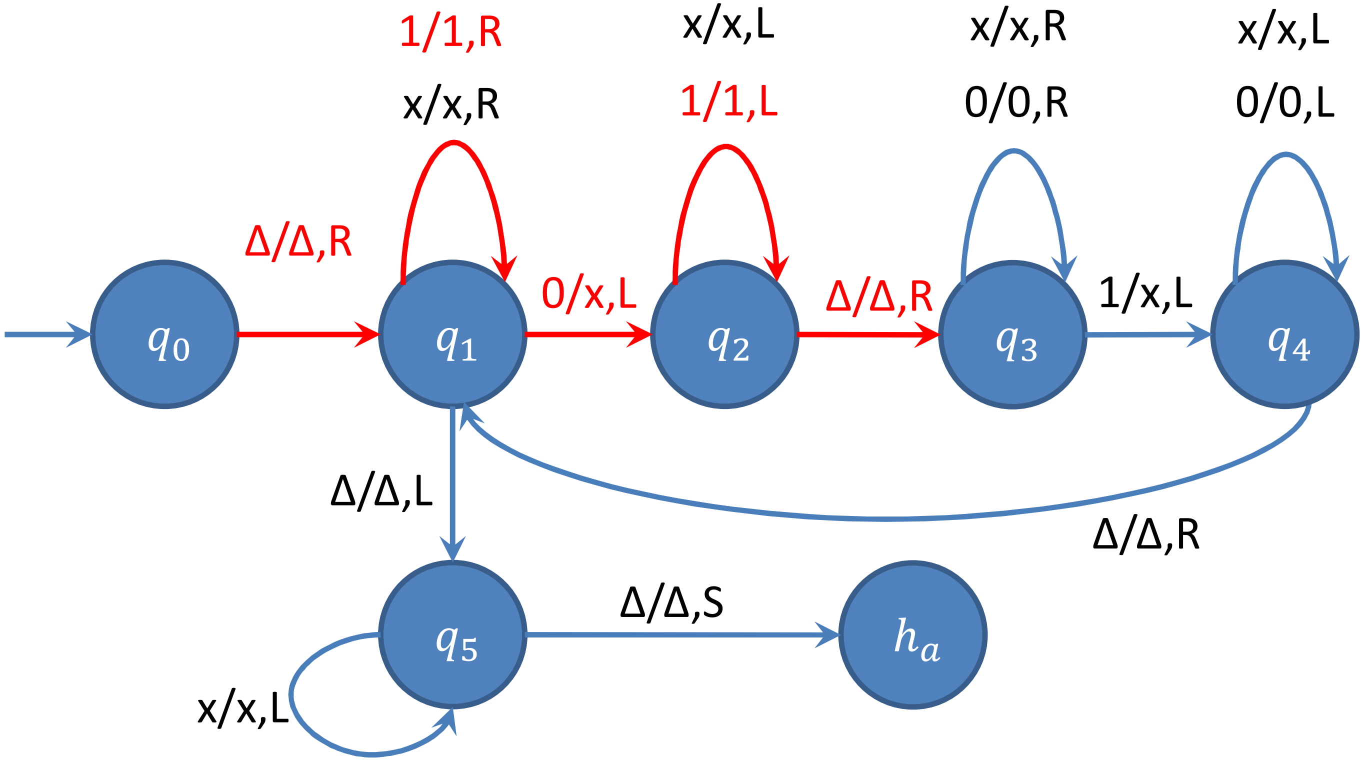
Design a TM for accepting same no of 0’s and 1’s

Δ	1	X	0	1	Δ

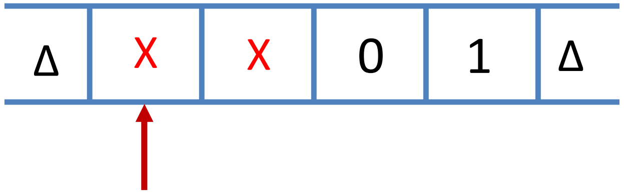


Design a TM for accepting same no of 0’s and 1’s

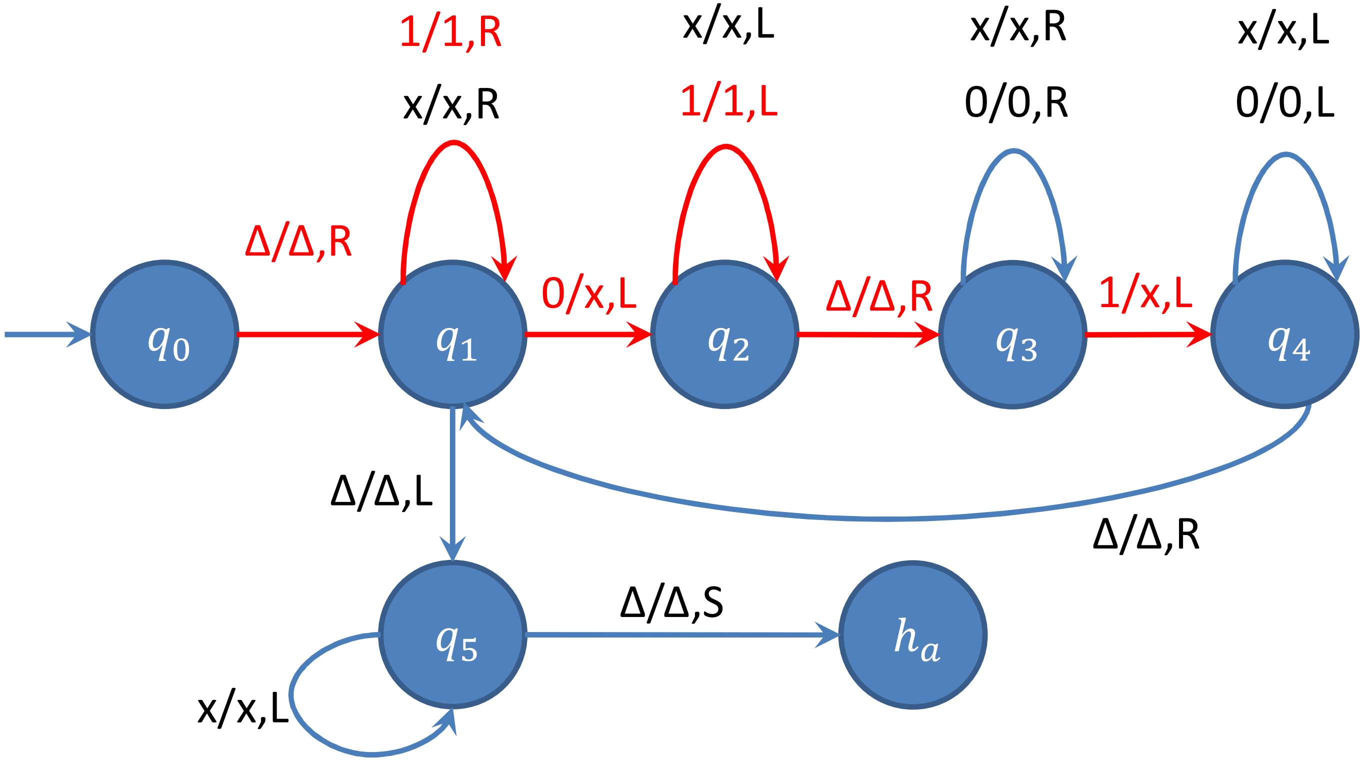
Δ	1	X	0	1	Δ



Design a TM for accepting same no of 0's and 1's

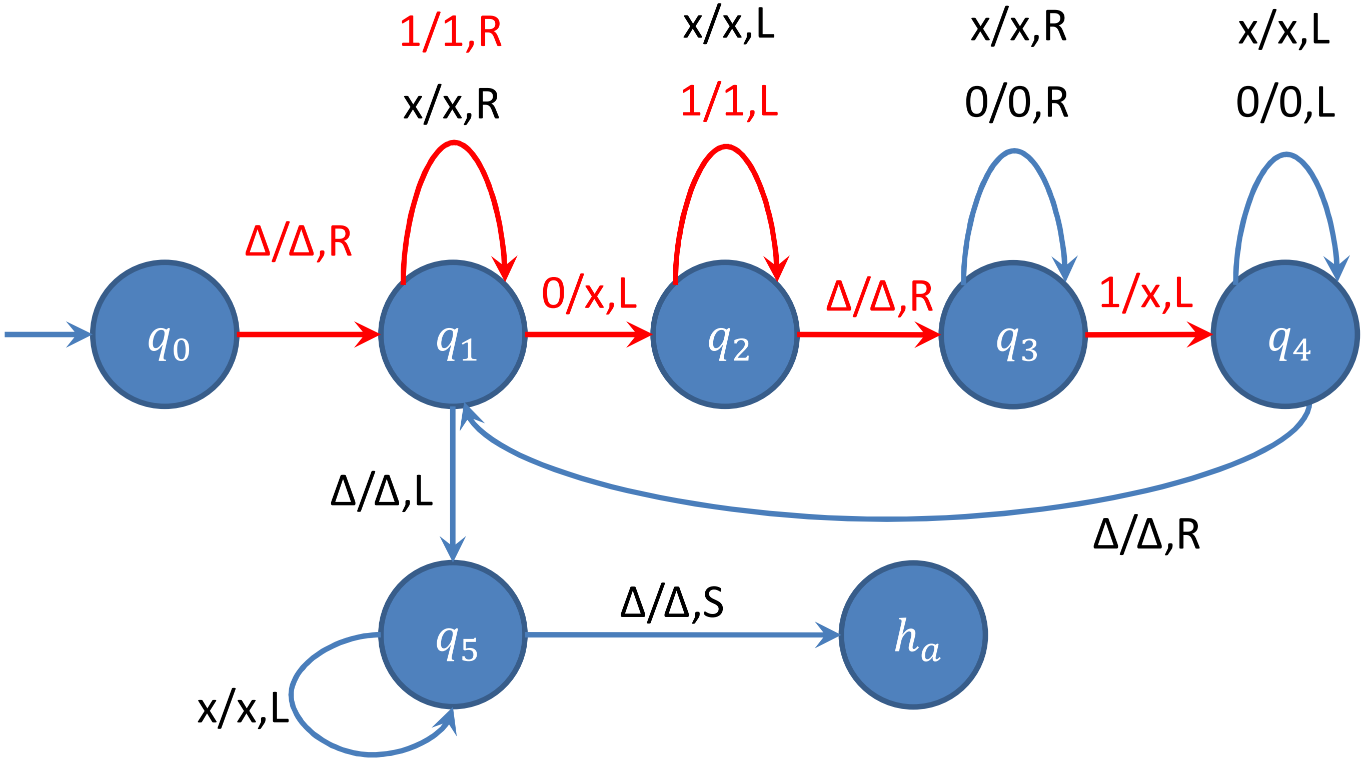


Δ	X	X	0	1	Δ



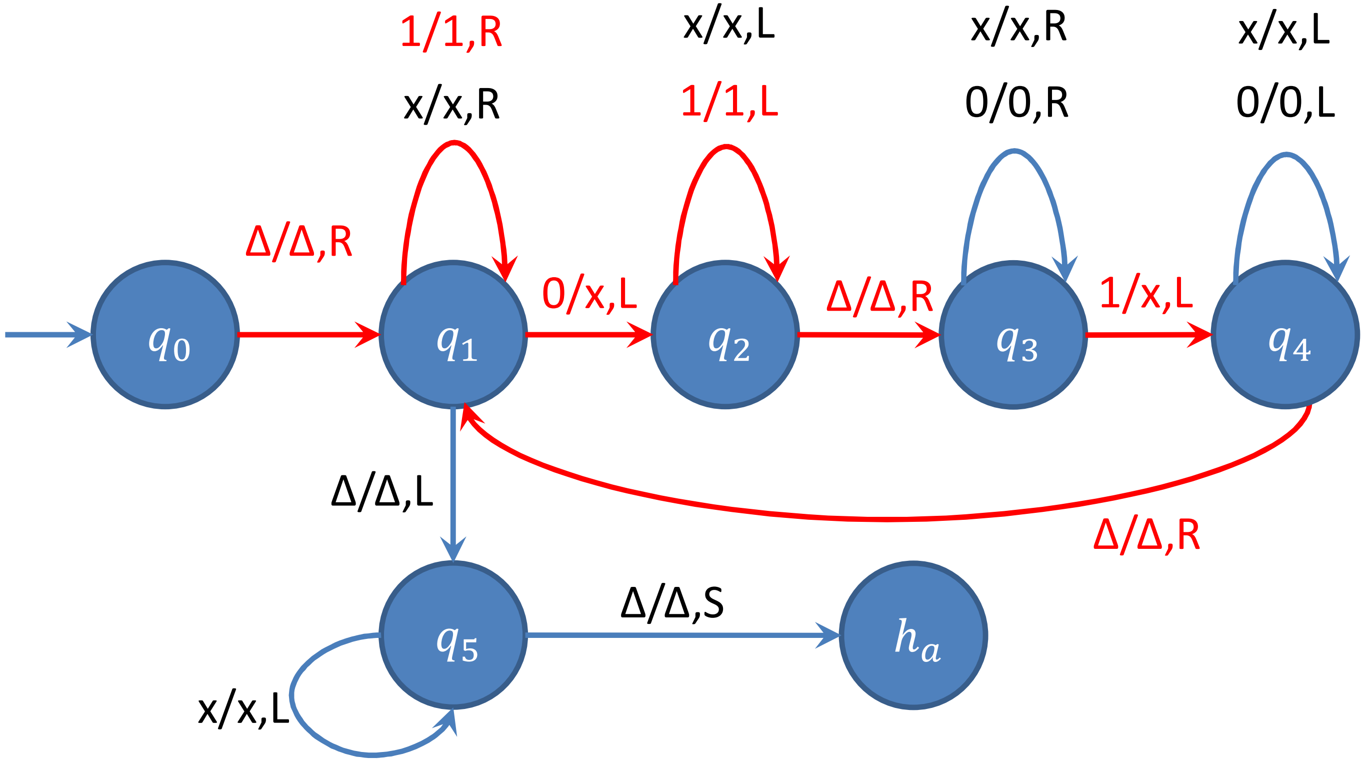
Design a TM for accepting same no of 0’s and 1’s

Δ	X	X	0	1	Δ

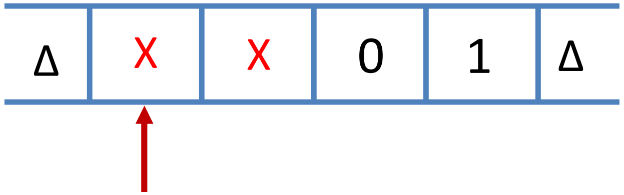


Design a TM for accepting same no of 0’s and 1’s

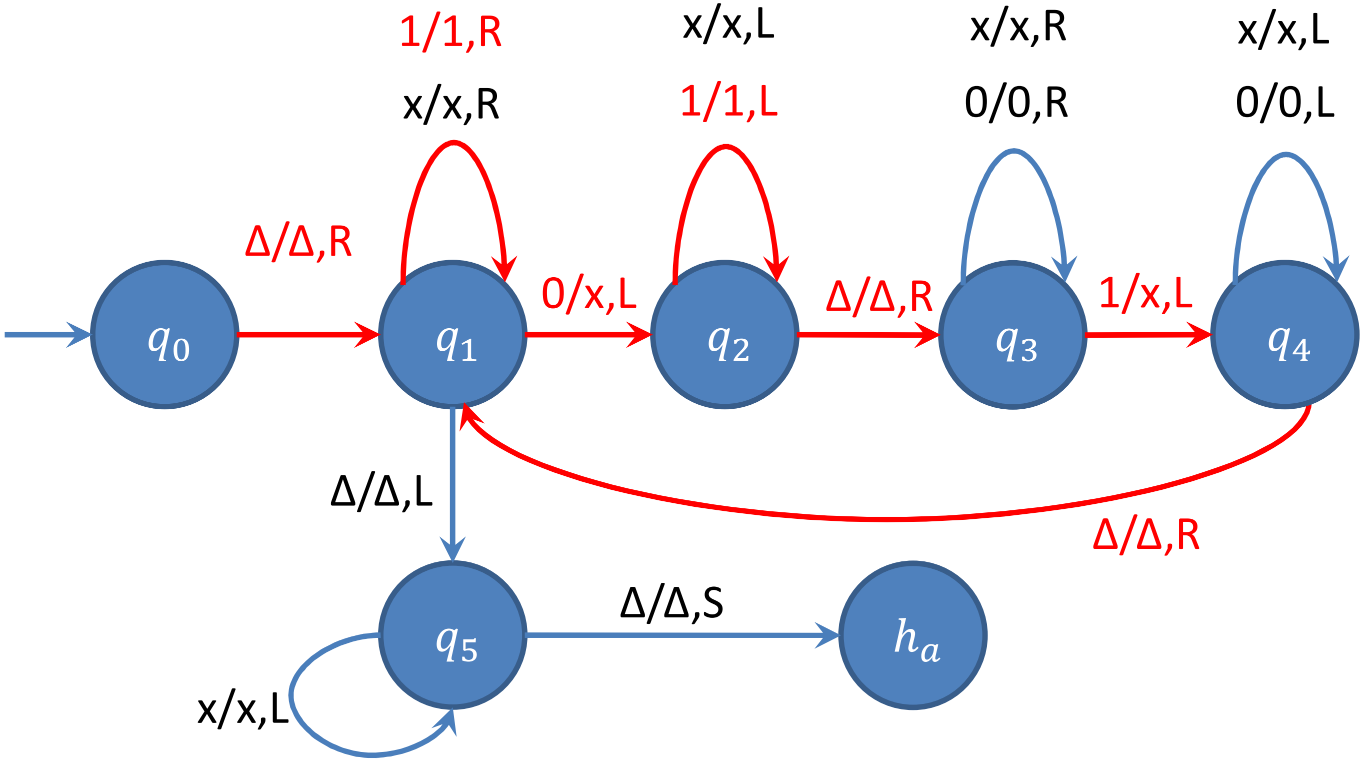
Δ	X	X	0	1	Δ



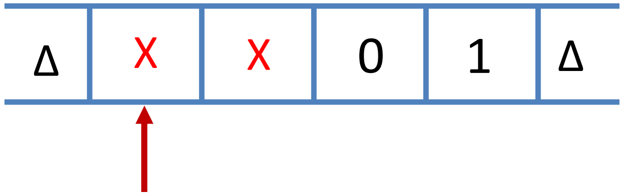
Design a TM for accepting same no of 0’s and 1’s



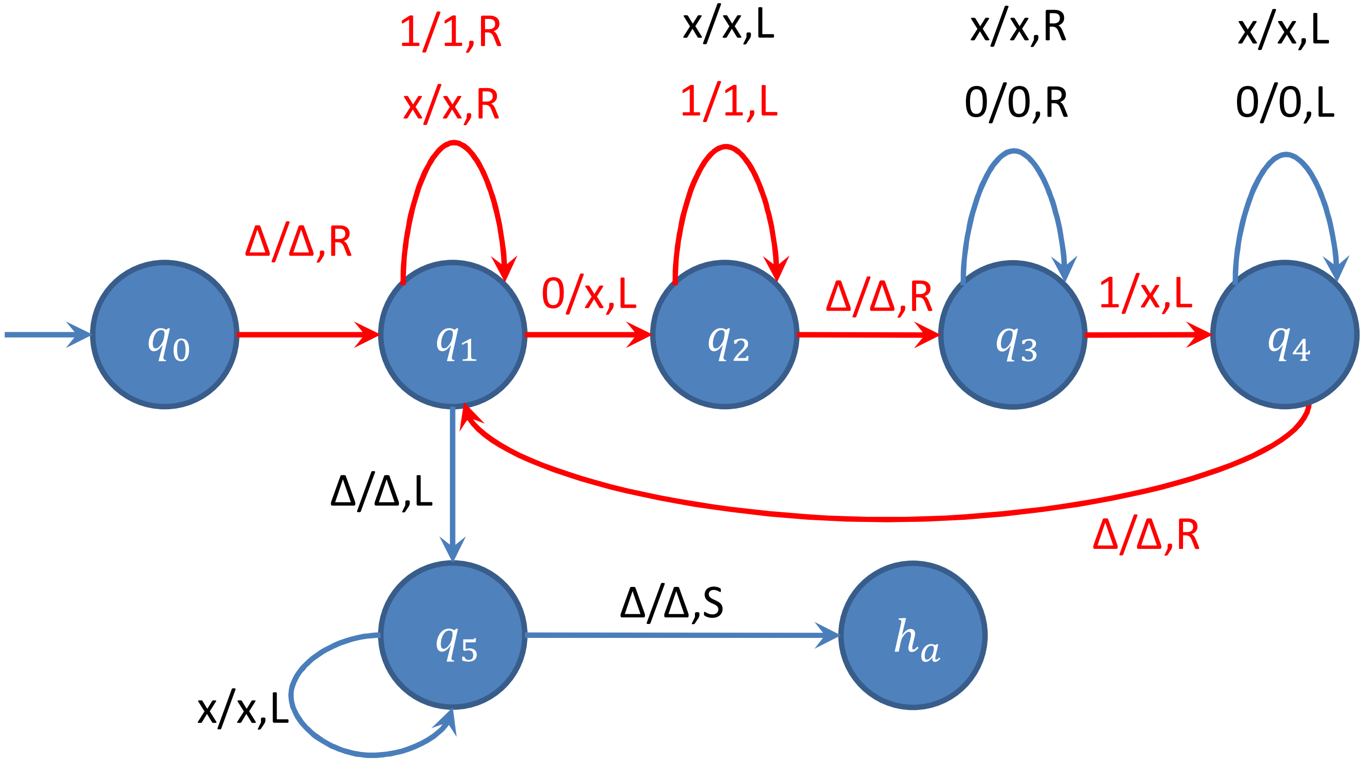
Δ	X	X	0	1	Δ



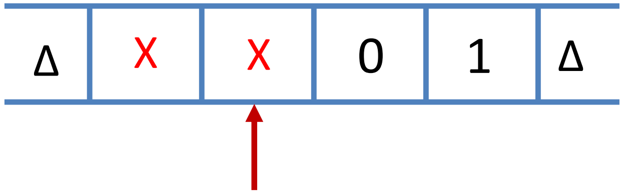
Design a TM for accepting same no of 0’s and 1’s



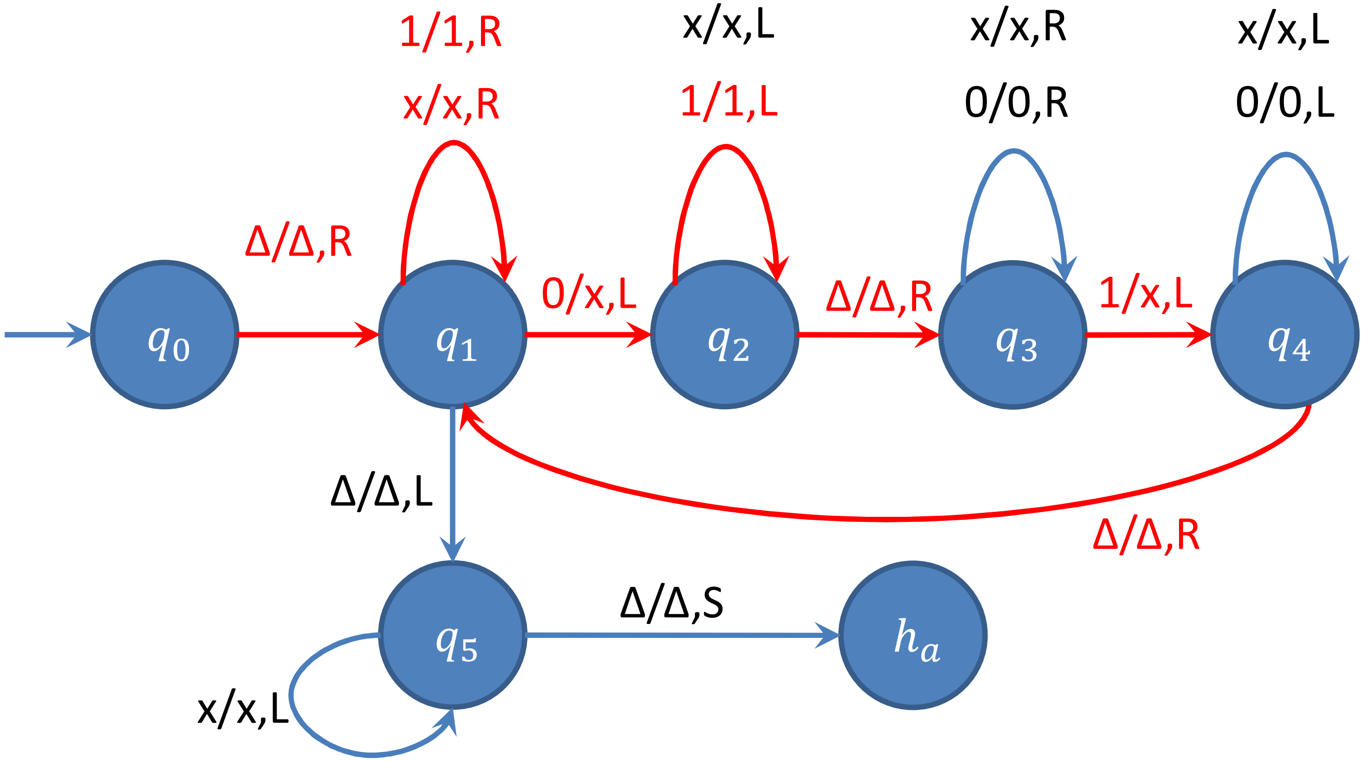
Δ	X	X	0	1	Δ



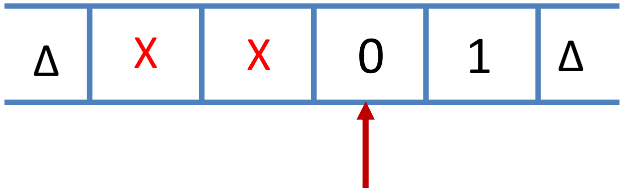
Design a TM for accepting same no of 0’s and 1’s

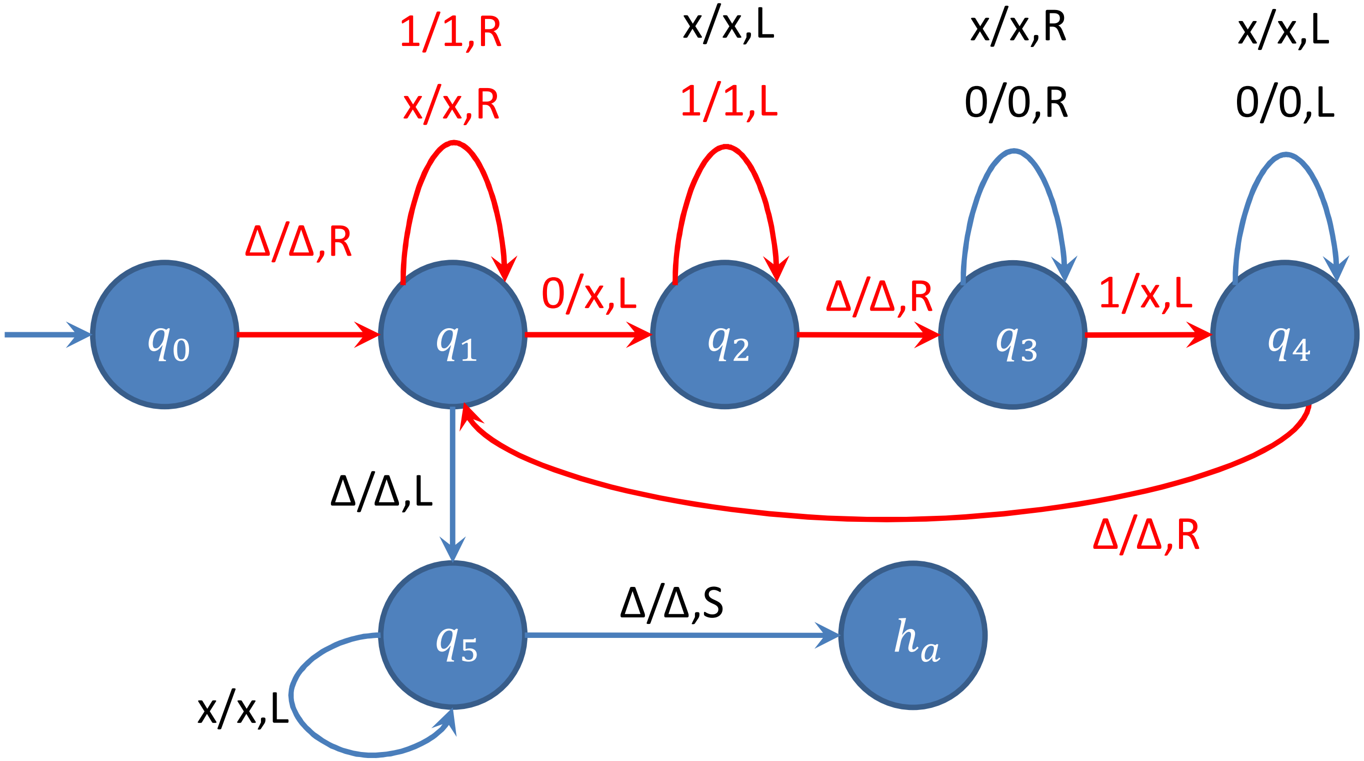


Δ	X	X	0	1	Δ

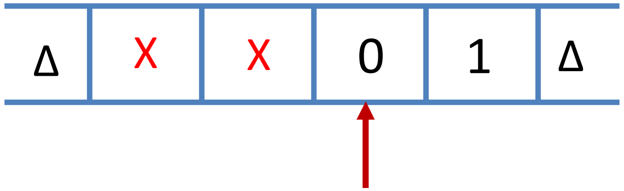


Design a TM for accepting same no of 0’s and 1’s

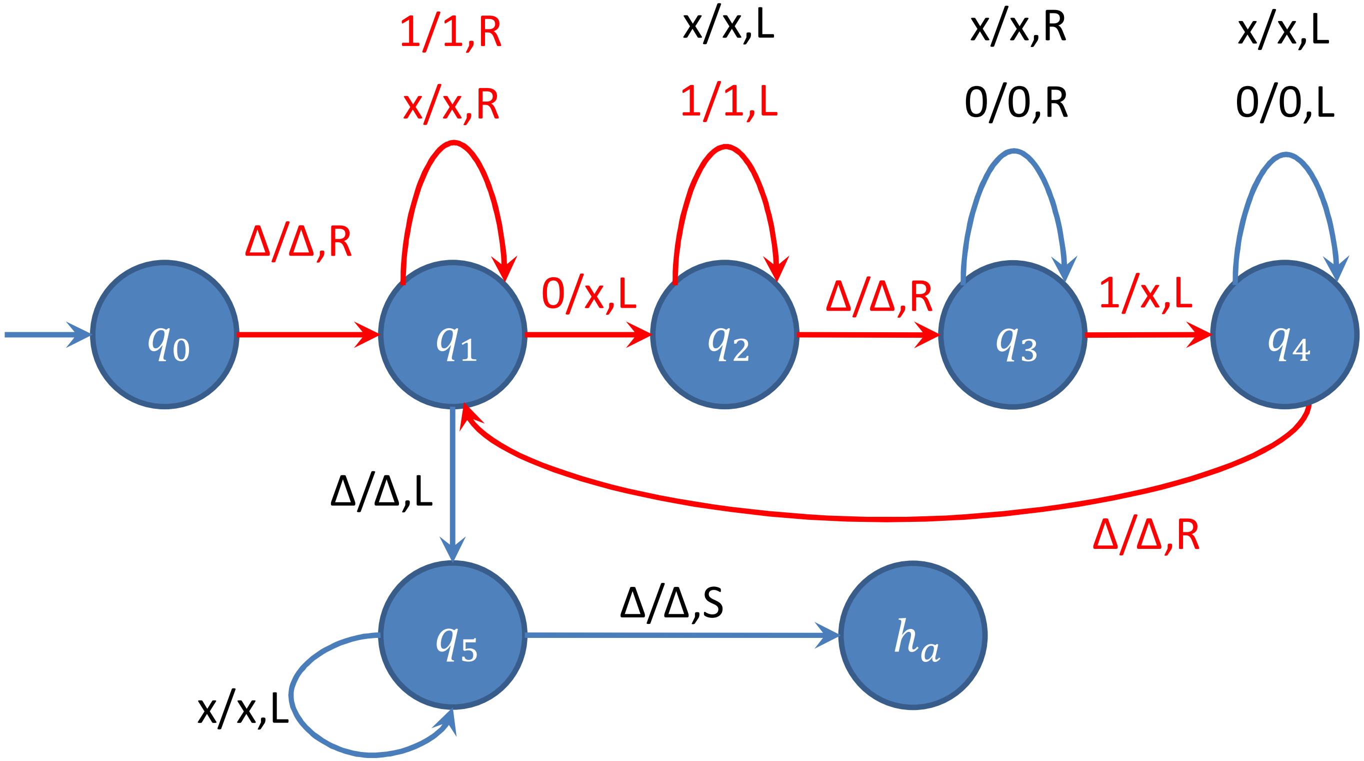




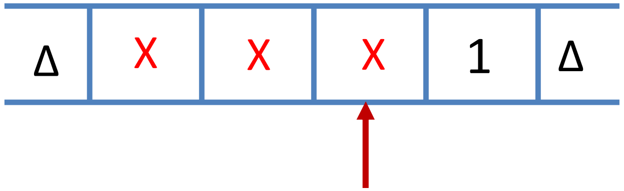
Design a TM for accepting same no of 0’s and 1’s



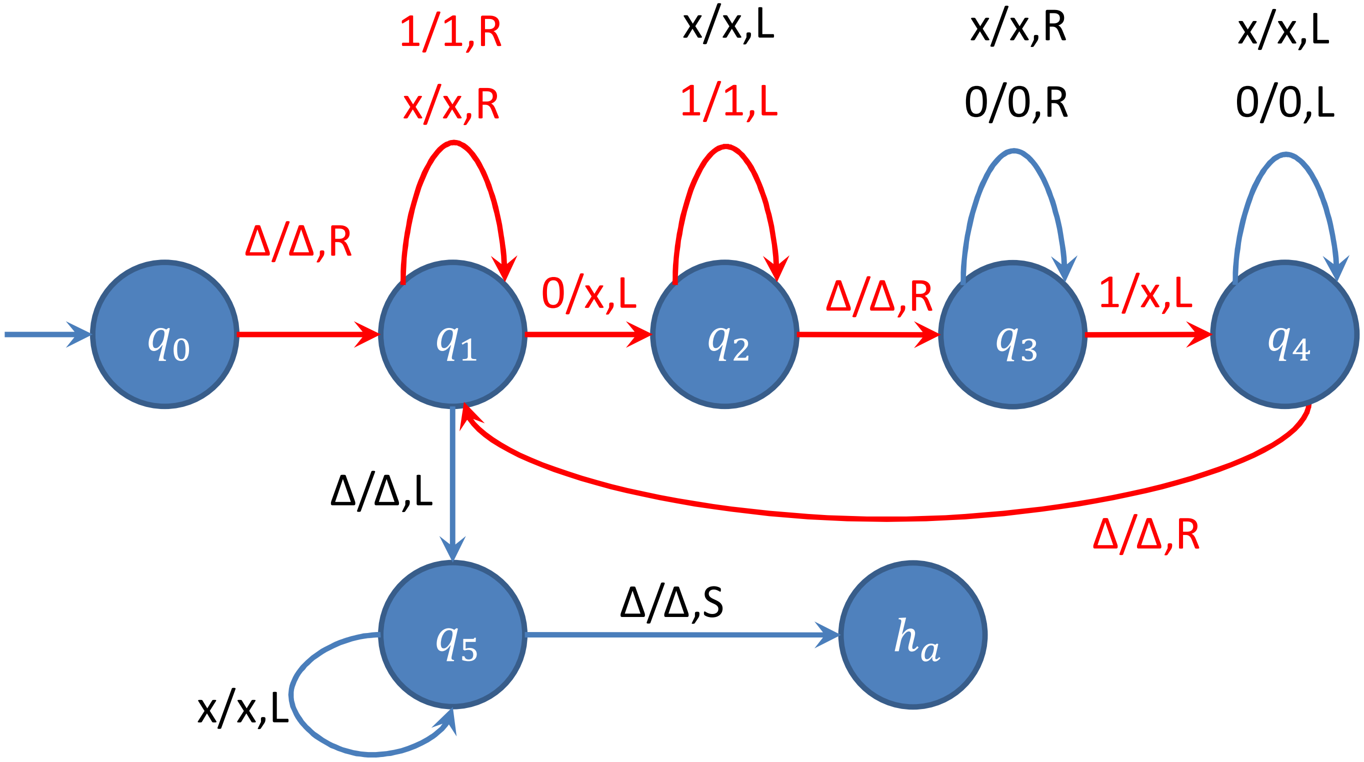
Δ	X	X	0	1	Δ



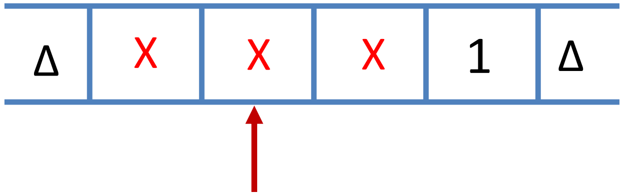
Design a TM for accepting same no of 0's and 1's



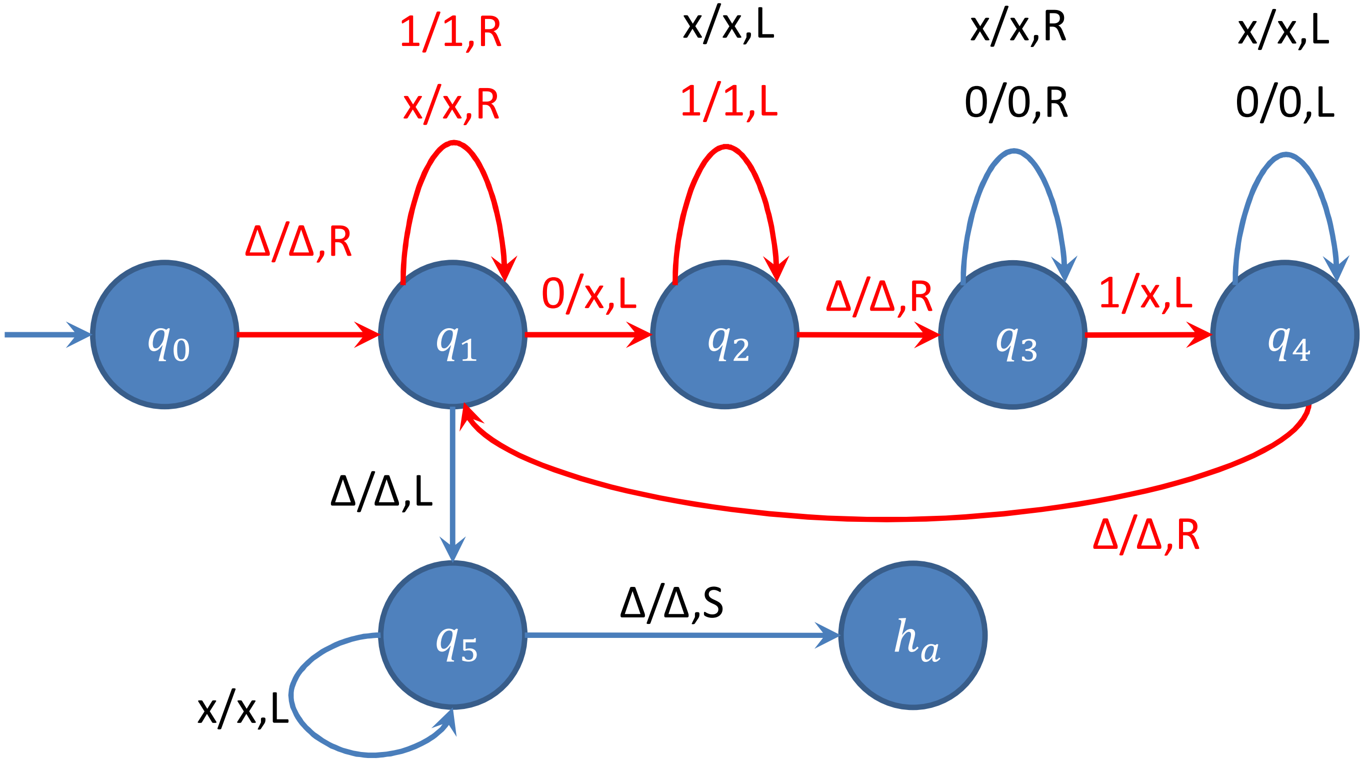
Δ	X	X	X	1	Δ



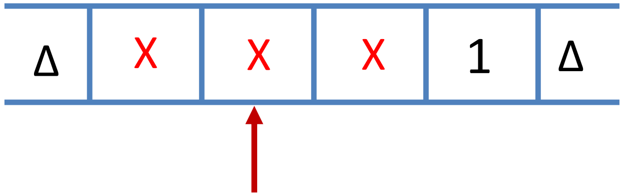
Design a TM for accepting same no of 0’s and 1’s



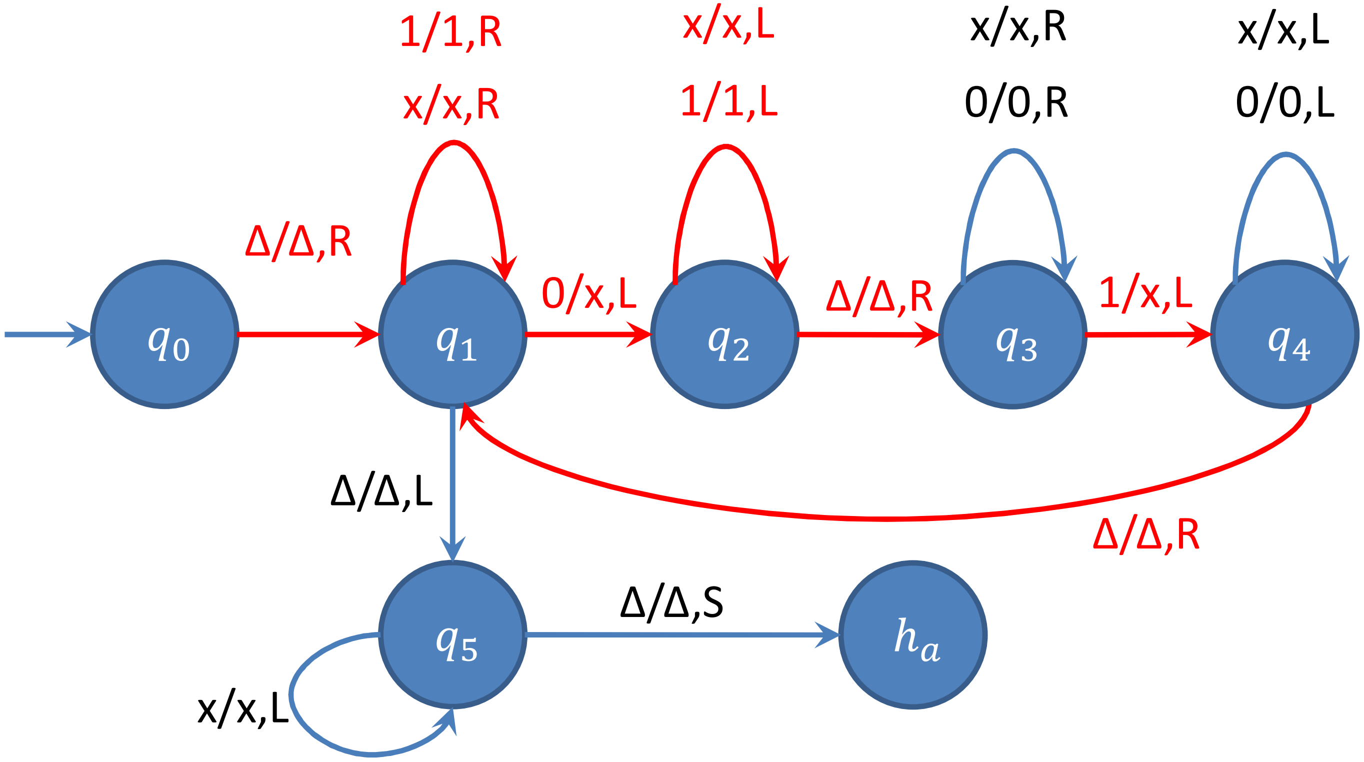
Δ	X	X	X	1	Δ



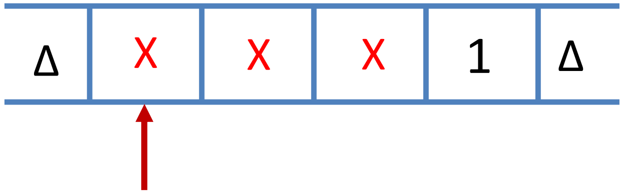
Design a TM for accepting same no of 0’s and 1’s



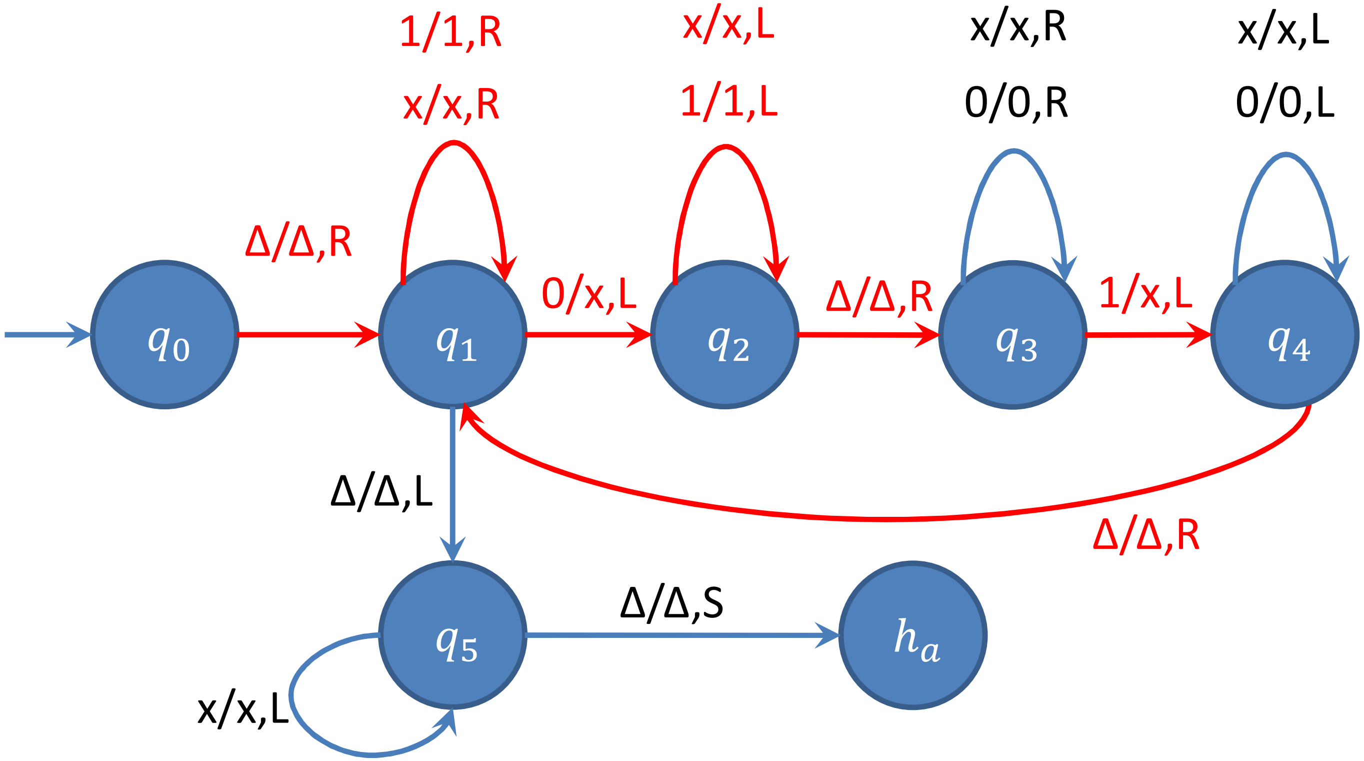
Δ	X	X	X	1	Δ



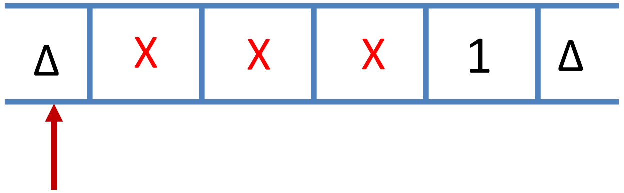
Design a TM for accepting same no of 0’s and 1’s



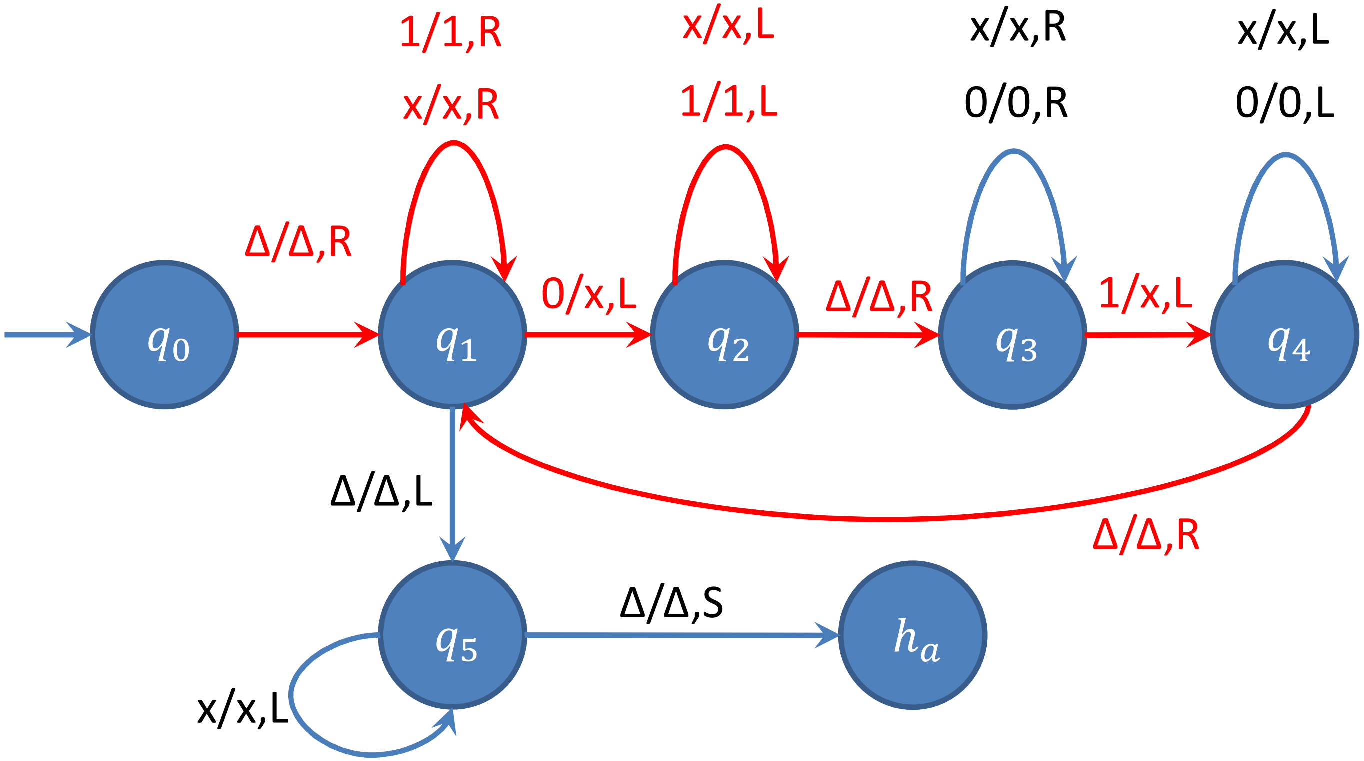
Δ	X	X	X	1	Δ



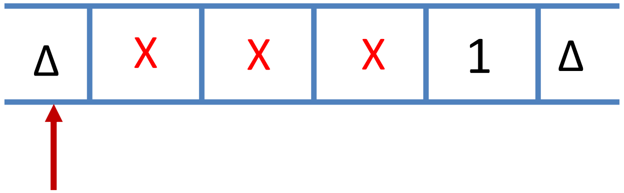
Design a TM for accepting same no of 0's and 1's

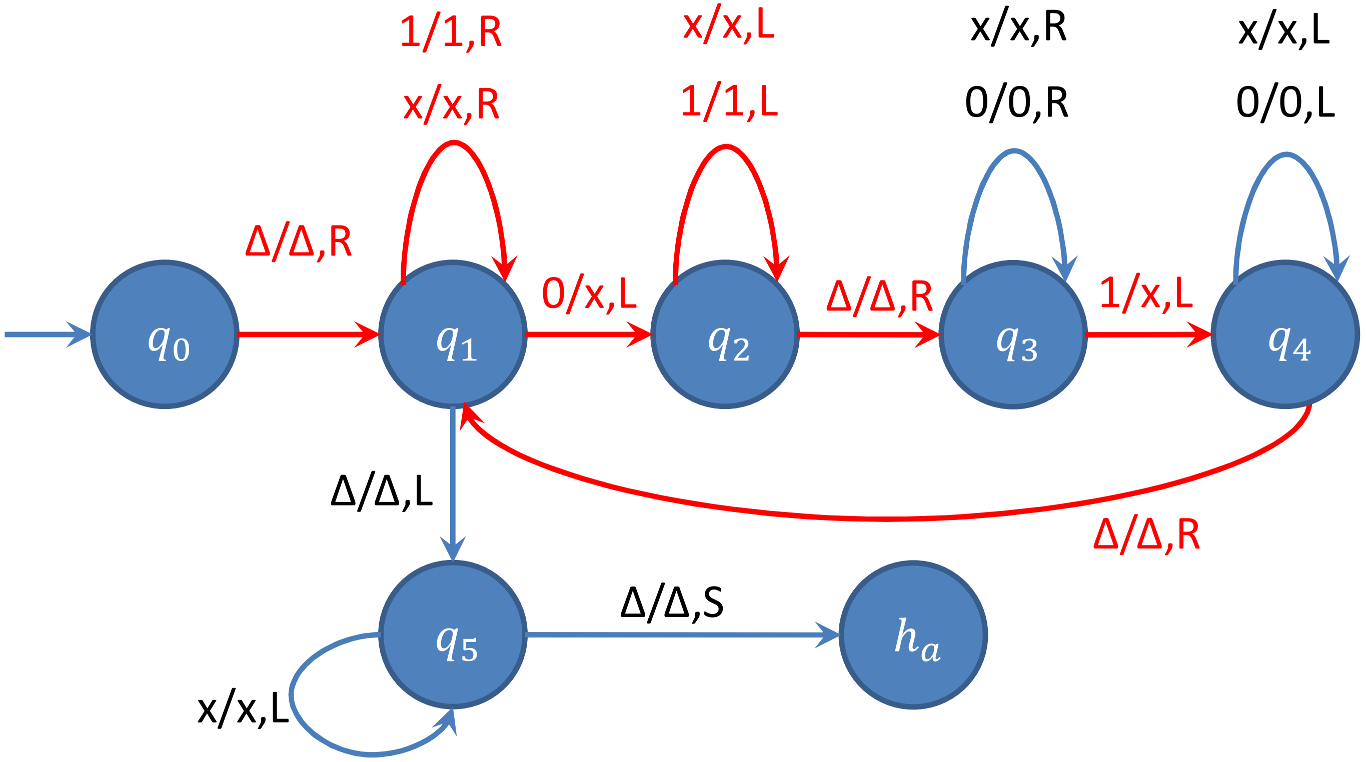


Δ	X	X	X	1	Δ

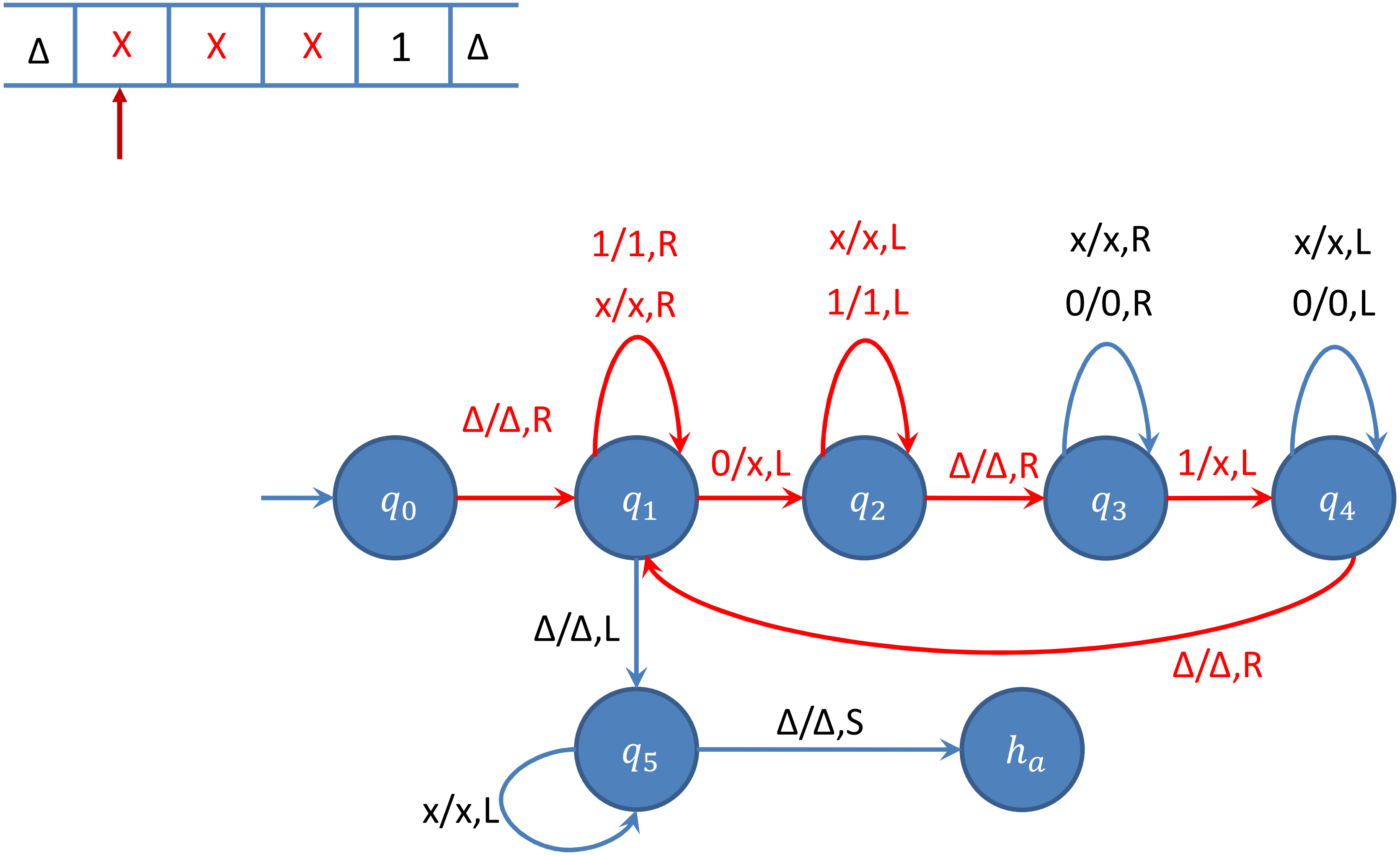


Design a TM for accepting same no of 0’s and 1’s



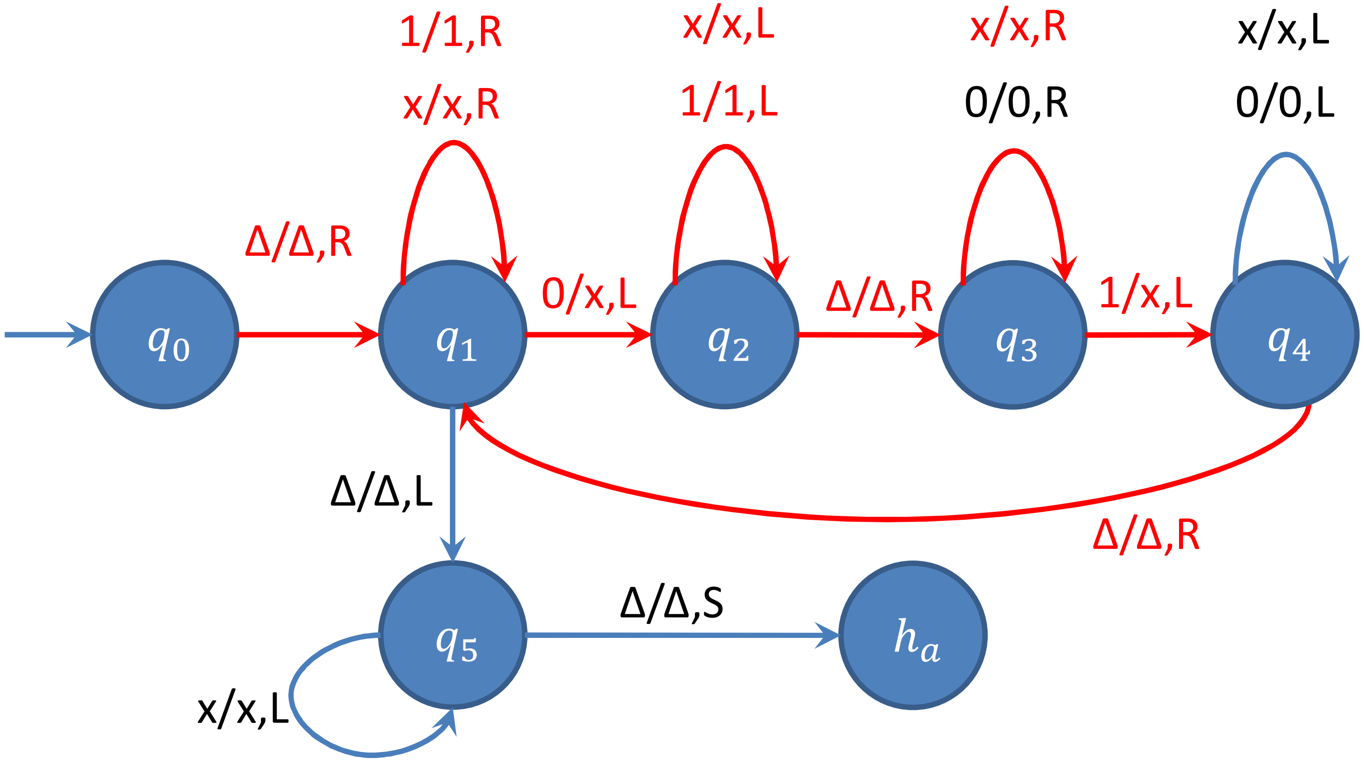


Design a TM for accepting same no of 0’s and 1’s

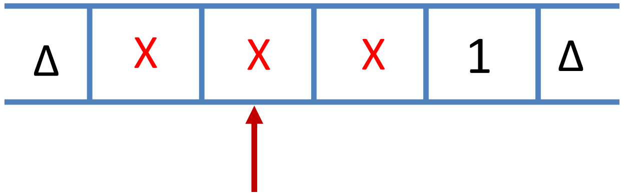


Design a TM for accepting same no of 0's and 1's

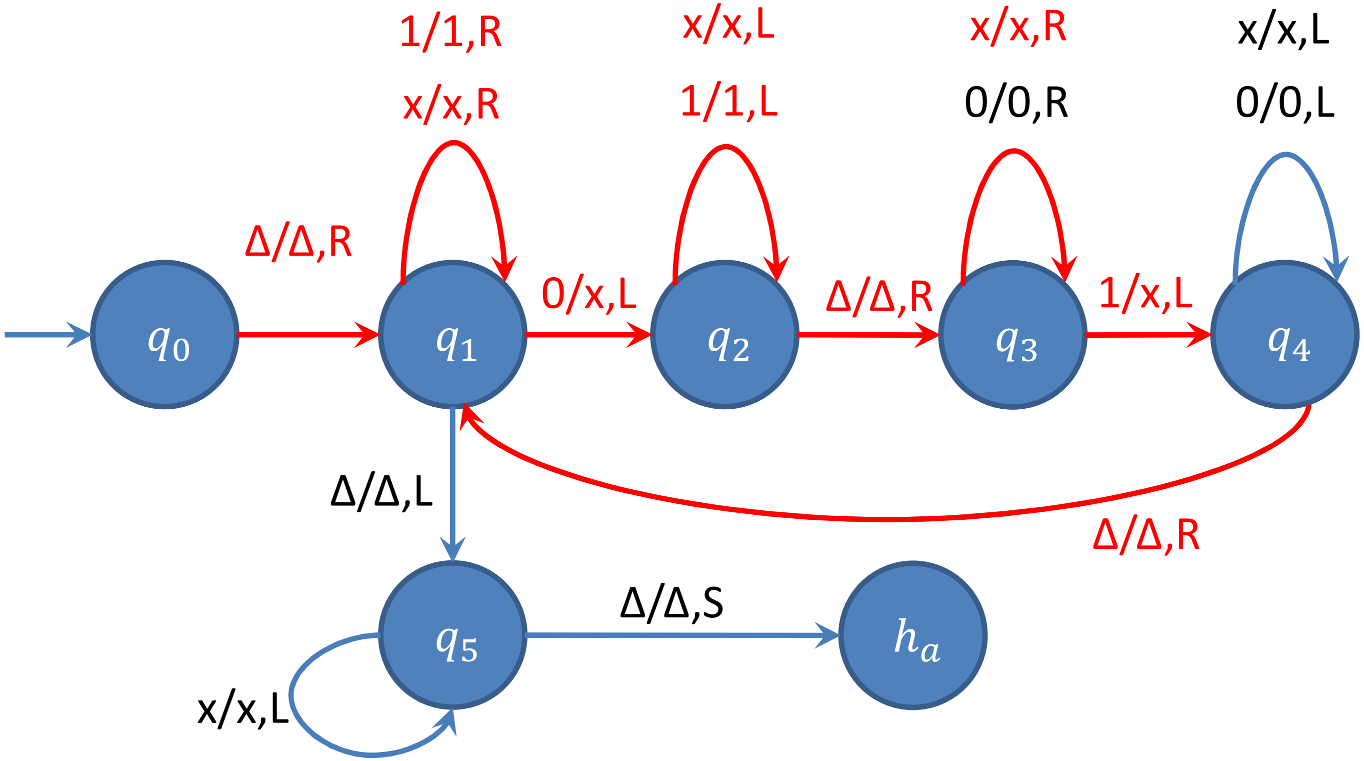
Δ	X	X	X	1	Δ



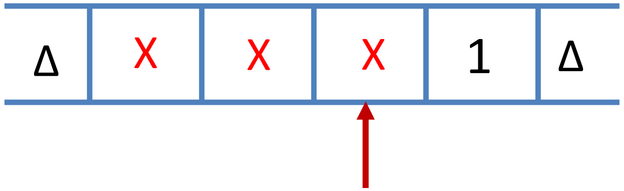
Design a TM for accepting same no of 0’s and 1’s

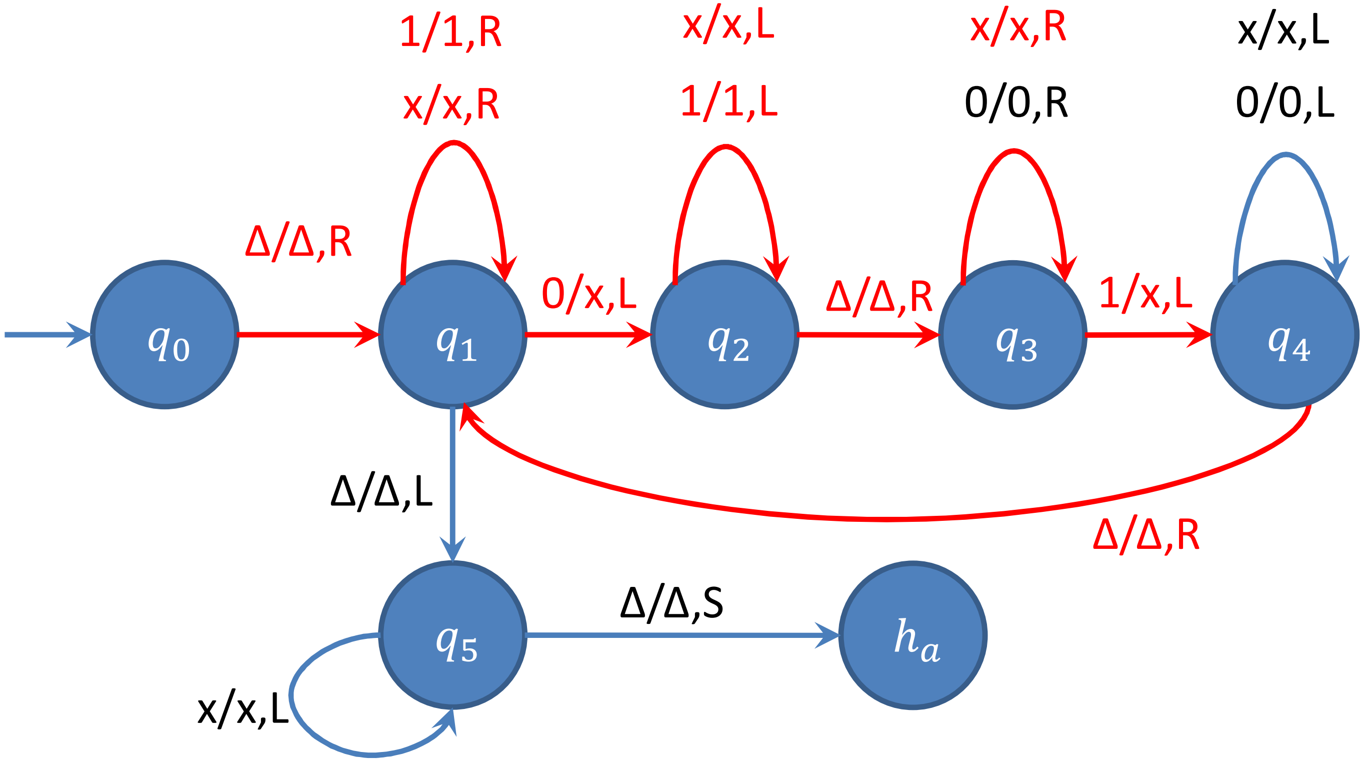


Δ	X	X	X	1	Δ

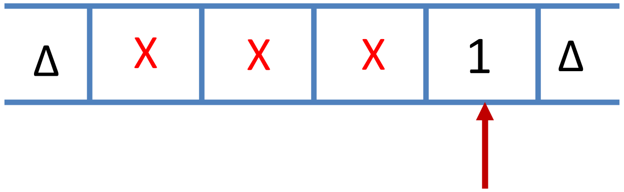


Design a TM for accepting same no of 0’s and 1’s

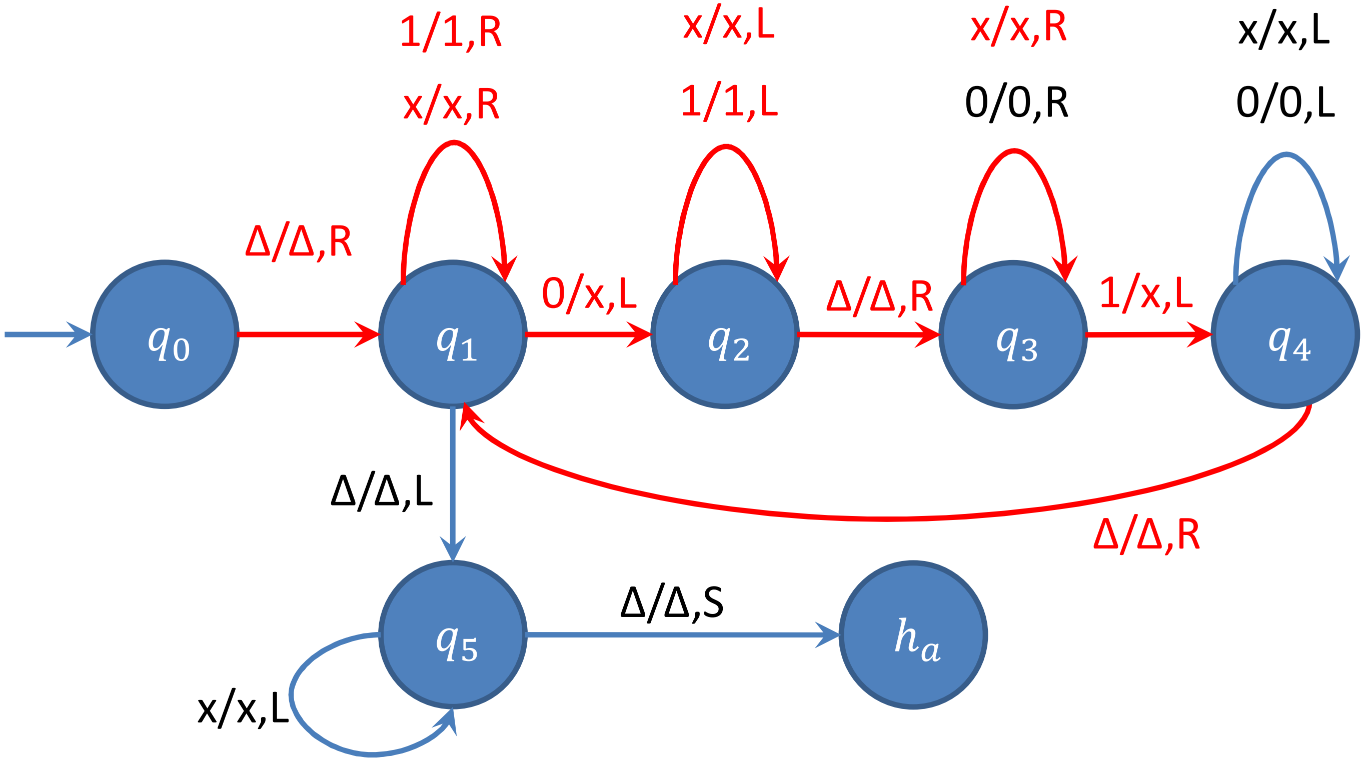




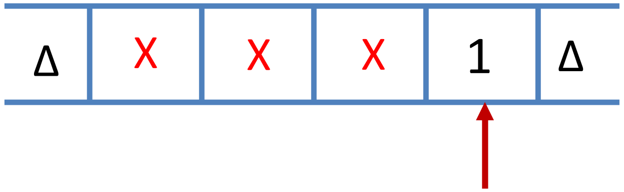
Design a TM for accepting same no of 0’s and 1’s



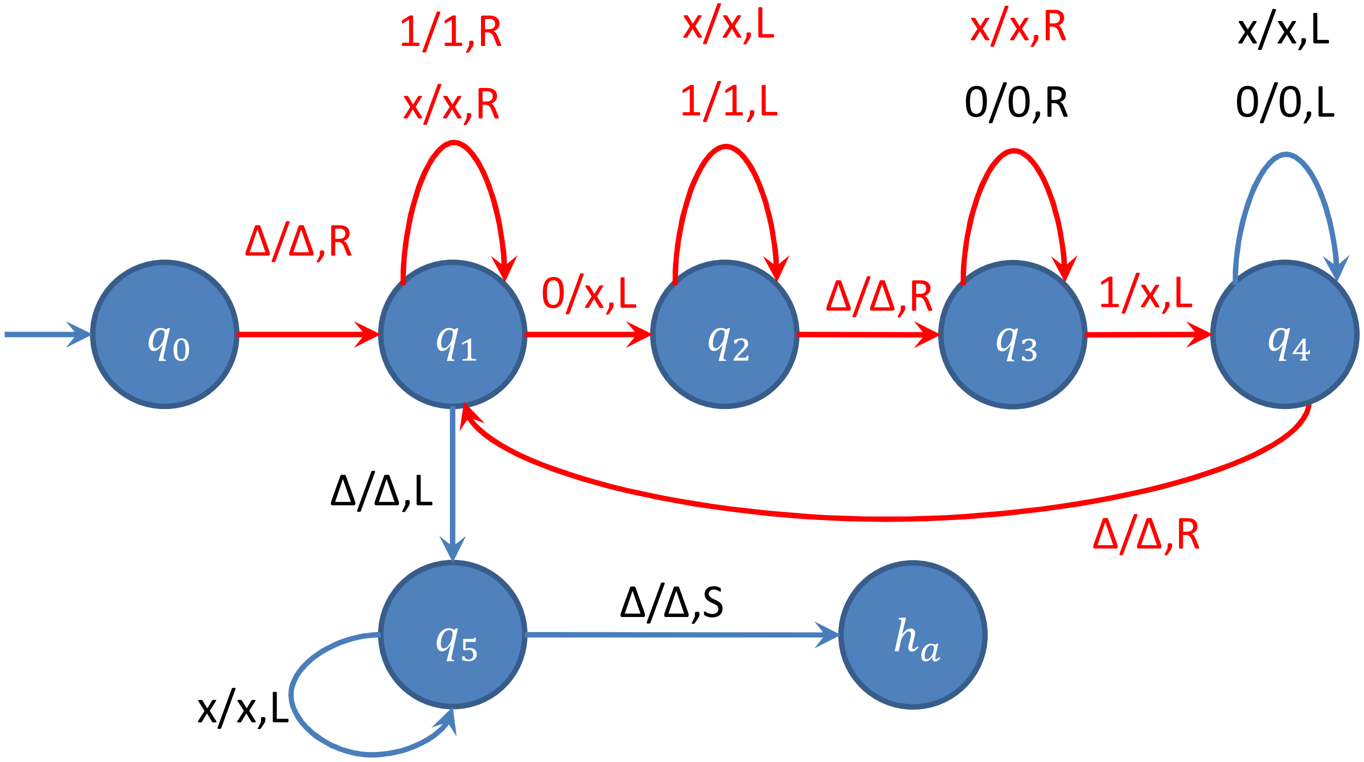
Δ	X	X	X	1	Δ



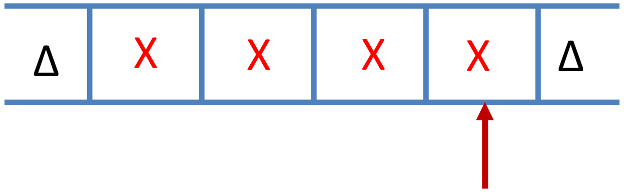
Design a TM for accepting same no of 0’s and 1’s

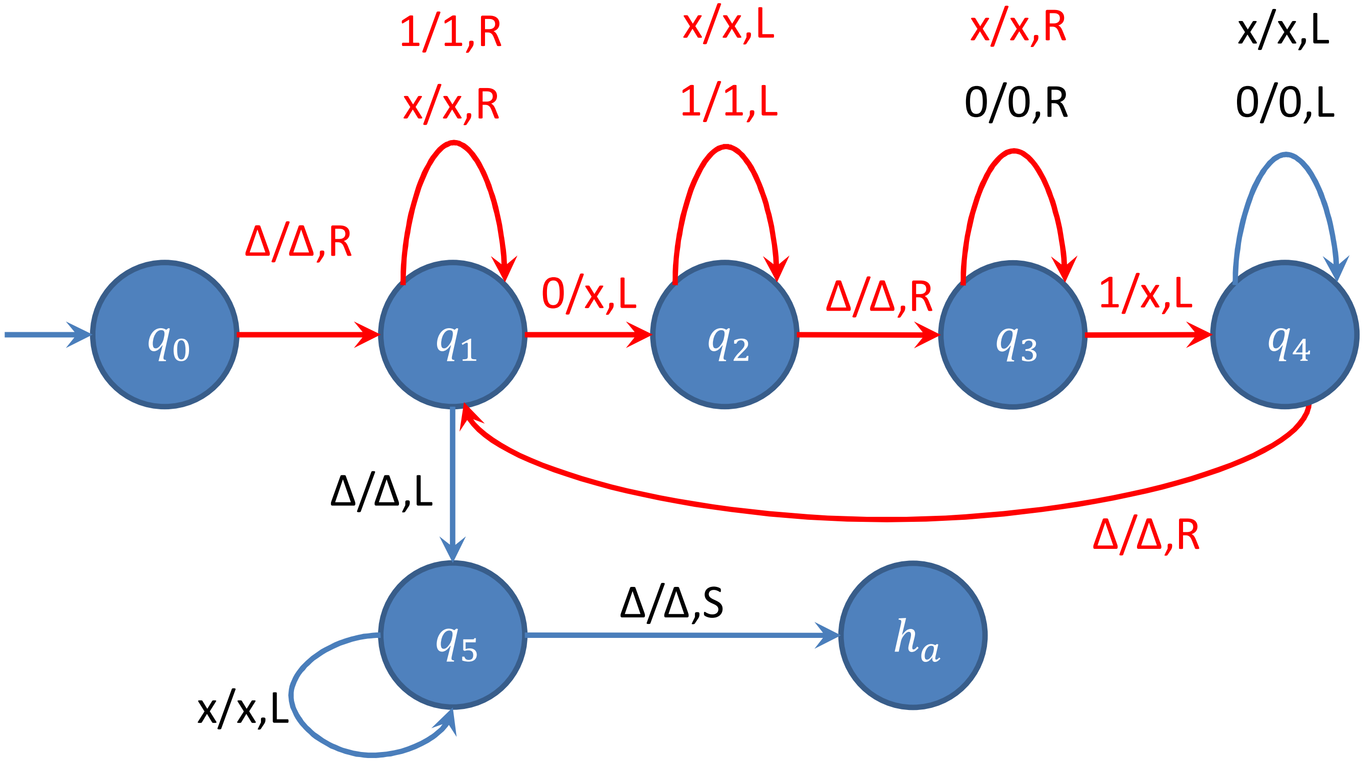


Δ	X	X	X	1	Δ

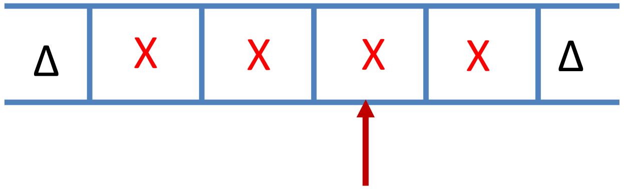


Design a TM for accepting same no of 0’s and 1’s

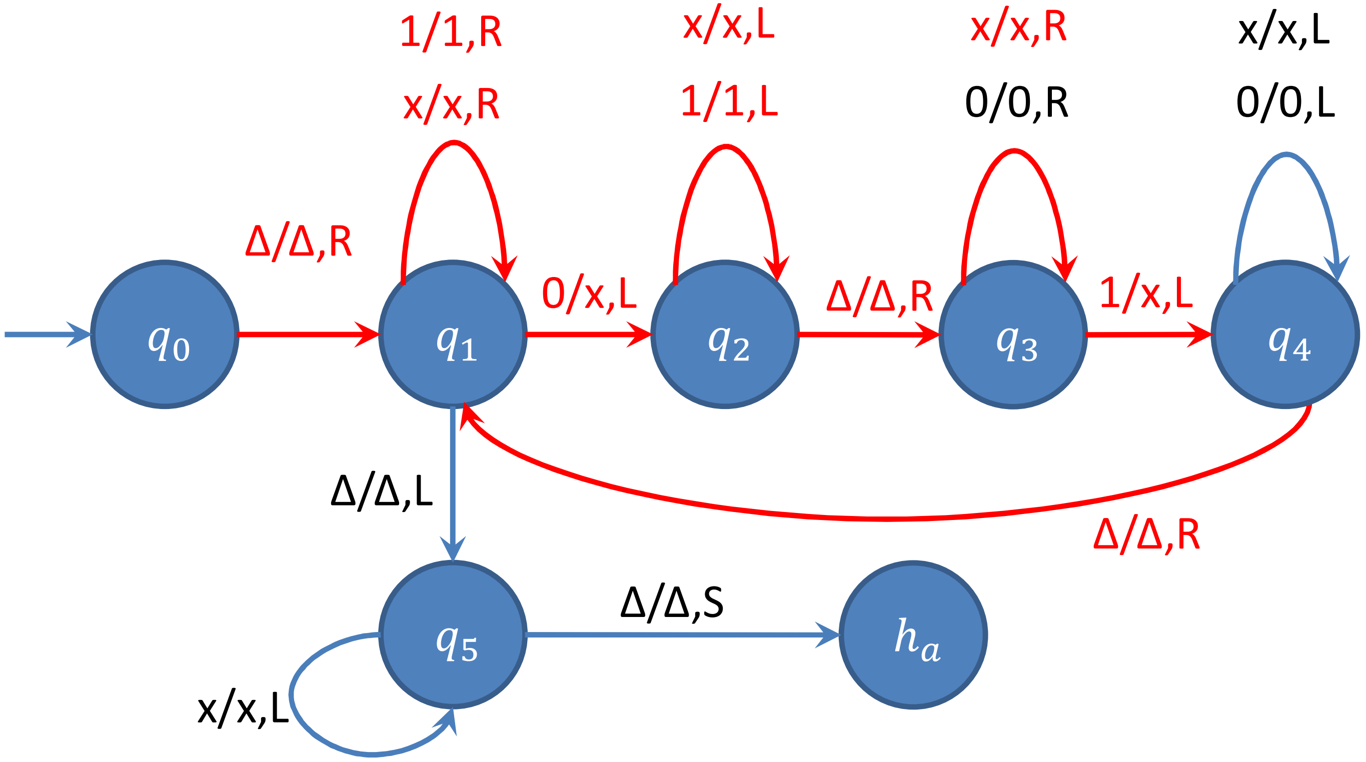




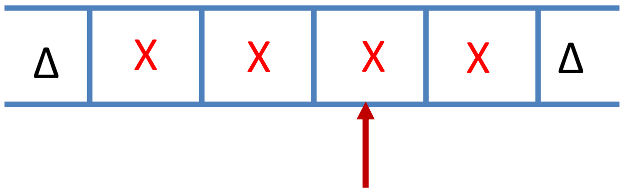
Design a TM for accepting same no of 0’s and 1’s

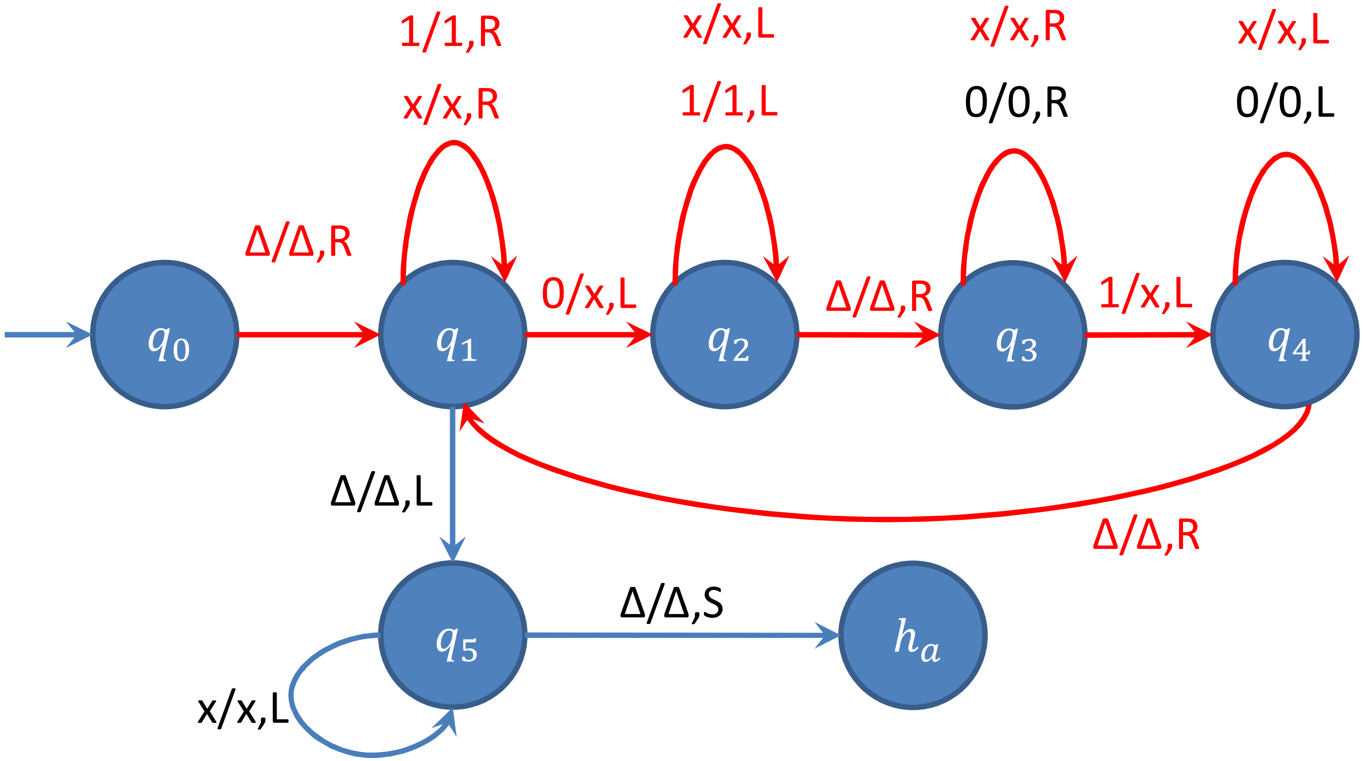


Δ	X	X	X	X	Δ



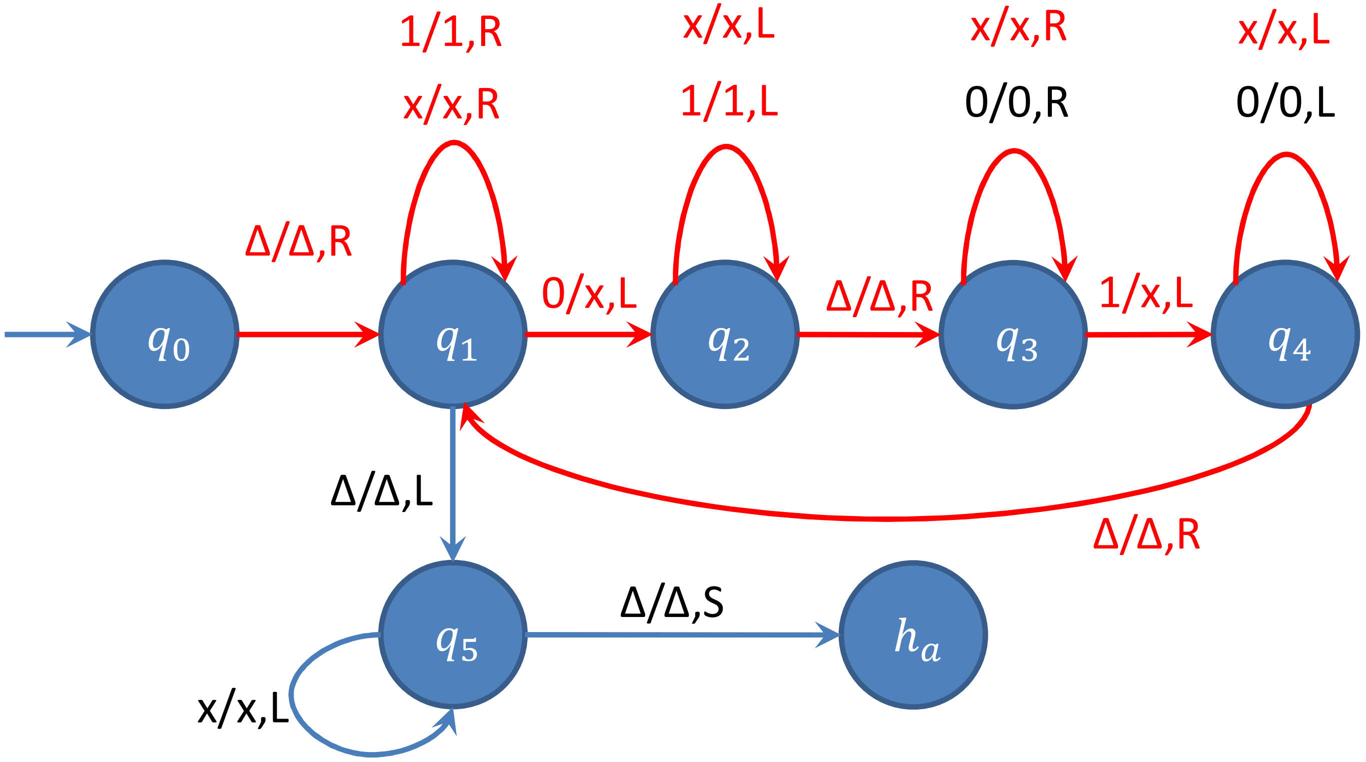
Design a TM for accepting same no of 0’s and 1’s



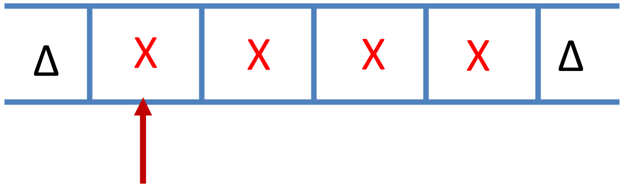


Design a TM for accepting same no of 0’s and 1’s

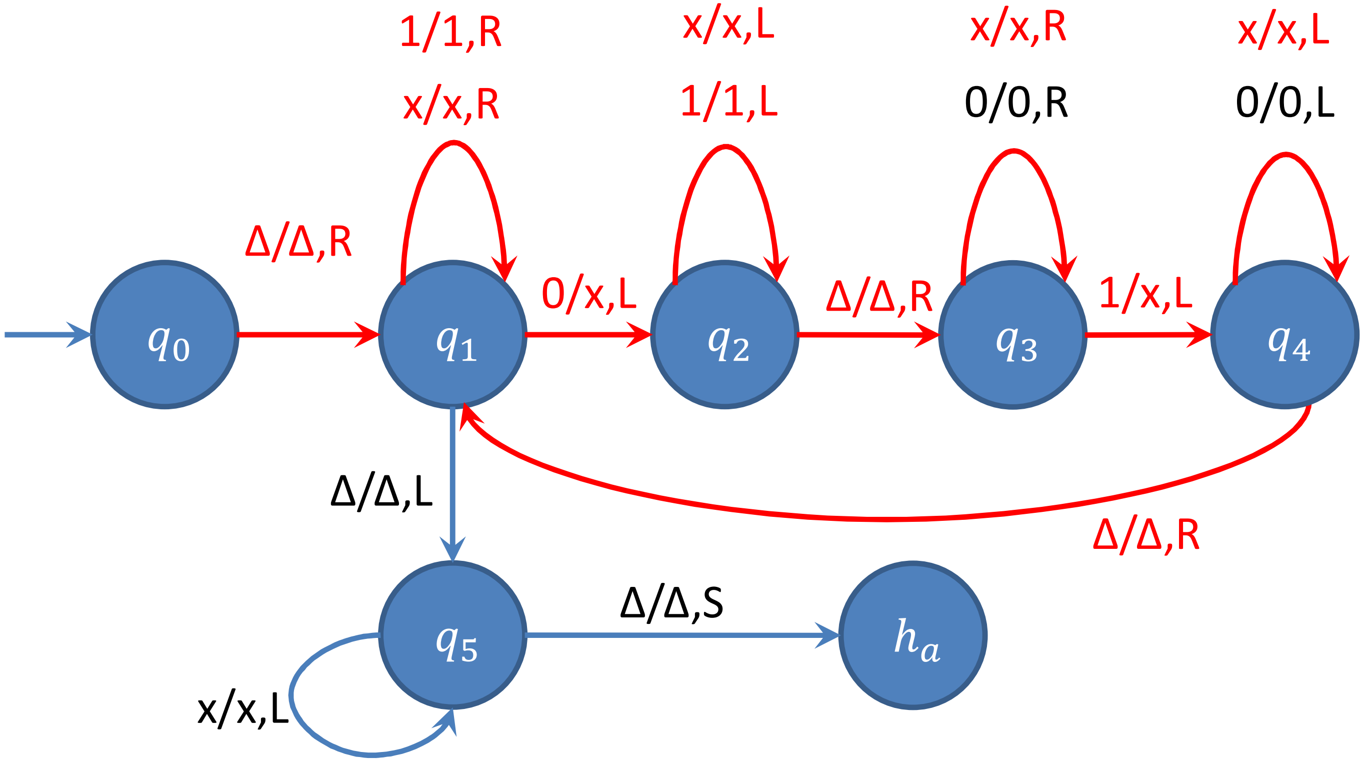
Δ	X	X	X	X	Δ



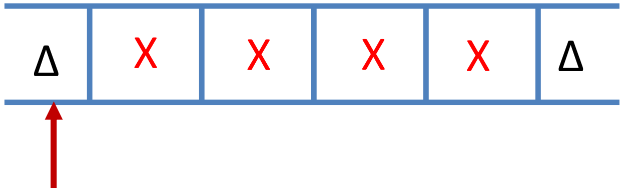
Design a TM for accepting same no of 0’s and 1’s

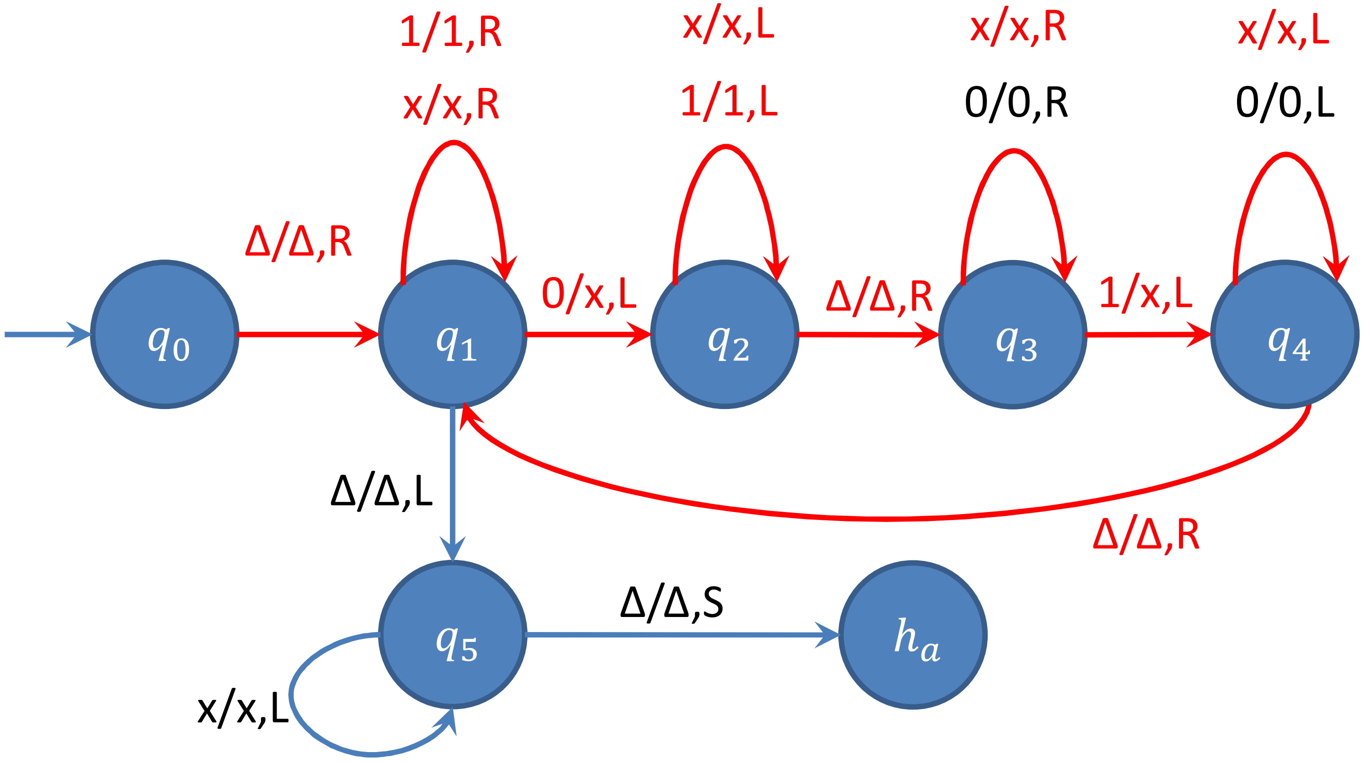


Δ	X	X	X	X	Δ

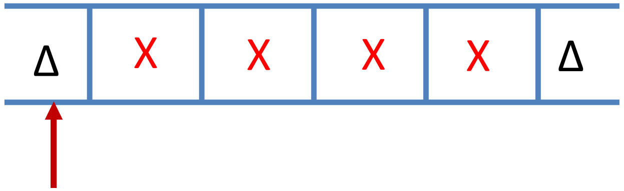


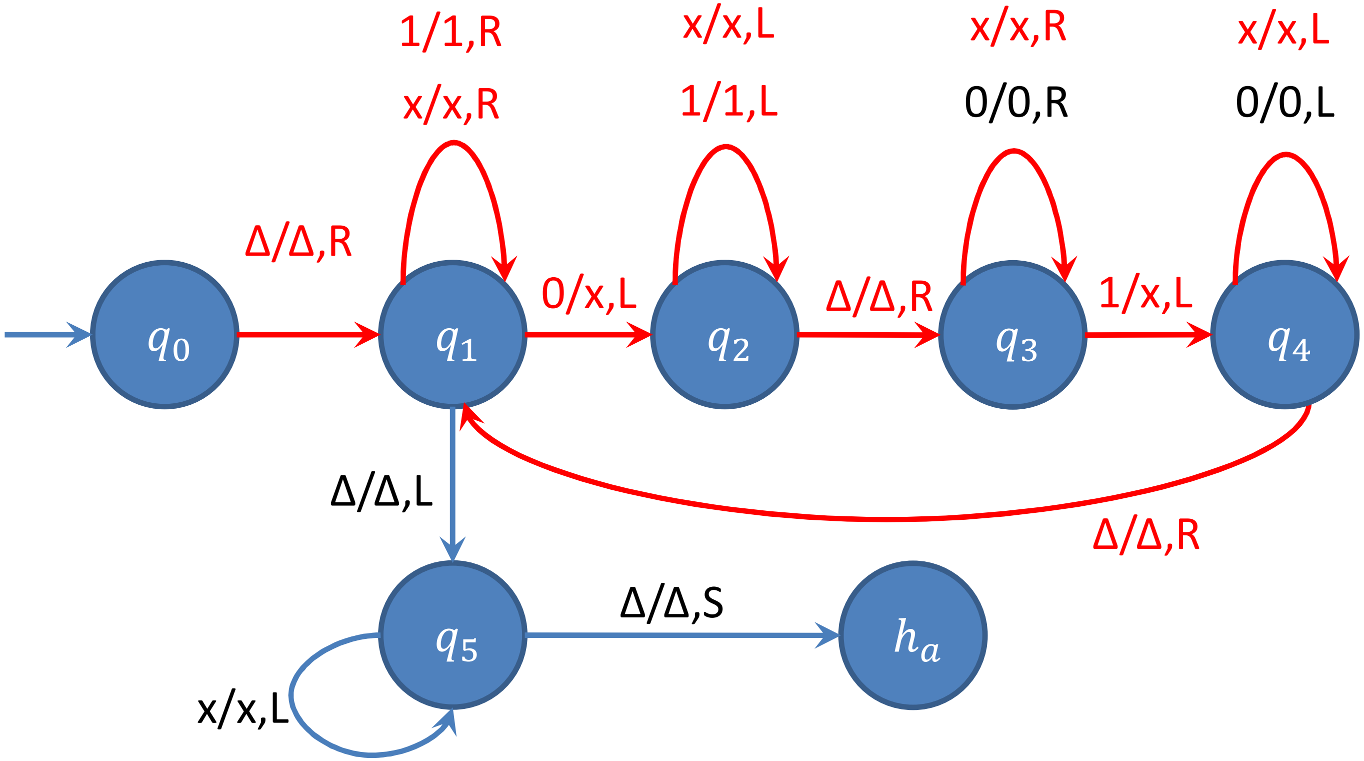
Design a TM for accepting same no of 0's and 1's



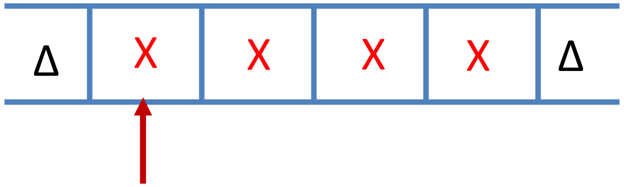


Design a TM for accepting same no of 0's and 1's

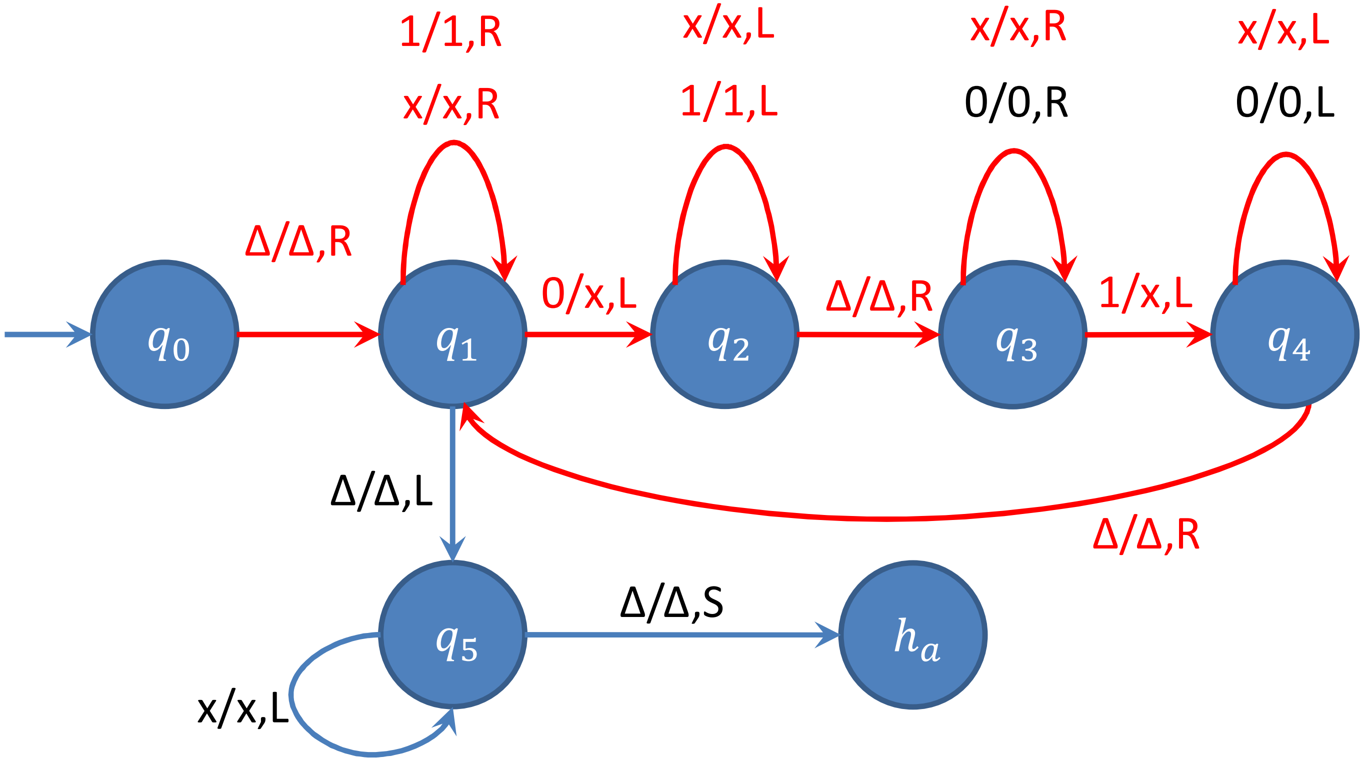




Design a TM for accepting same no of 0’s and 1’s

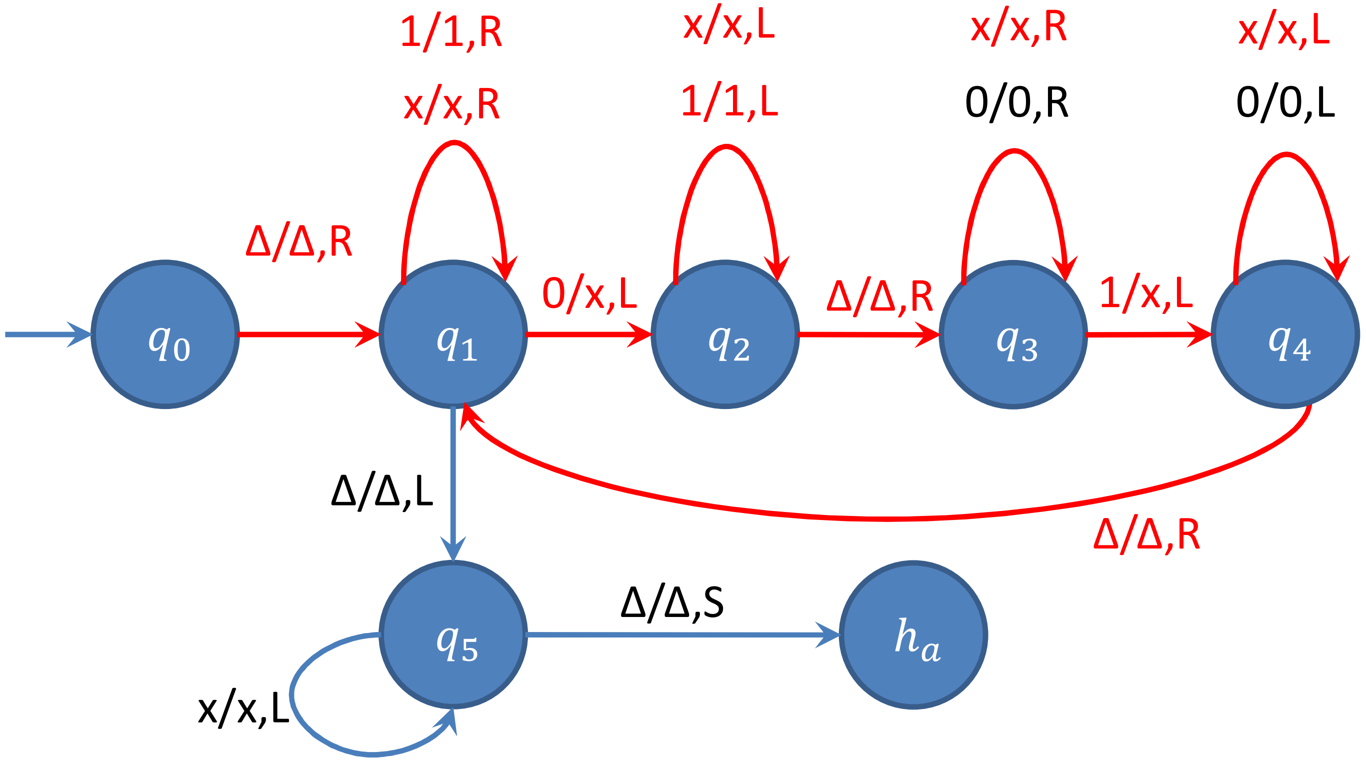


Δ	X	X	X	X	Δ

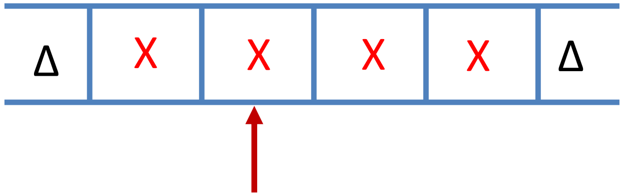


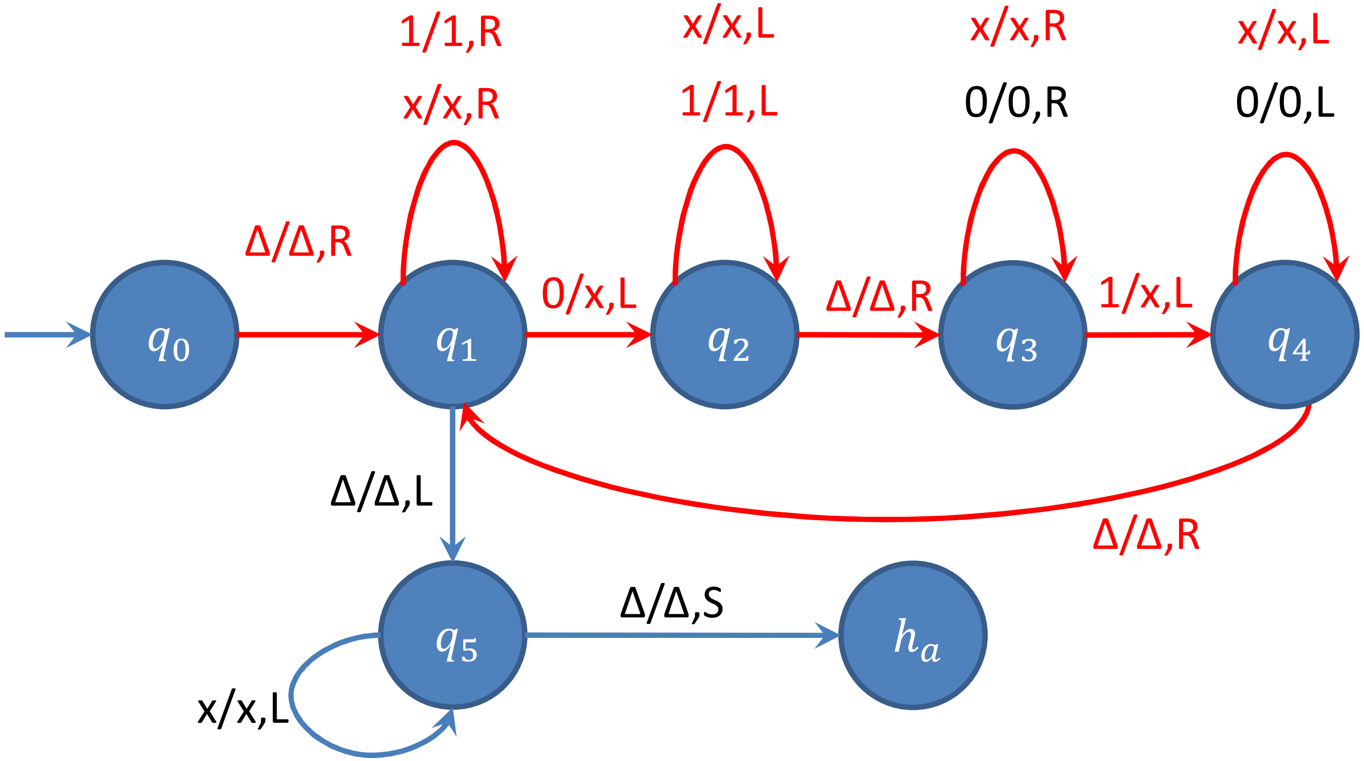
Design a TM for accepting same no of 0's and 1's

Δ	X	X	X	X	Δ

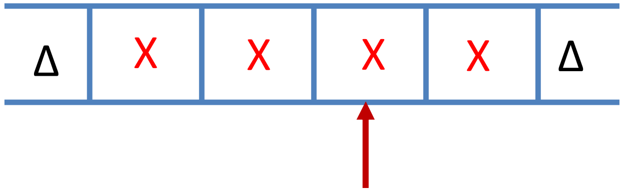


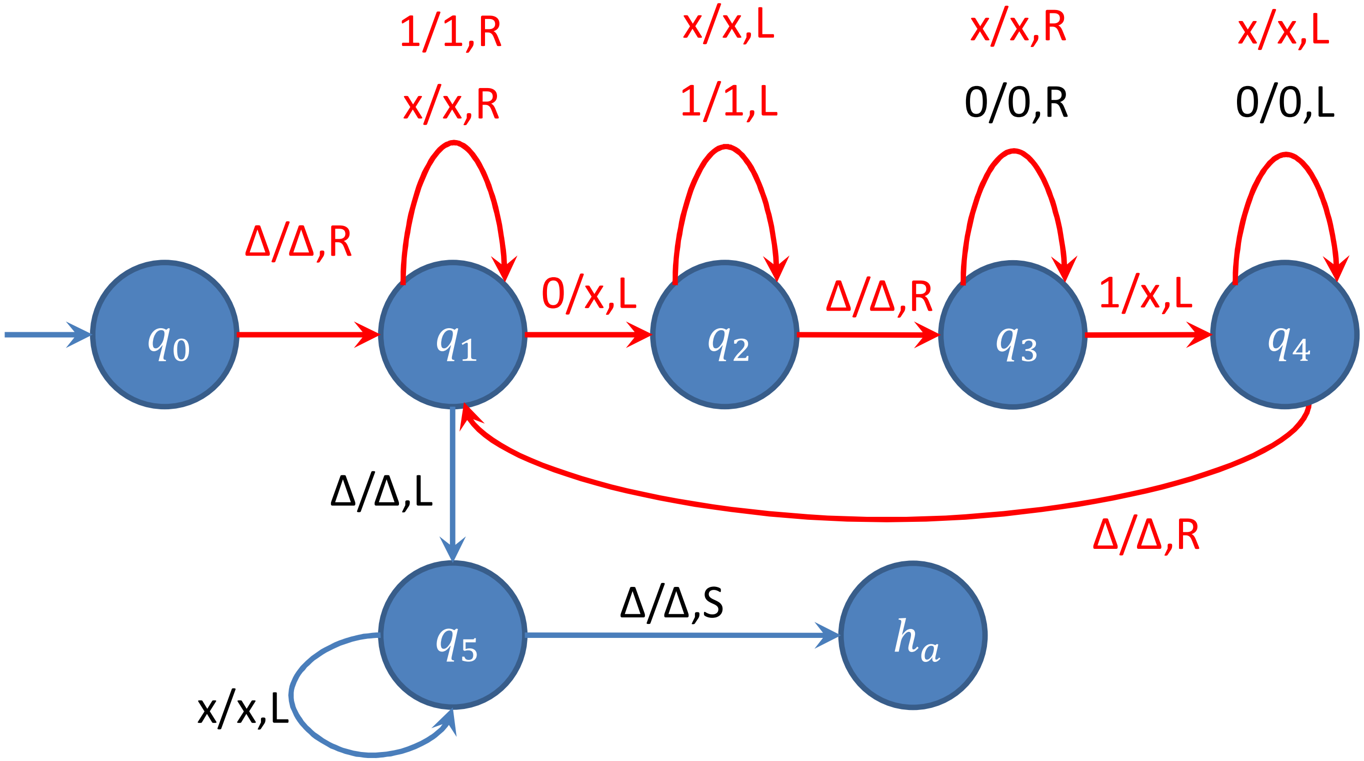
Design a TM for accepting same no of 0’s and 1’s



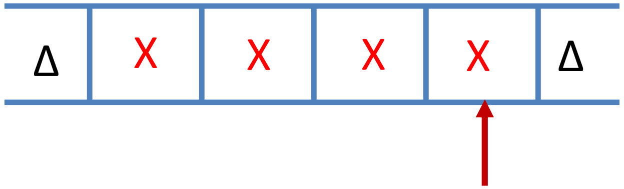


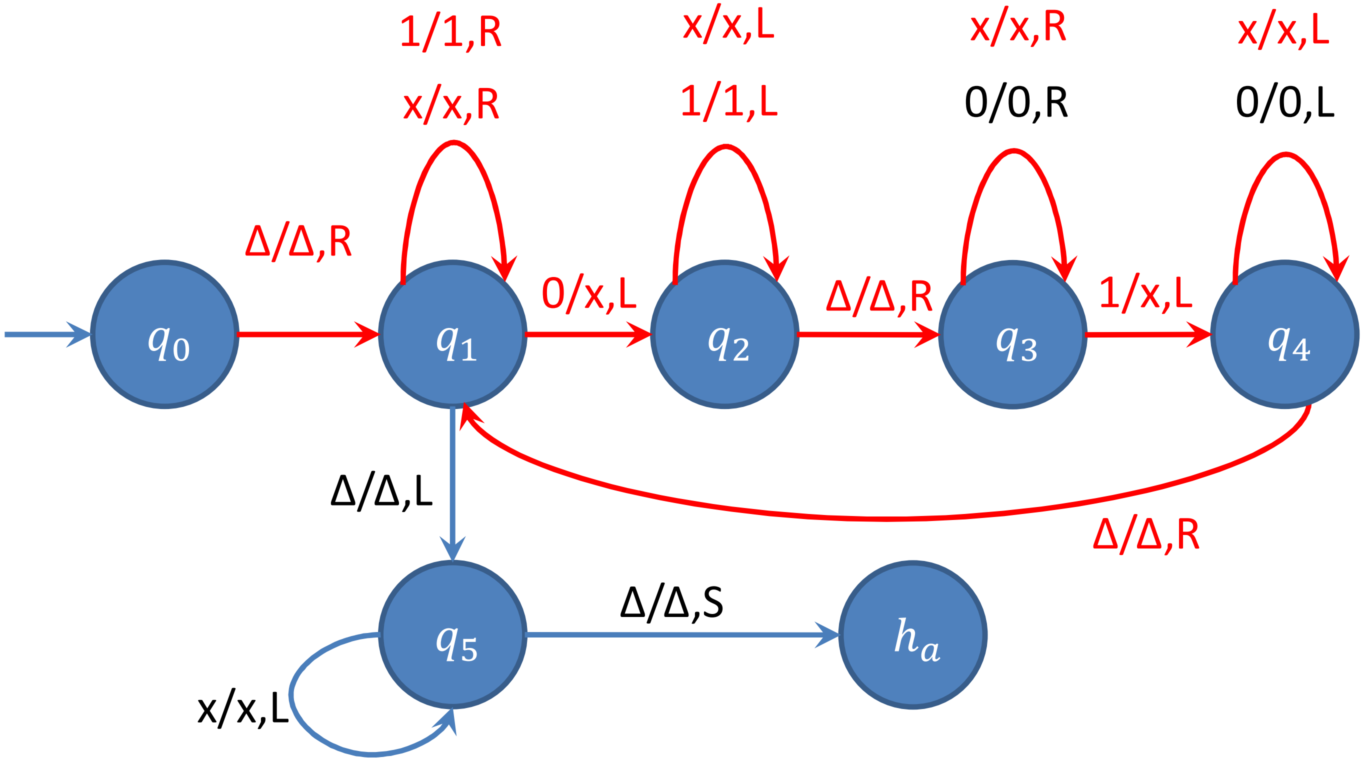
Design a TM for accepting same no of 0's and 1's



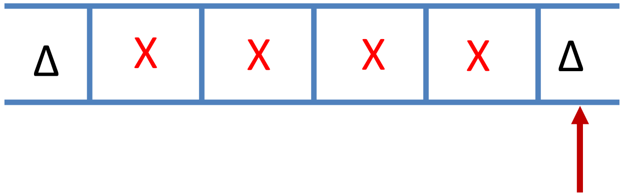


Design a TM for accepting same no of 0’s and 1’s

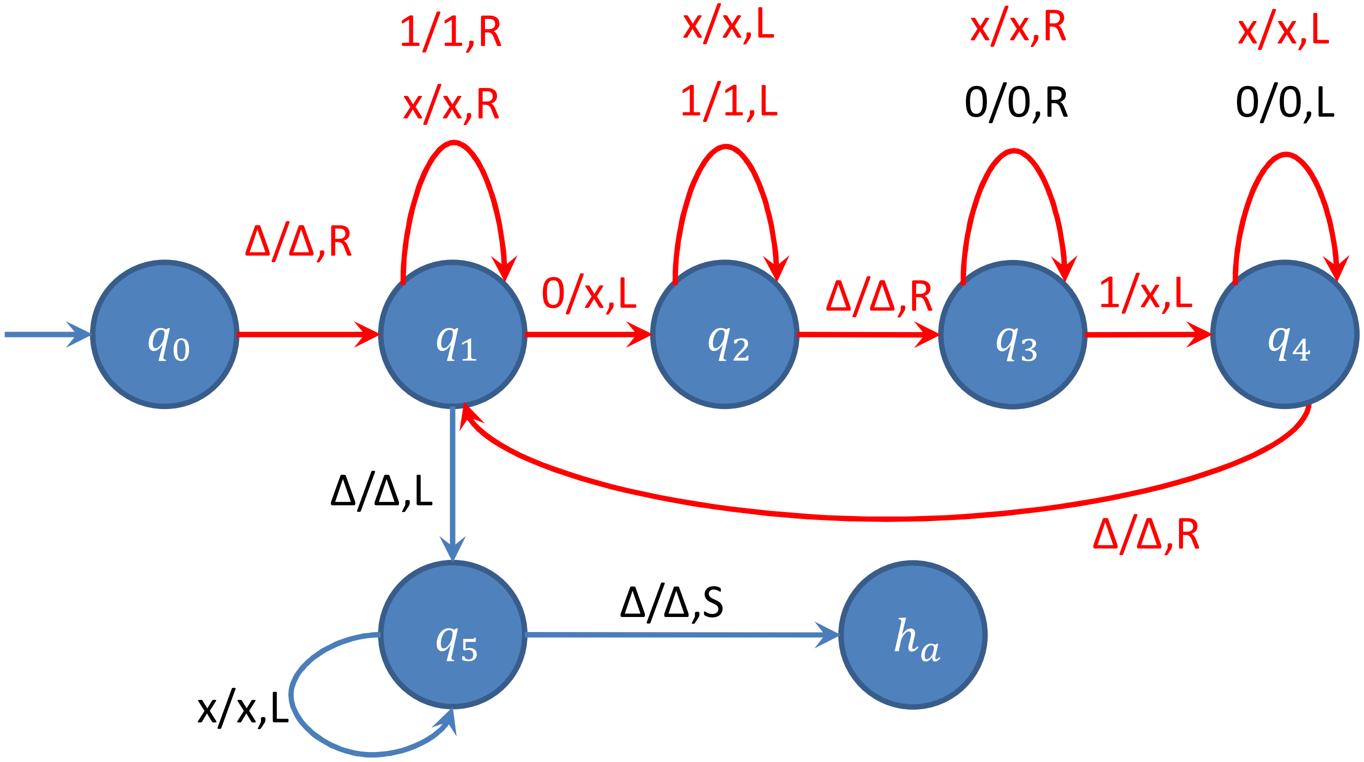




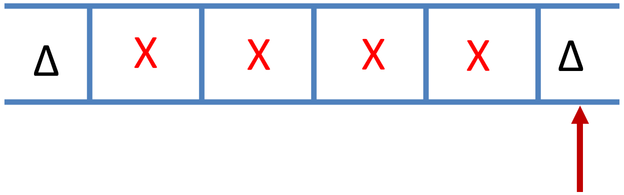
Design a TM for accepting same no of 0’s and 1’s



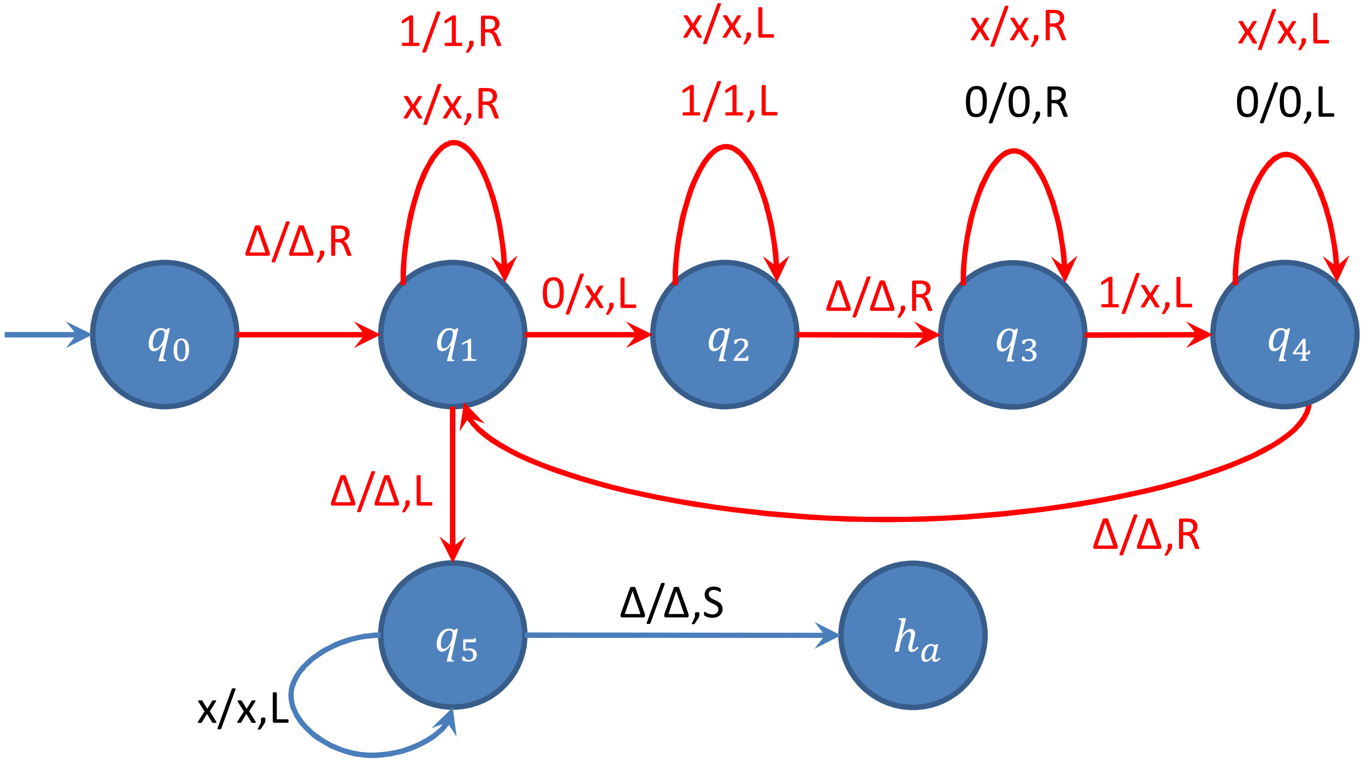
Δ	X	X	X	X	Δ



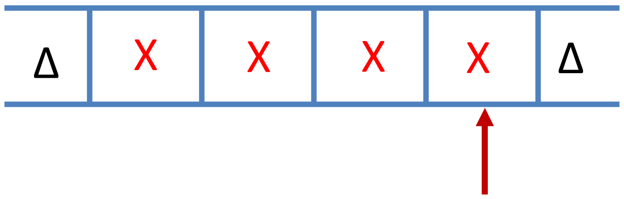
Design a TM for accepting same no of 0’s and 1’s

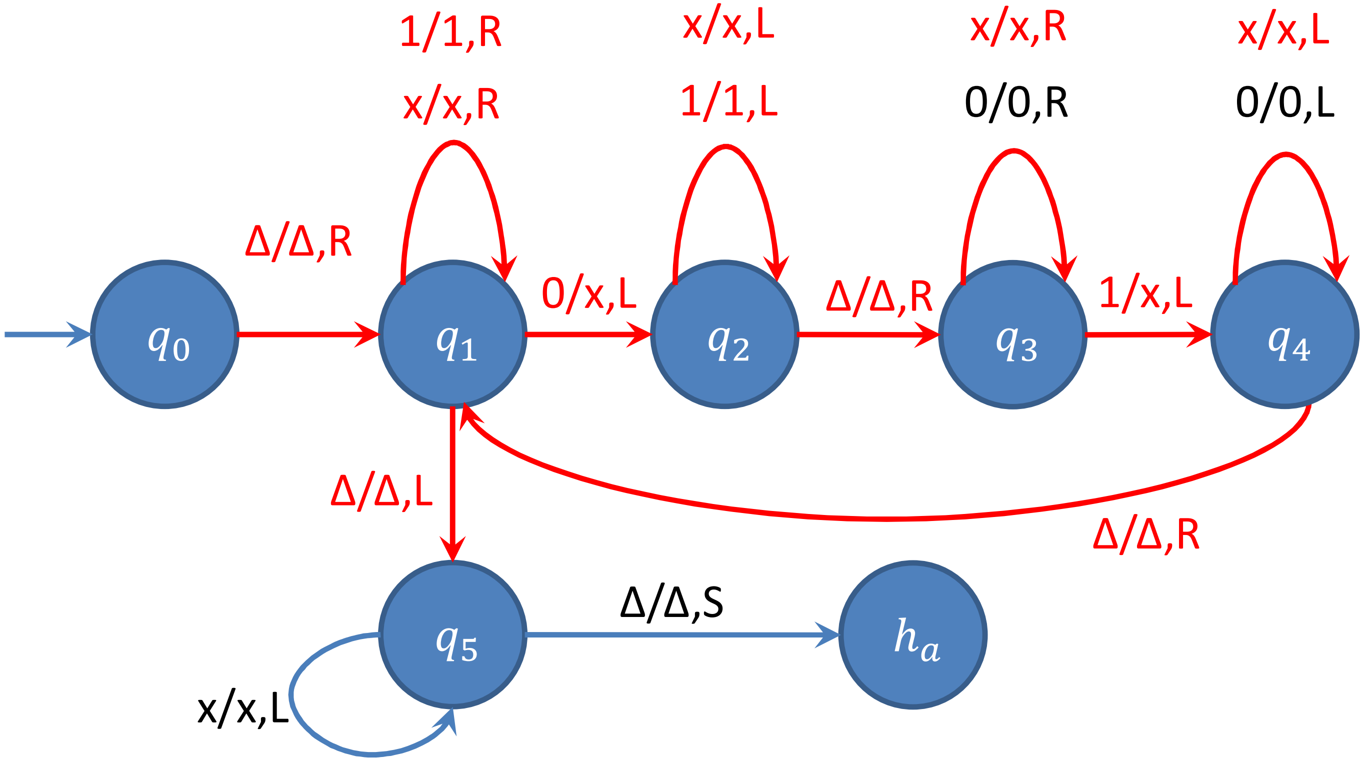


Δ	X	X	X	X	Δ

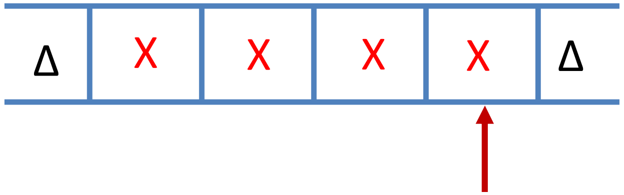


Design a TM for accepting same no of 0’s and 1’s

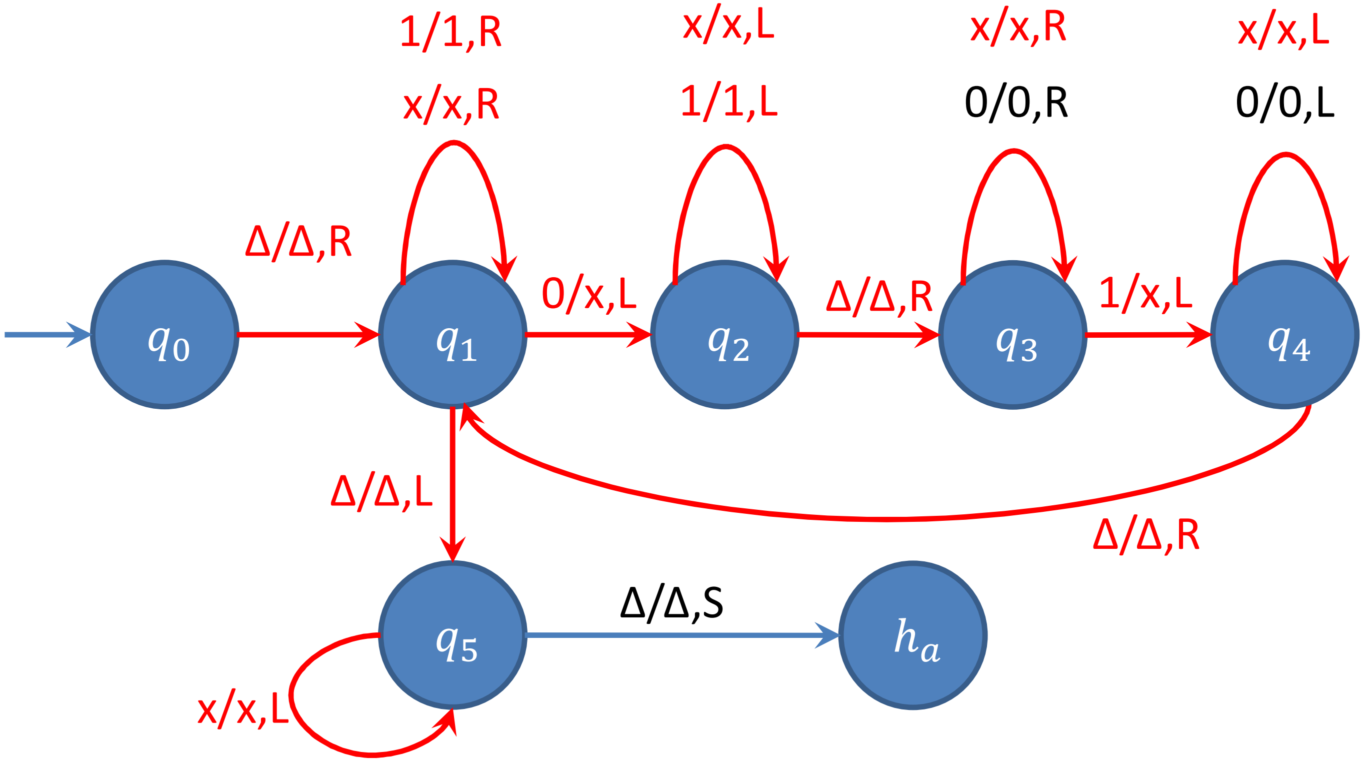




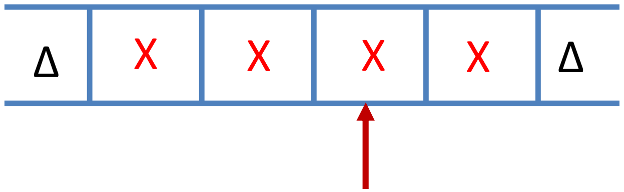
Design a TM for accepting same no of 0’s and 1’s

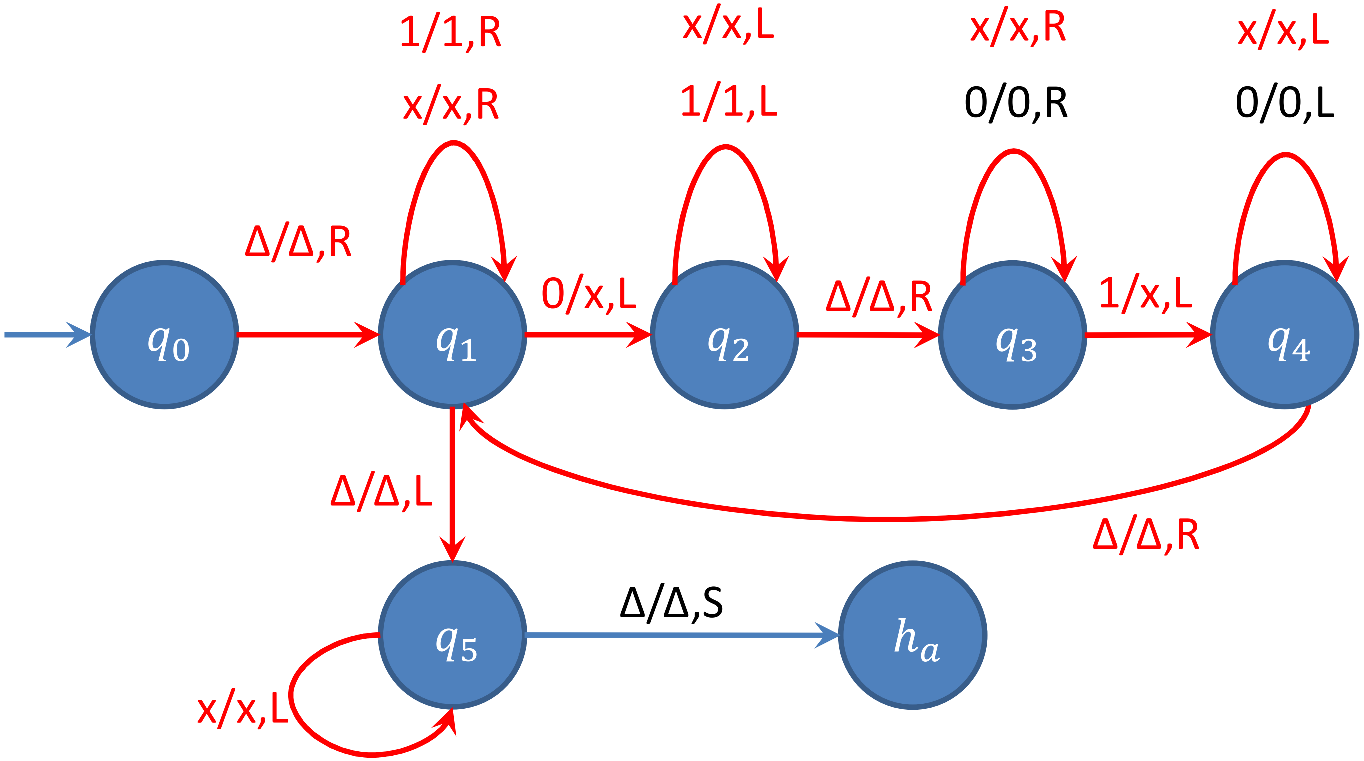


Δ	X	X	X	X	Δ

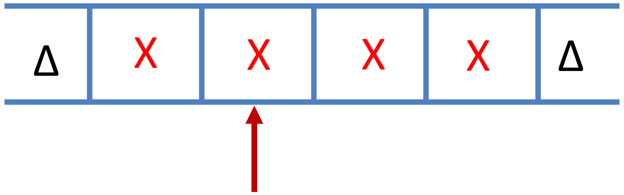


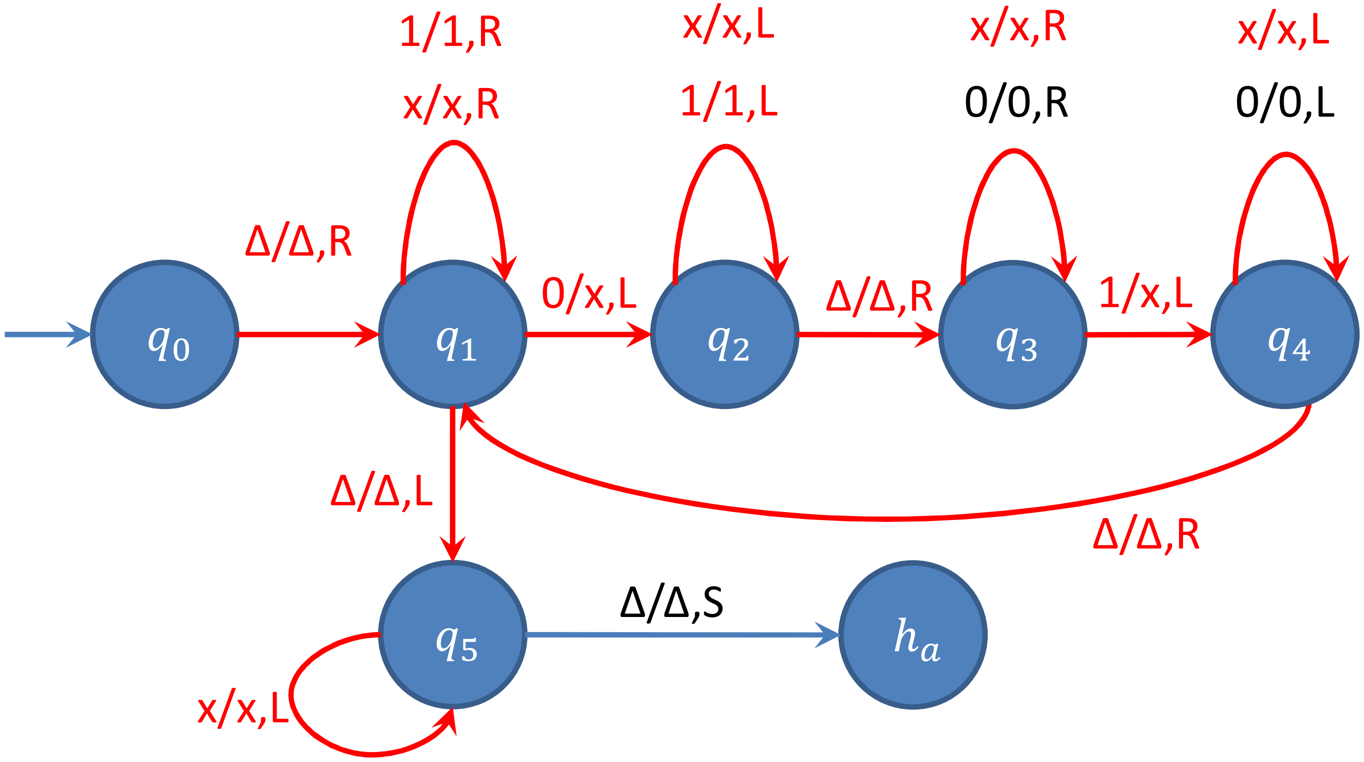
Design a TM for accepting same no of 0’s and 1’s



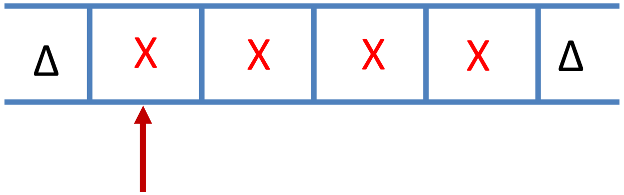


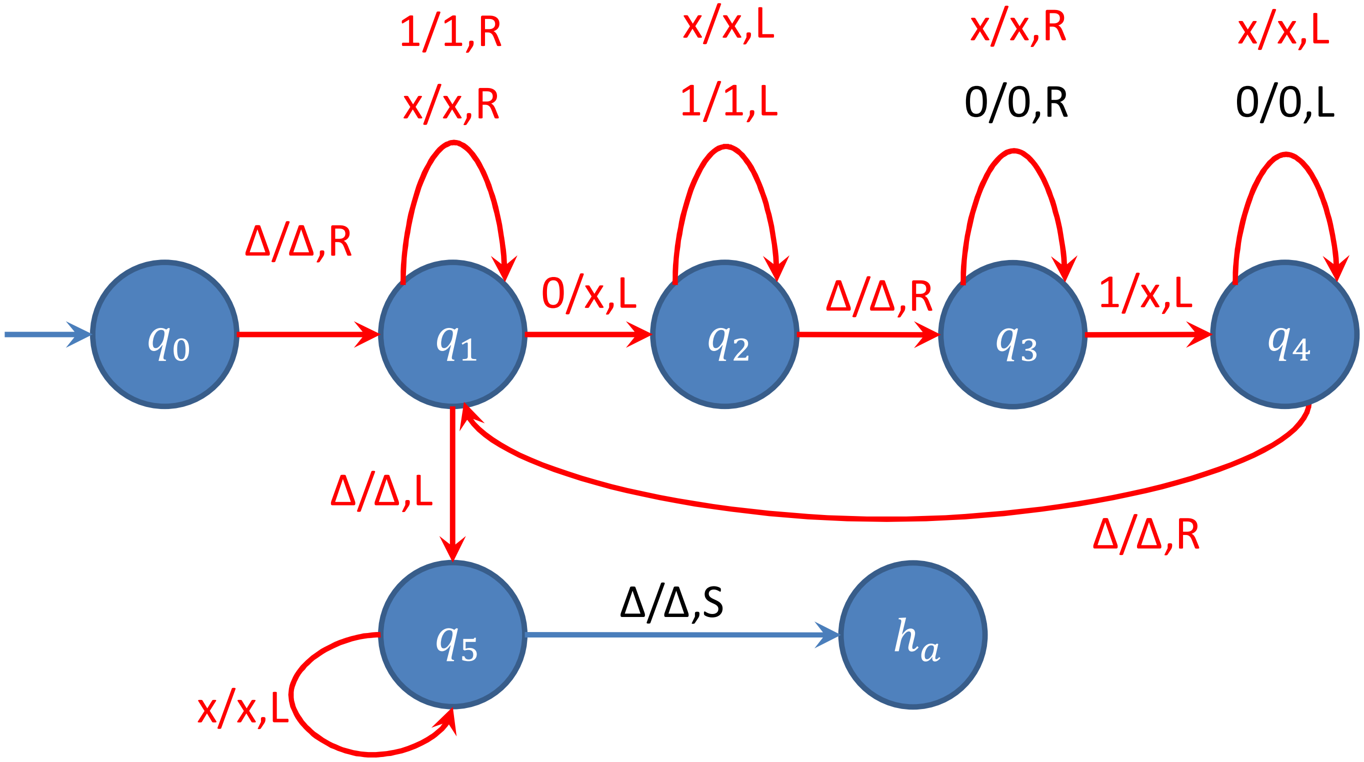
Design a TM for accepting same no of 0’s and 1’s





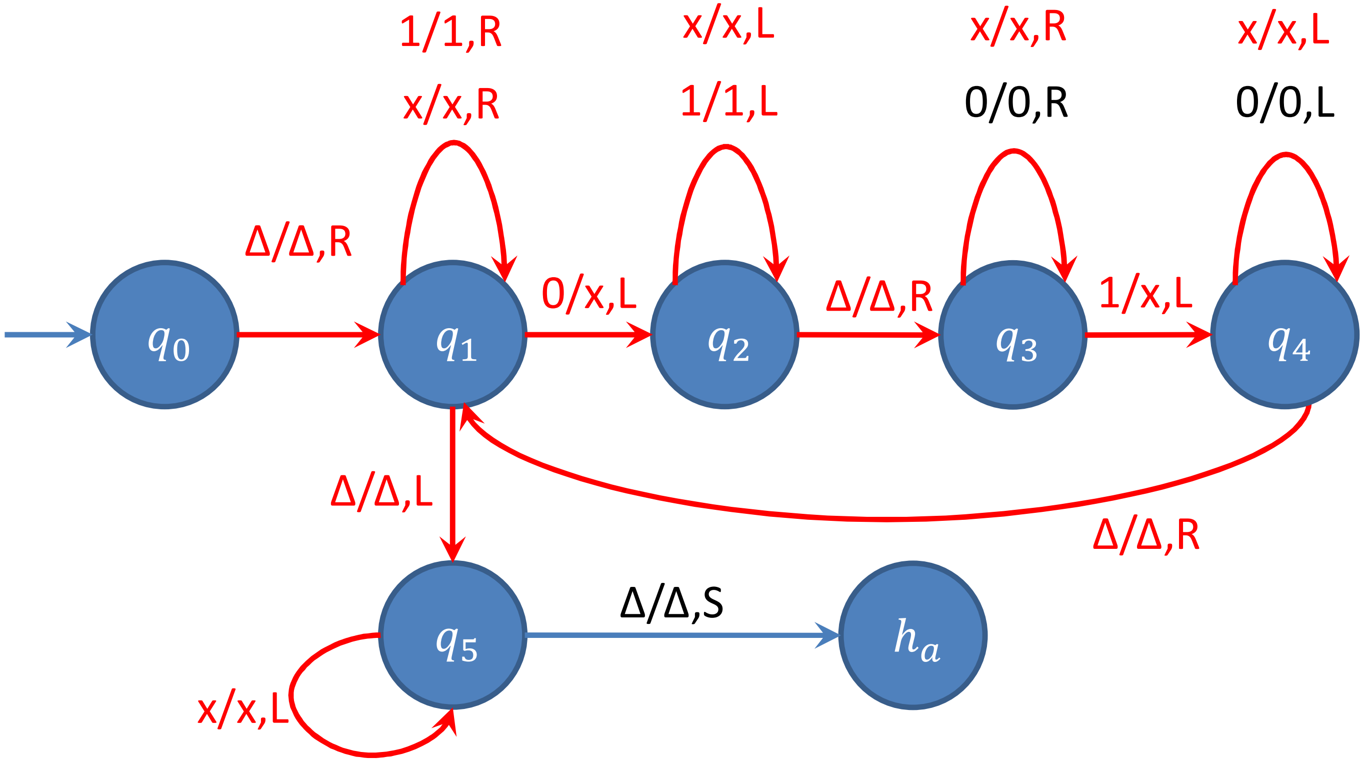
Design a TM for accepting same no of 0’s and 1’s



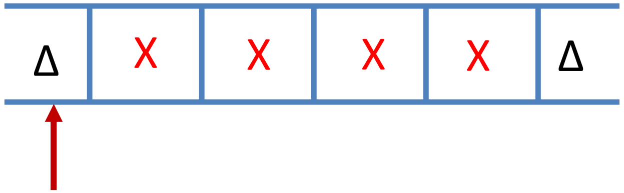


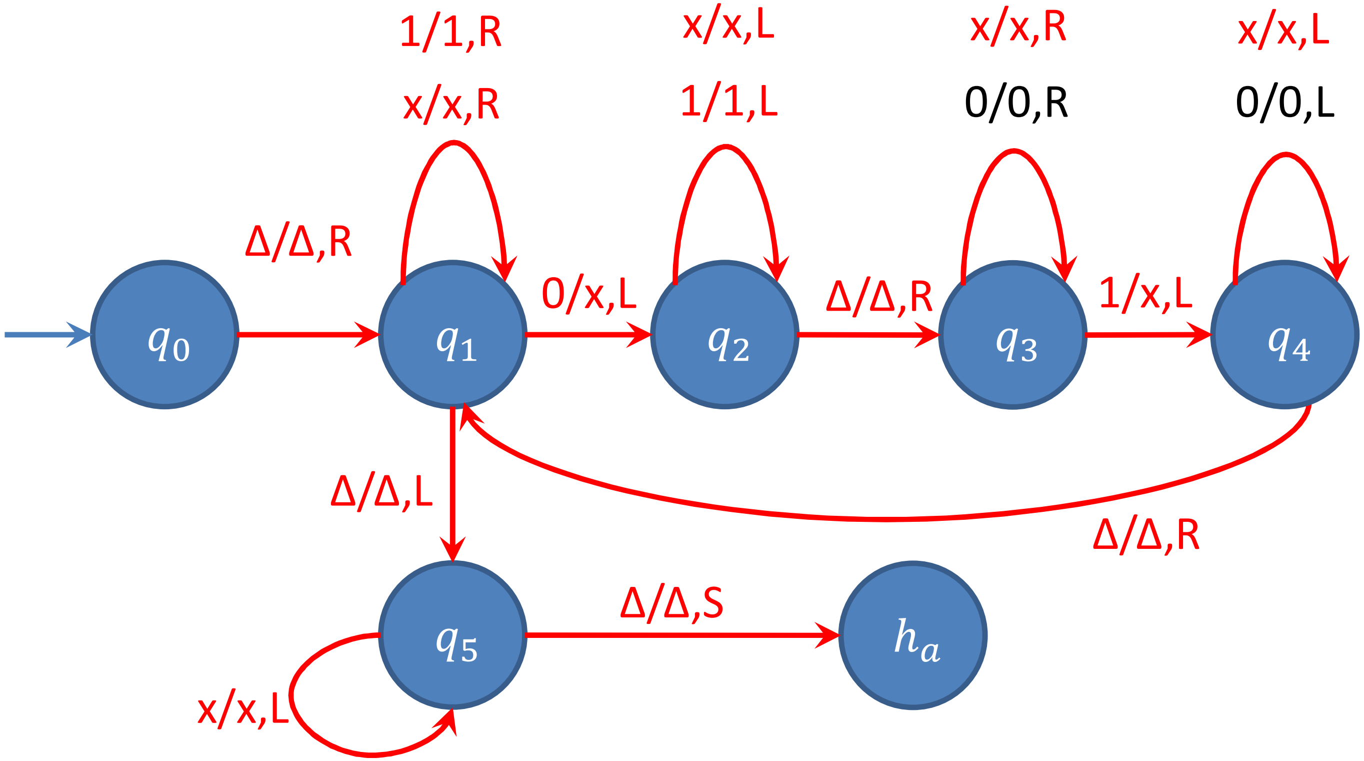
Design a TM for accepting same no of 0's and 1's

Δ	X	X	X	X	Δ



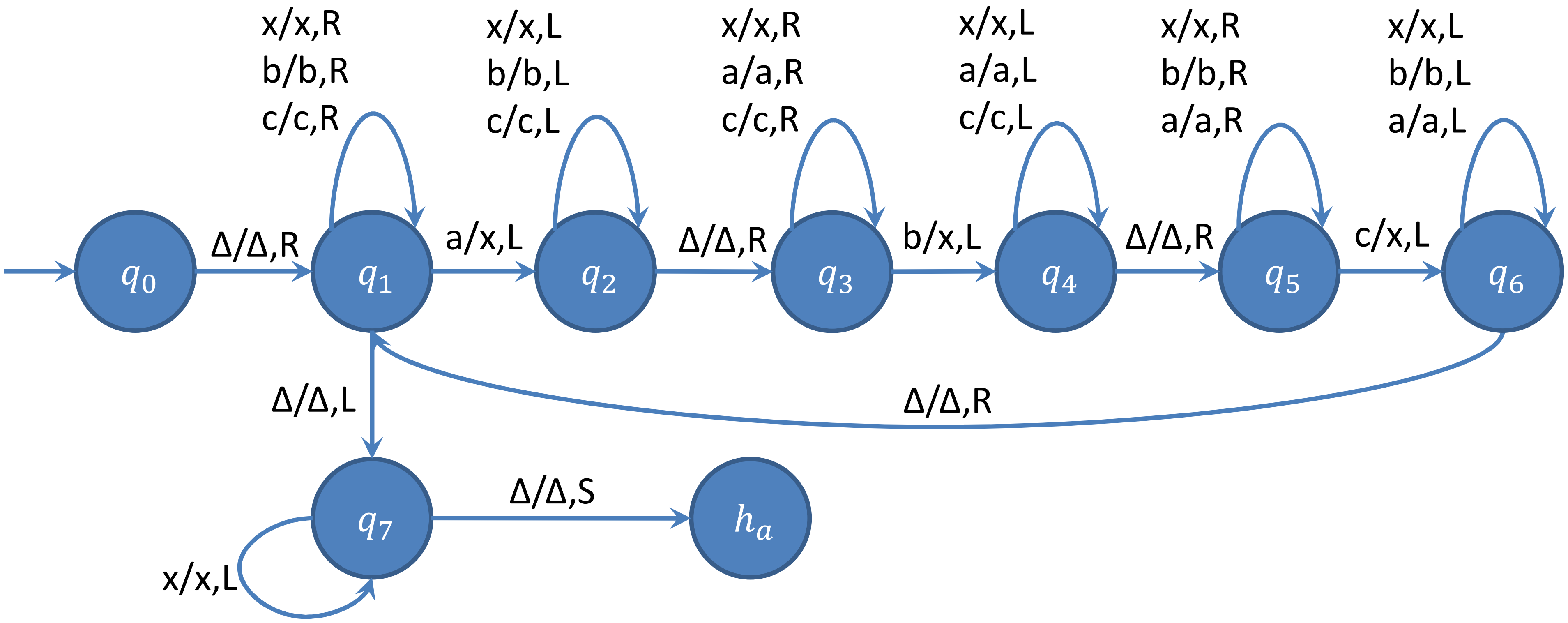
Design a TM for accepting same no of 0's and 1's





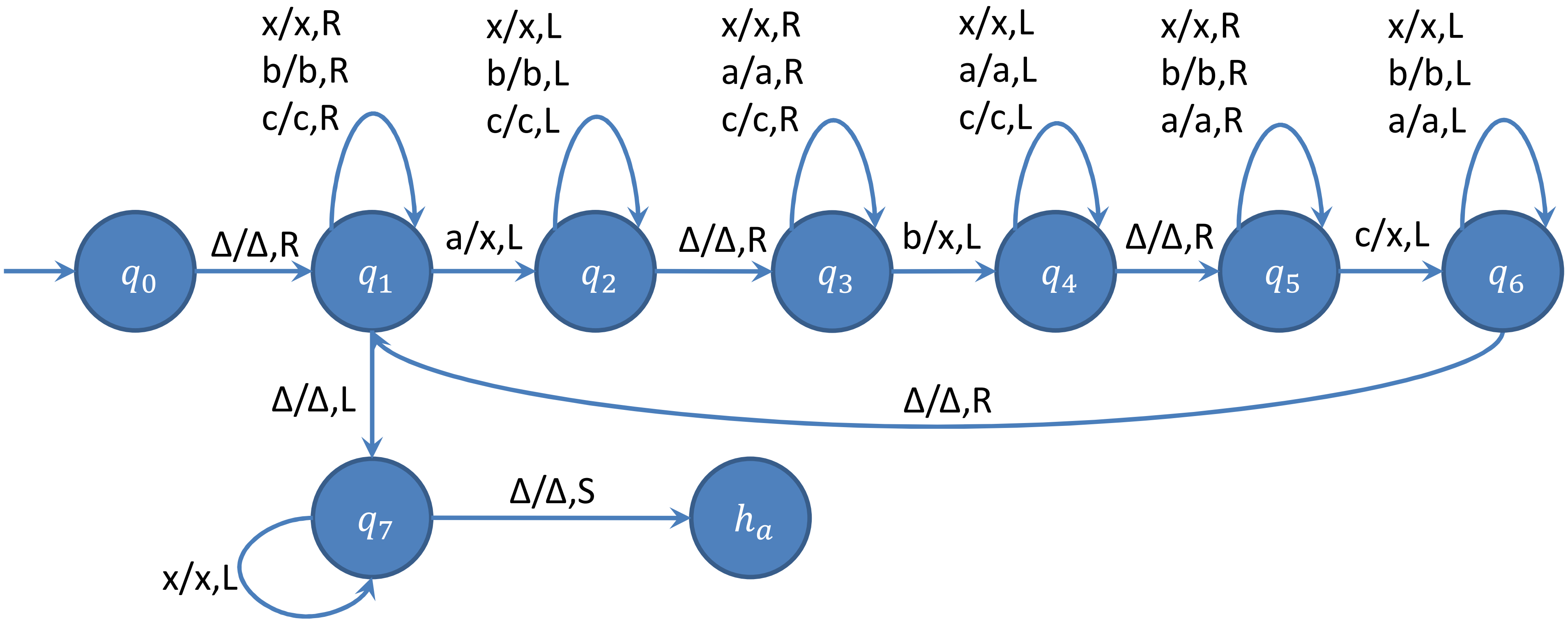
TM Accepting same no of a’s, b’s & c’s

Design a TM for Accepting $\{x \in \{a, b, c\}^* |\ na(x) = nb(x) = nc(x)\ \}$



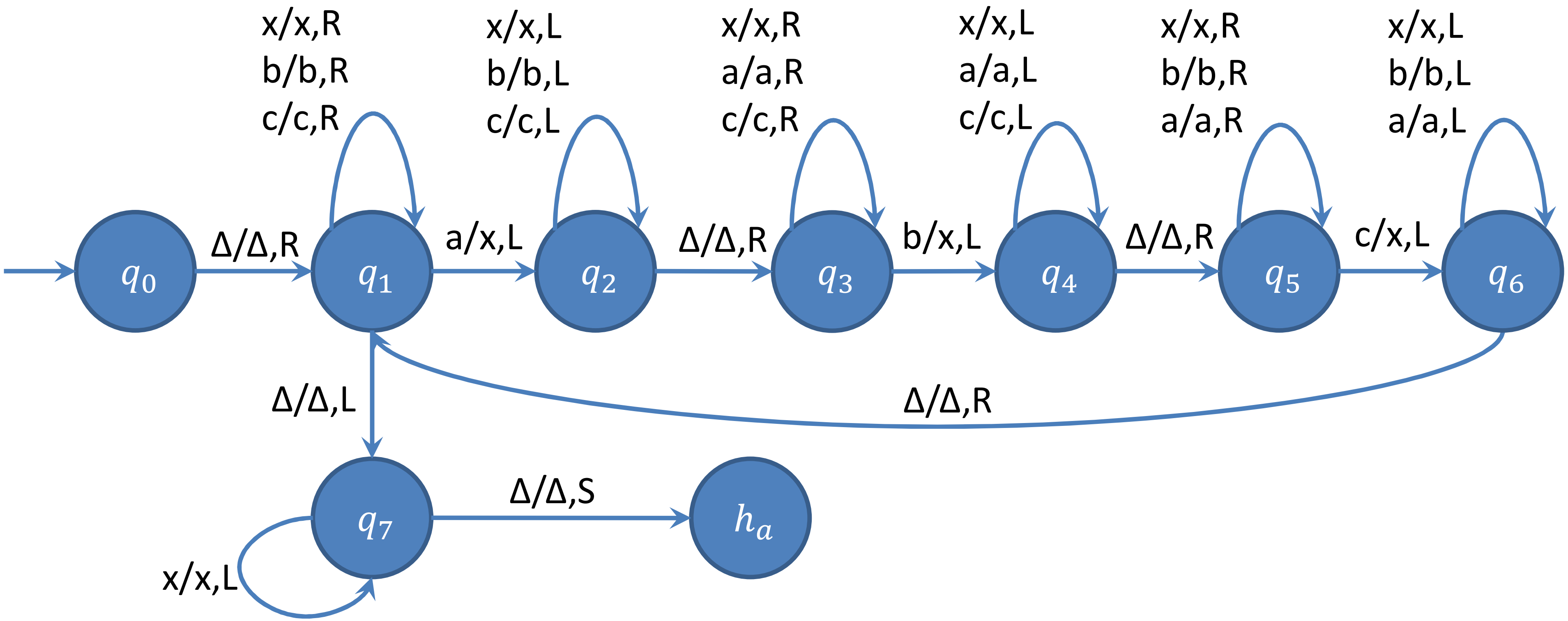
Design a TM for Accepting $\{x \in \{a,b,c\}^* \mid na(x) = nb(x) = nc(x)\}$

Δ	b	a	c	c	a	b	Δ



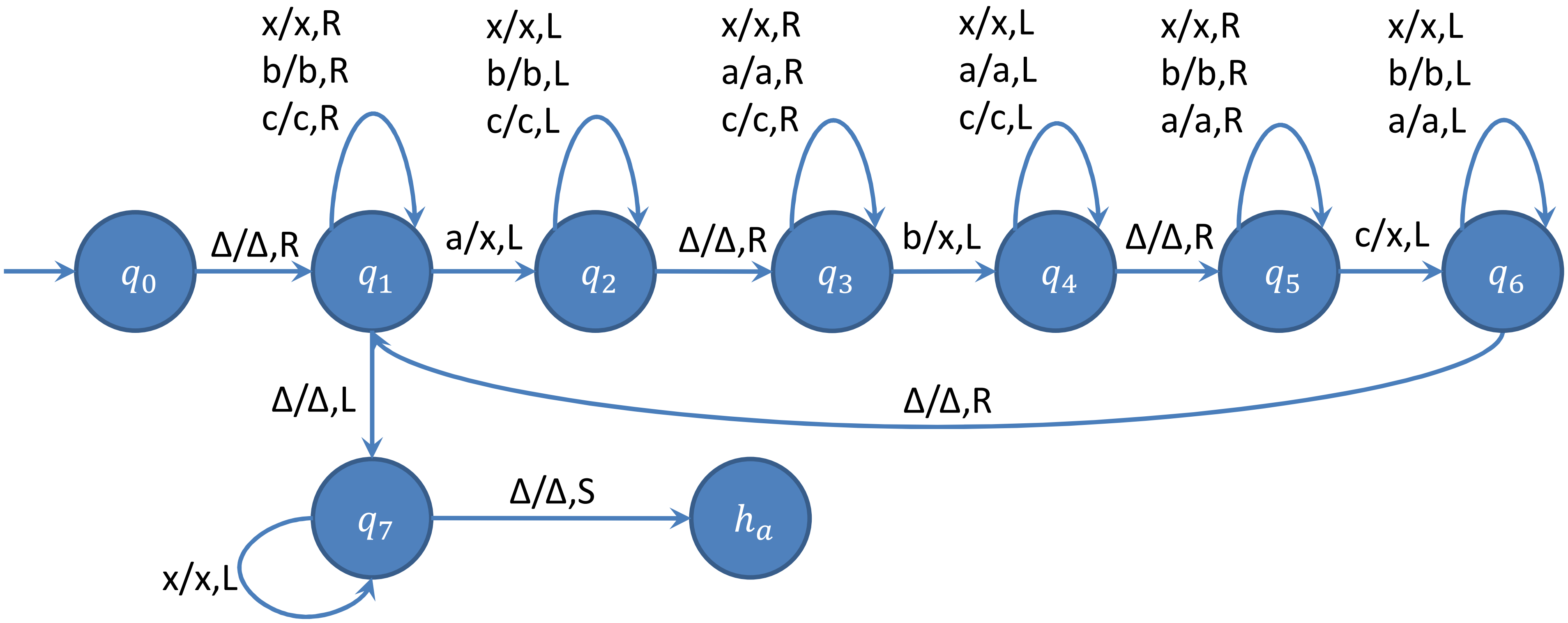
Design a TM for Accepting $\{x \ \in \{a, b, c\}^* |\ na(x) = nb(x) = nc(x)\ \}$

Δ	b	X	c	c	a	b	Δ



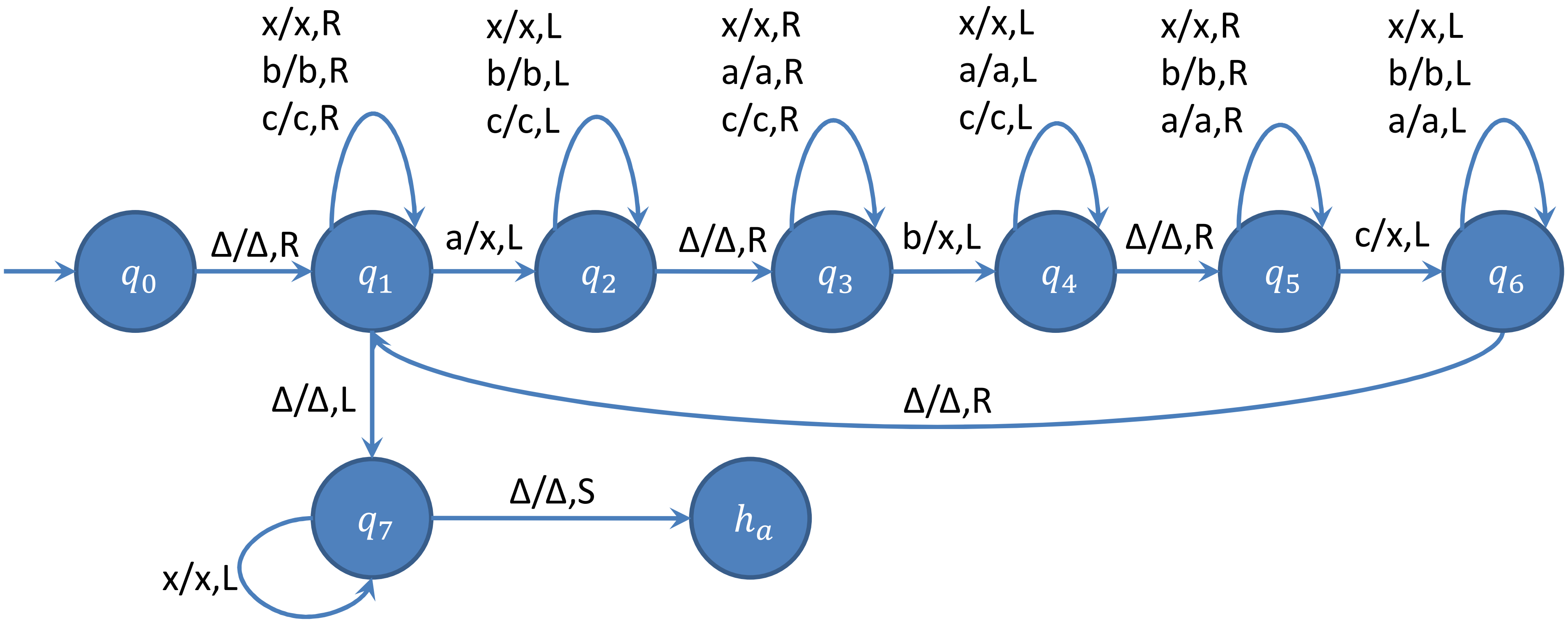
Design a TM for Accepting $\{x \in \{a, b, c\}^* \mid na(x) = nb(x) = nc(x)\}$

Δ	X	X	c	c	a	b	Δ



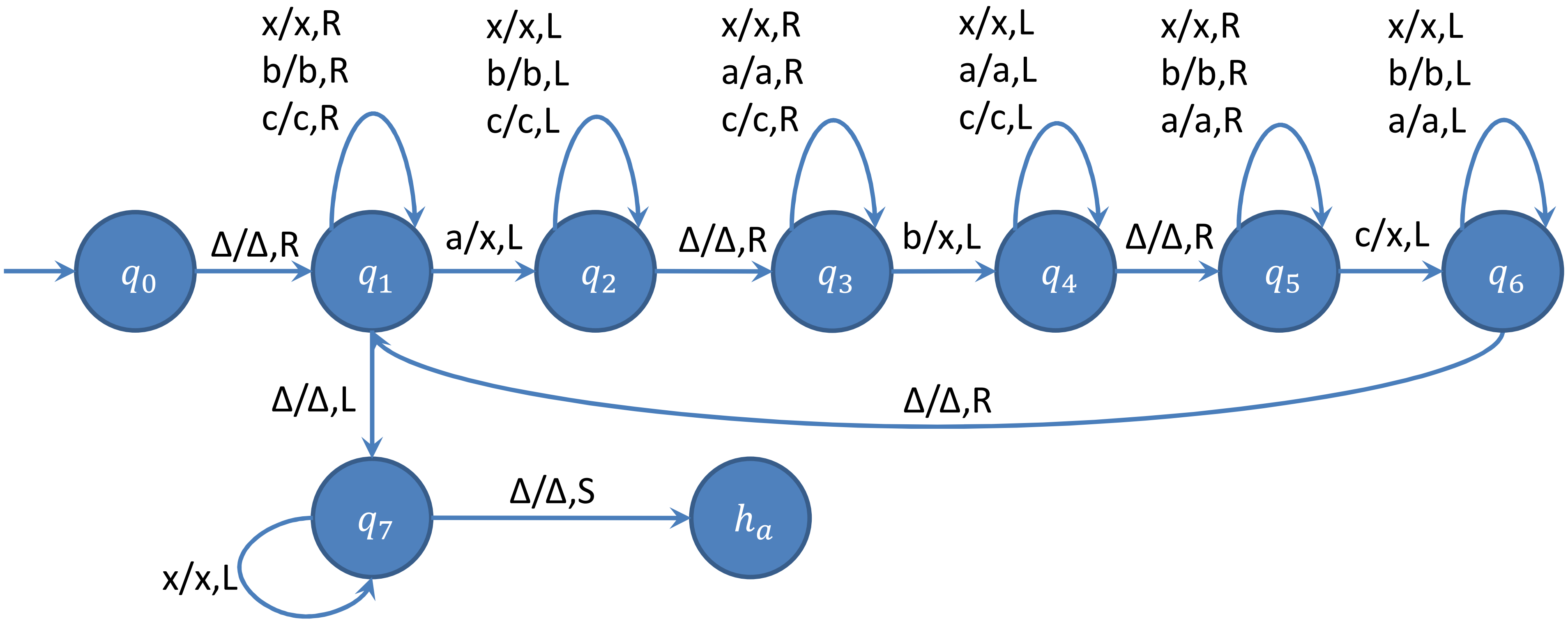
Design a TM for Accepting $\{x \in \{a, b, c\}^* \mid na(x) = nb(x) = nc(x)\}$

Δ	X	X	X	c	a	b	Δ



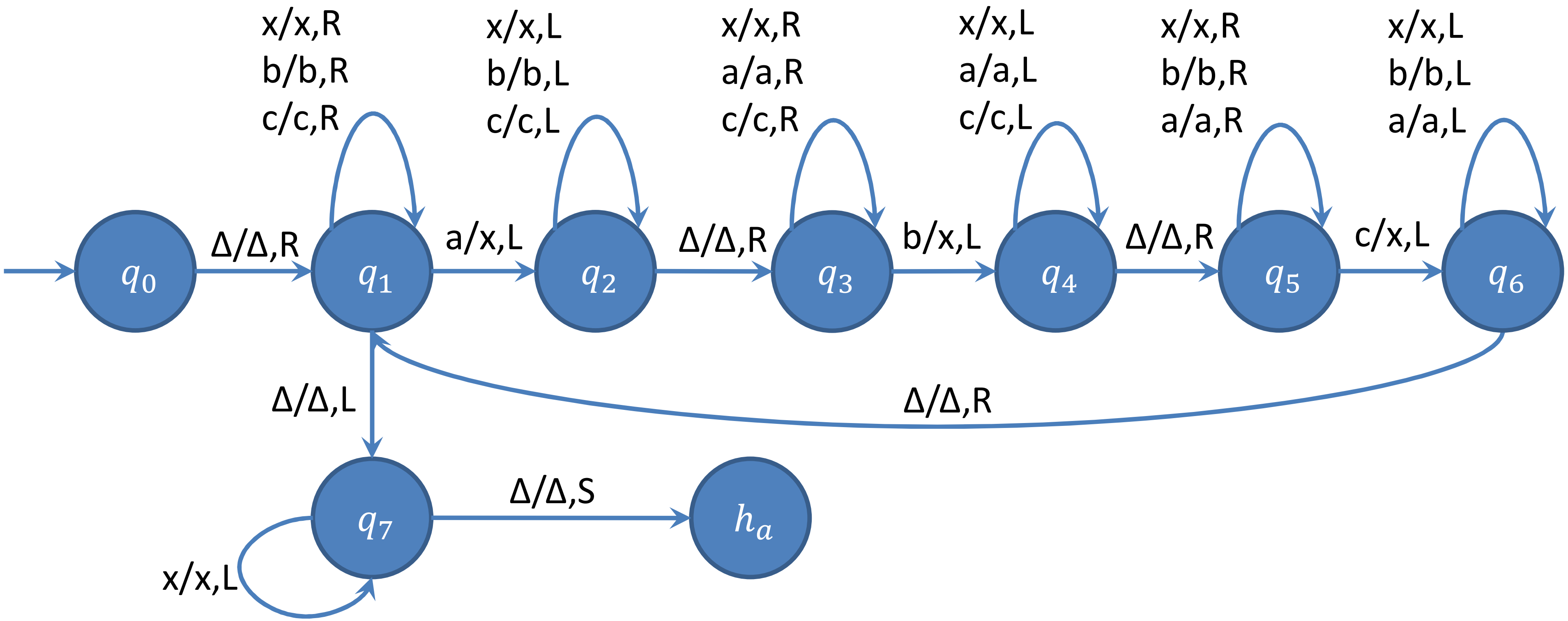
Design a TM for Accepting $\{x \in \{a, b, c\}^* \mid na(x) = nb(x) = nc(x)\}$

Δ	X	X	X	c	X	b	Δ



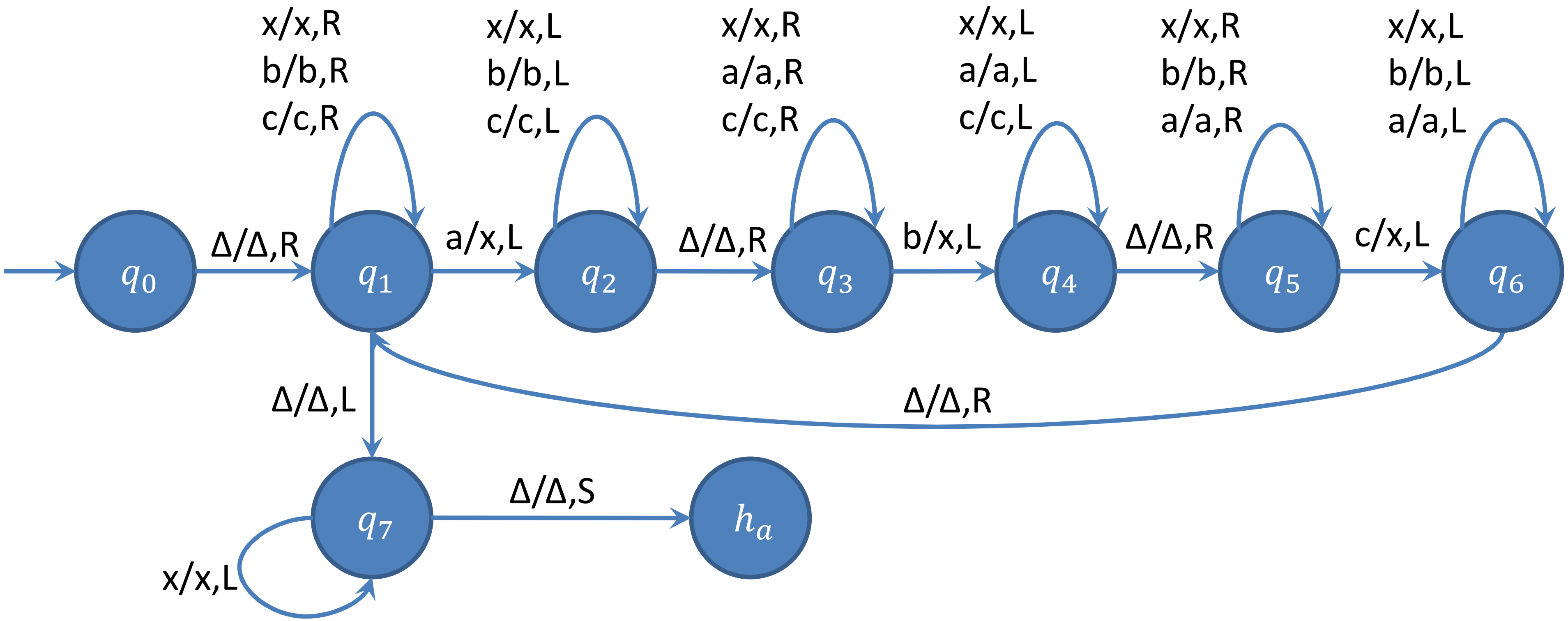
Design a TM for Accepting $\{x \ \in \{a, b, c\}^* |\ na(x) = nb(x) = nc(x)\ \}$

Δ	X	X	X	c	X	X	Δ



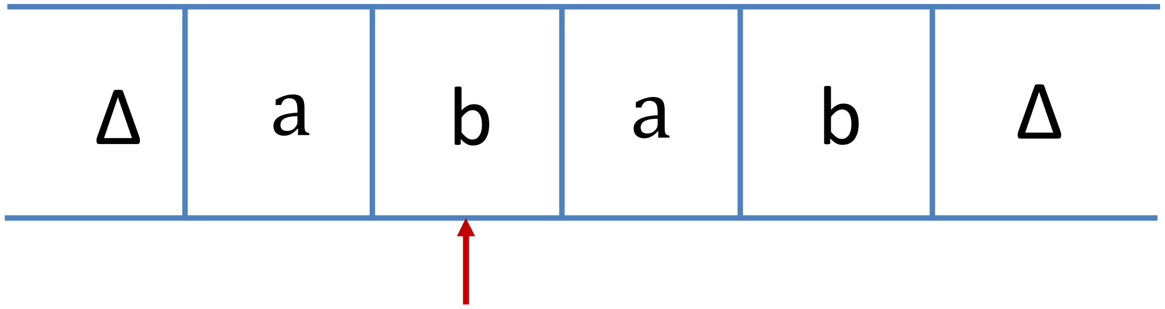
Design a TM for Accepting $\{x \in \{a, b, c\}^* \mid na(x) = nb(x) = nc(x)\}$

Δ	X	X	X	X	X	X	Δ



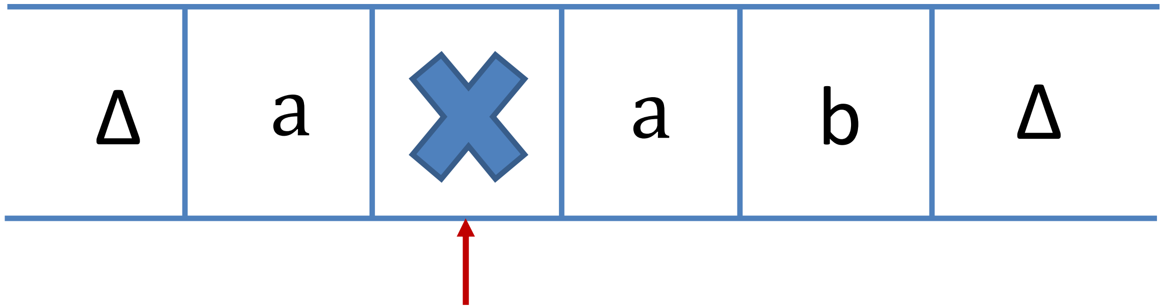
Turing machine to delete a symbol

Design a Turing machine to delete a symbol



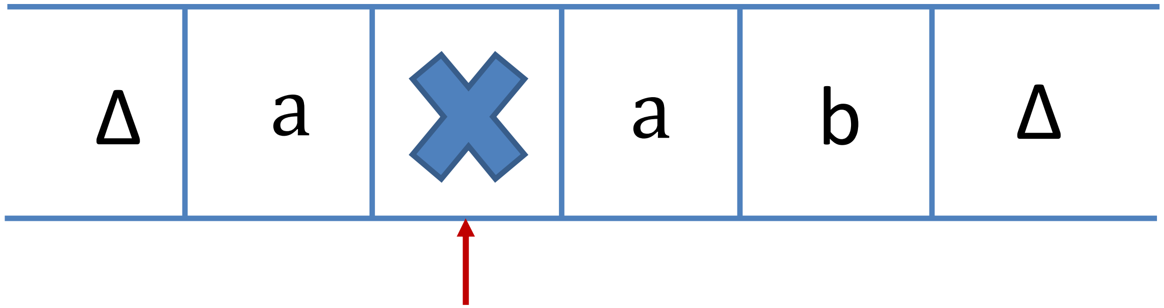
Δ	a	b	a	b	Δ

Design a Turing machine to delete a symbol



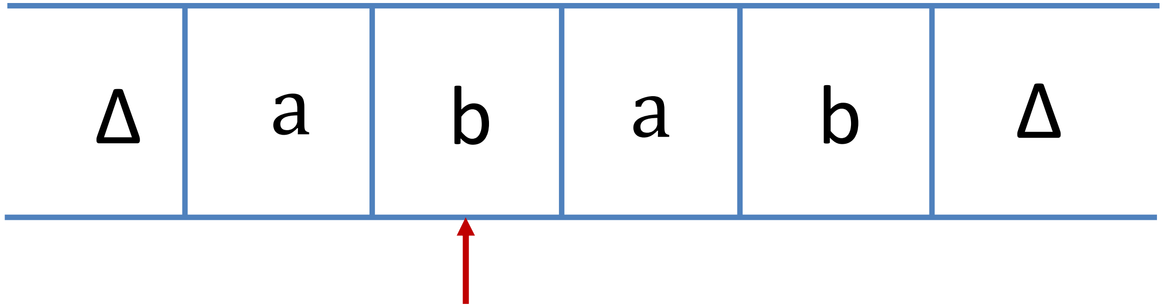
Design a Turing machine to delete a symbol

ΔaabΔ



Design a Turing machine to delete a symbol

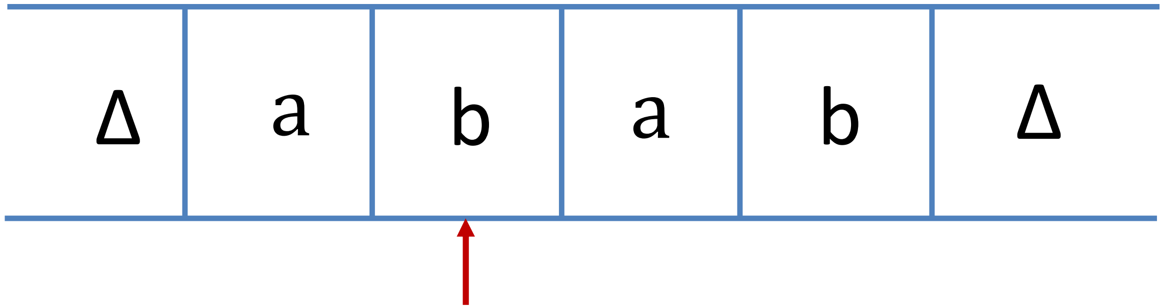
ΔaabΔ



Δ	a	b	a	b	Δ

Design a Turing machine to delete a symbol

ΔaabΔ

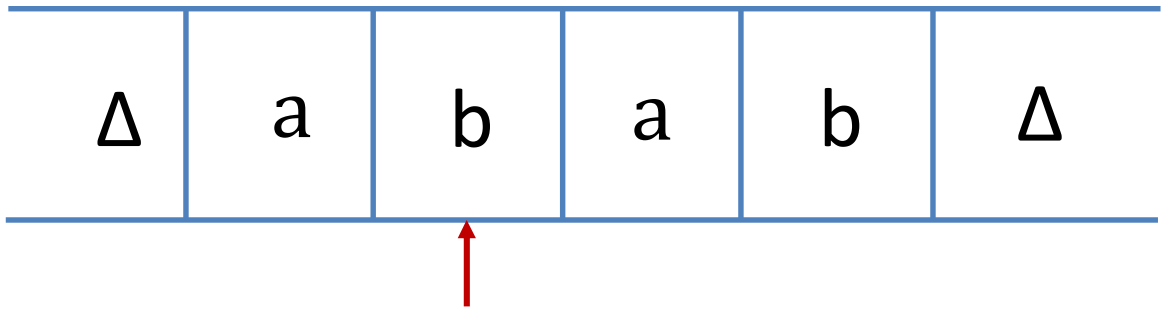


Δ	a	b	a	b	Δ

Logic:

Design a Turing machine to delete a symbol

ΔaabΔ



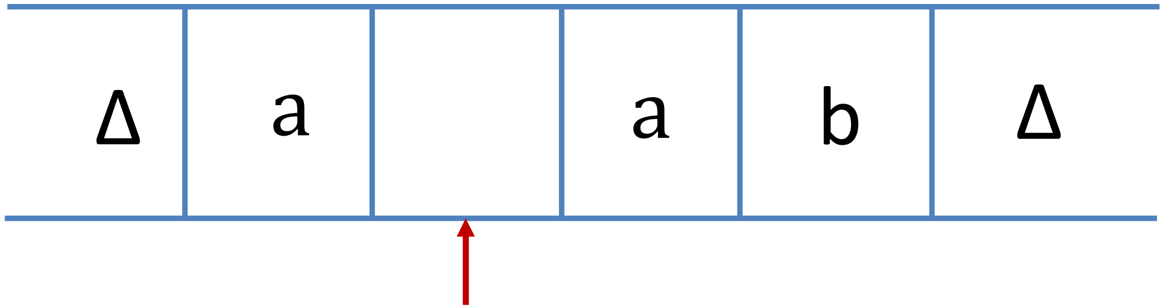
Δ	a	b	a	b	Δ

Logic:

1. Replace the symbol you want to delete by Δ.

Design a Turing machine to delete a symbol

ΔaabΔ

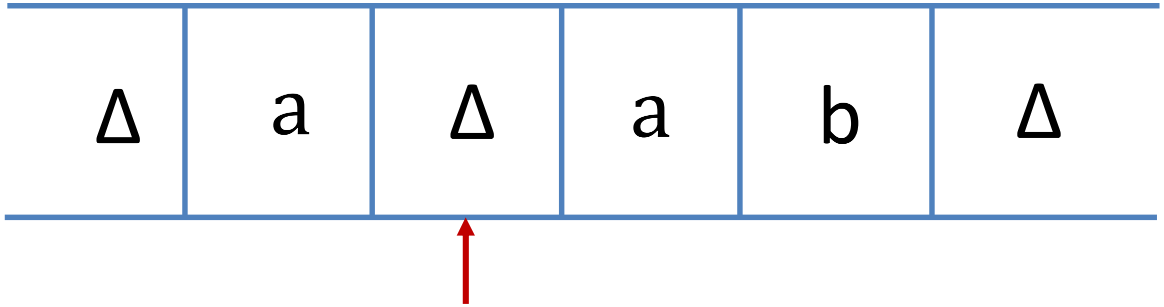


Logic:

1. Replace the symbol you want to delete by Δ.

Design a Turing machine to delete a symbol

ΔaabΔ

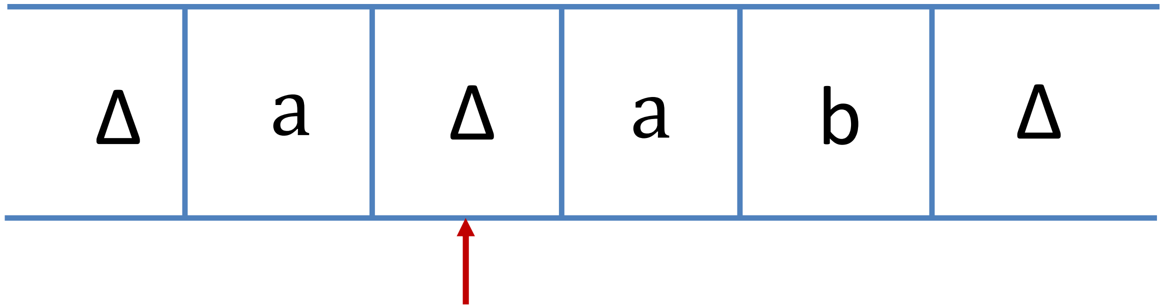


Logic:

1. Replace the symbol you want to delete by Δ.

Design a Turing machine to delete a symbol

ΔaabΔ



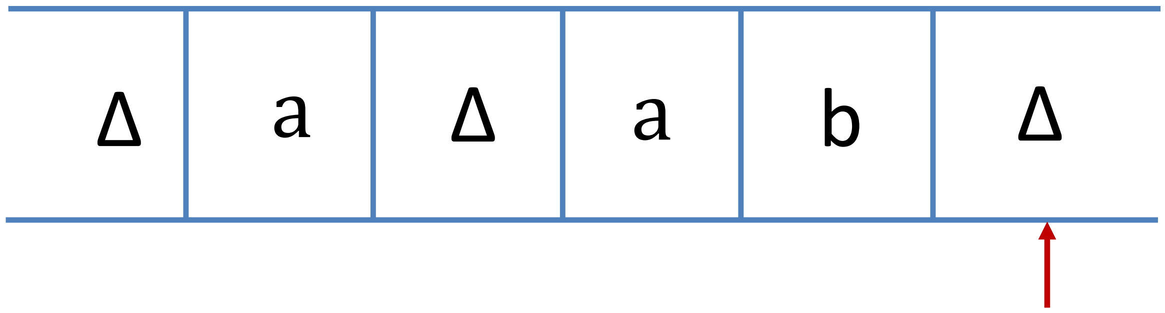
Δ	a	Δ	a	b	Δ

Logic:

1. Replace the symbol you want to delete by Δ.
2. Keep moving to the Right until you encounter Δ.

Design a Turing machine to delete a symbol

ΔaabΔ



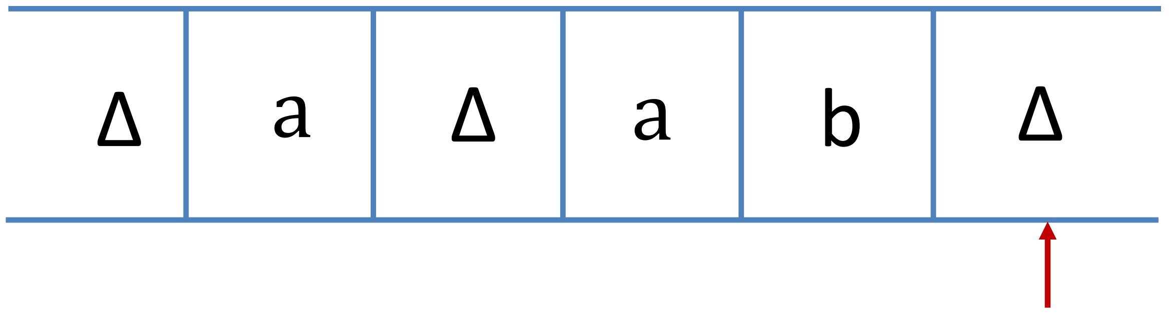
Δ	a	Δ	a	b	Δ

Logic:

1. Replace the symbol you want to delete by Δ.
2. Keep moving to the Right until you encounter Δ.

Design a Turing machine to delete a symbol

ΔaabΔ



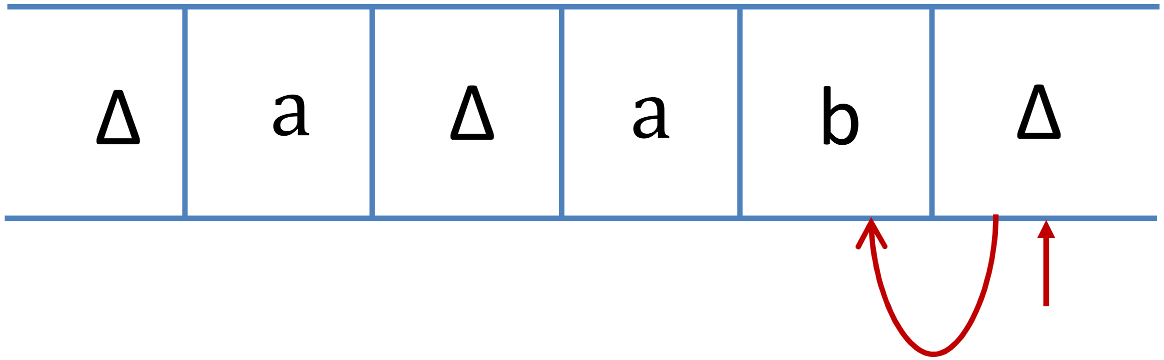
Δ	a	Δ	a	b	Δ

Logic:

1. Replace the symbol you want to delete by Δ.
2. Keep moving to the Right until you encounter Δ.
3. One by one Keep shifting every symbol one step to left until you encounter Δ.

Design a Turing machine to delete a symbol

ΔaabΔ

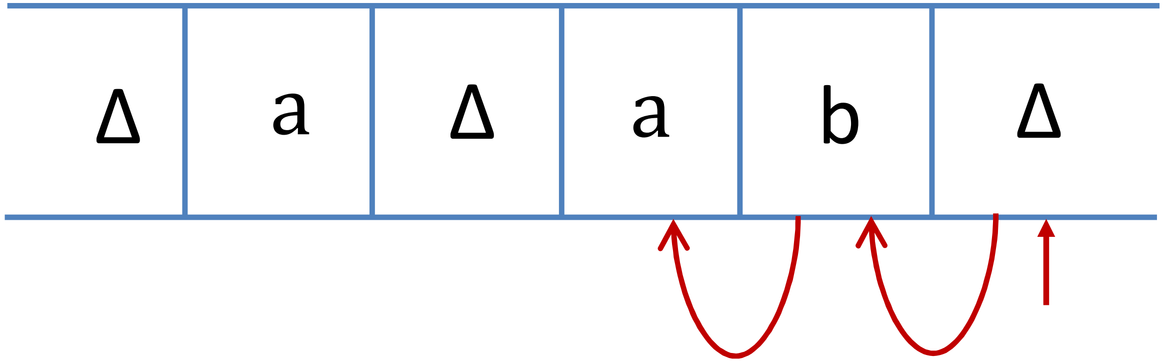


Logic:

1. Replace the symbol you want to delete by Δ.
2. Keep moving to the Right until you encounter Δ.
3. One by one Keep shifting every symbol one step to left until you encounter Δ.

Design a Turing machine to delete a symbol

ΔaabΔ

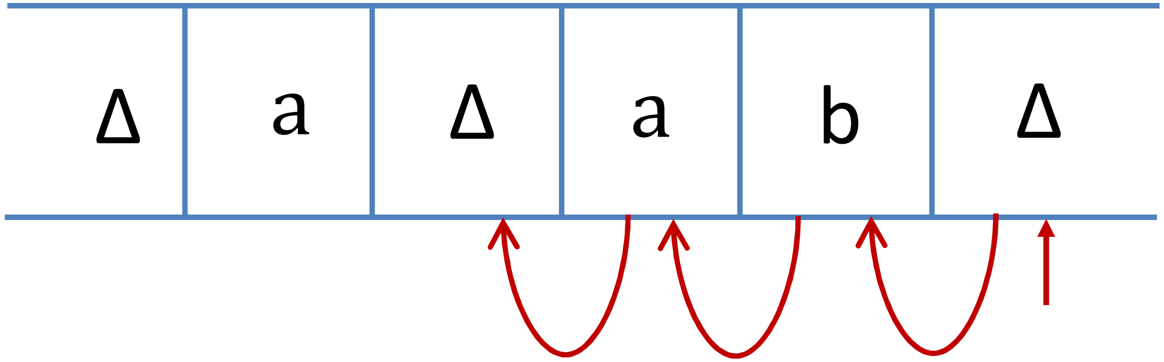


Logic:

1. Replace the symbol you want to delete by Δ.
2. Keep moving to the Right until you encounter Δ.
3. One by one Keep shifting every symbol one step to left until you encounter Δ.

Design a Turing machine to delete a symbol

ΔaabΔ

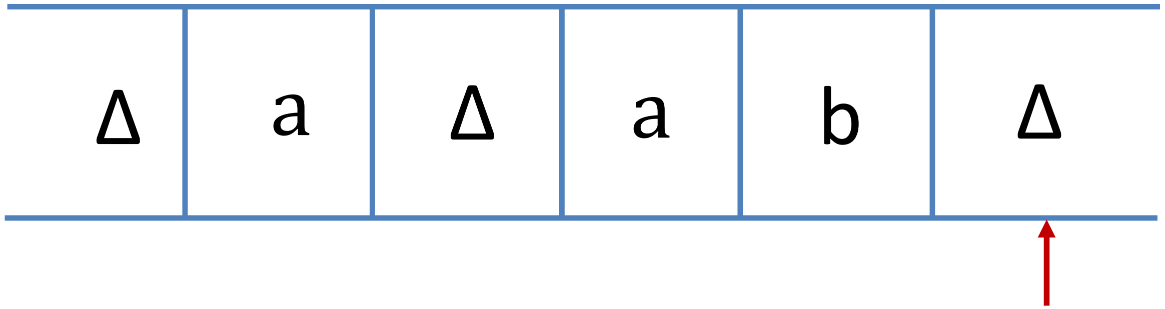


Logic:

1. Replace the symbol you want to delete by Δ.
2. Keep moving to the Right until you encounter Δ.
3. One by one Keep shifting every symbol one step to left until you encounter Δ.

Design a Turing machine to delete a symbol

ΔaabΔ



Δ	a	Δ	a	b	Δ

Logic:

1. Replace the symbol you want to delete by Δ.
2. Keep moving to the Right until you encounter Δ.
3. One by one Keep shifting every symbol one step to left until you encounter Δ.

Design a Turing machine to delete a symbol

ΔaabΔ

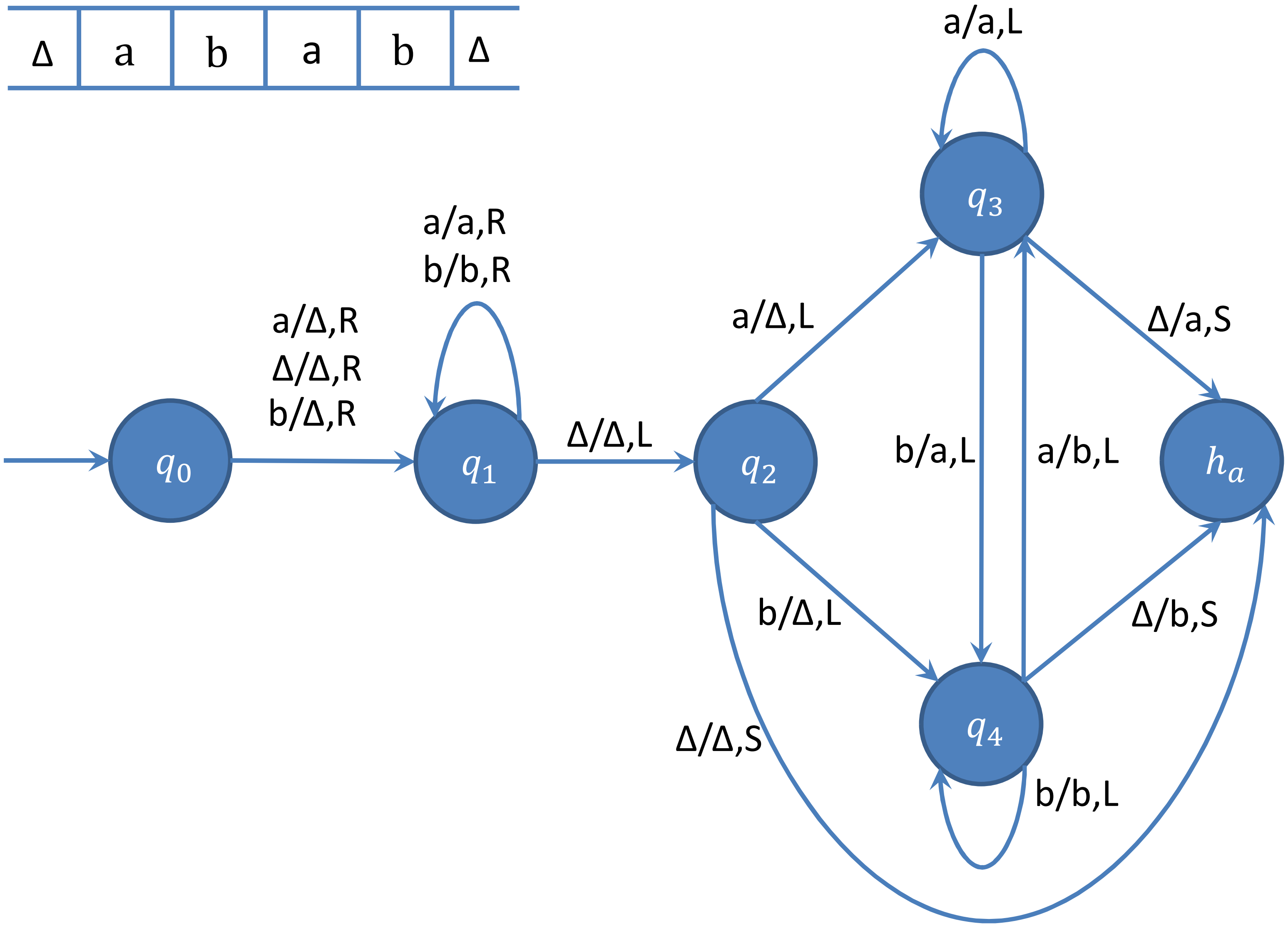
Δ	a	a	b	Δ	

Logic:

1. Replace the symbol you want to delete by Δ.
2. Keep moving to the Right until you encounter Δ.
3. One by one Keep shifting every symbol one step to left until you encounter Δ.

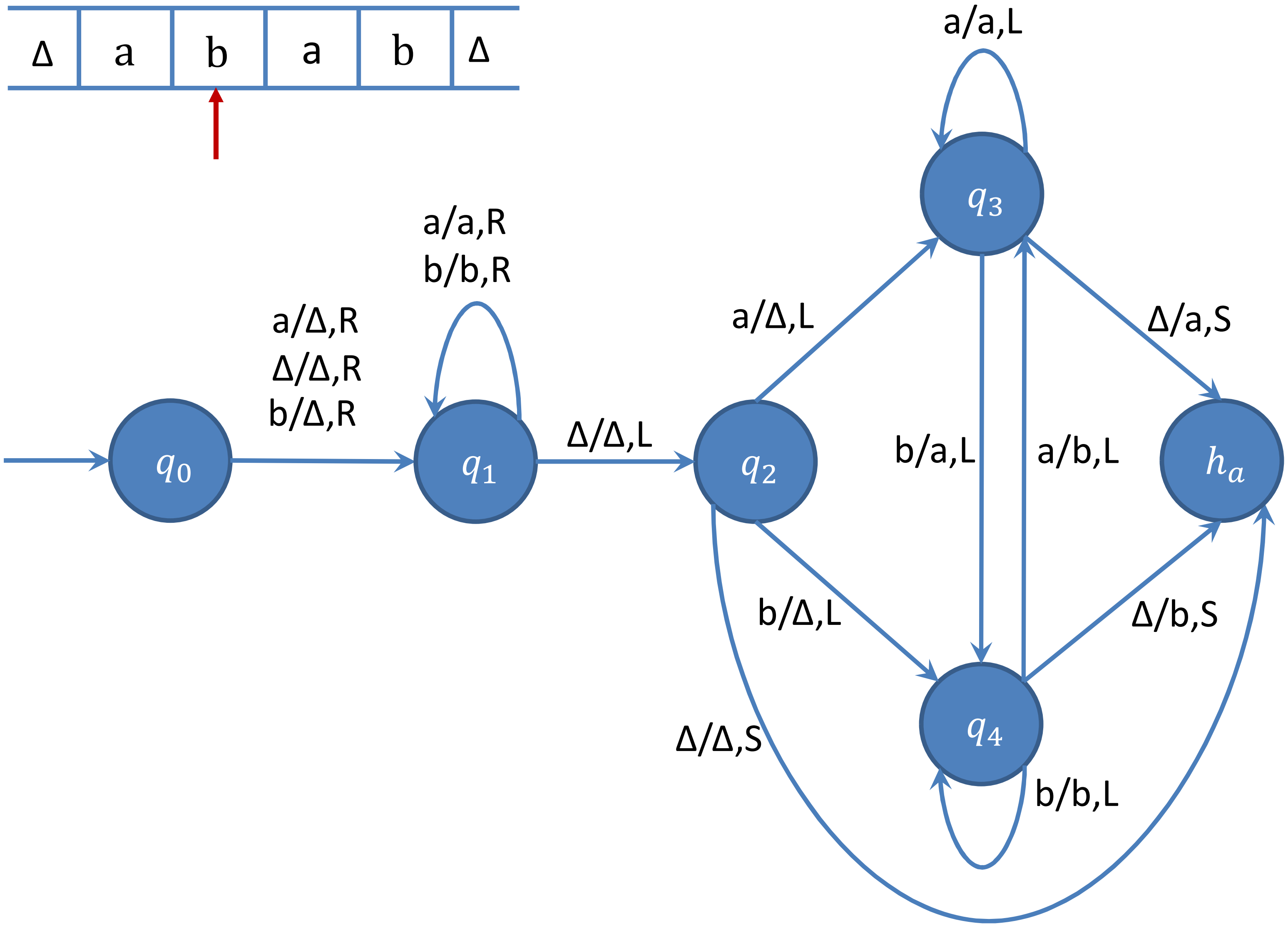
Design a Turing machine to delete a symbol

Δ	a	b	a	b	Δ

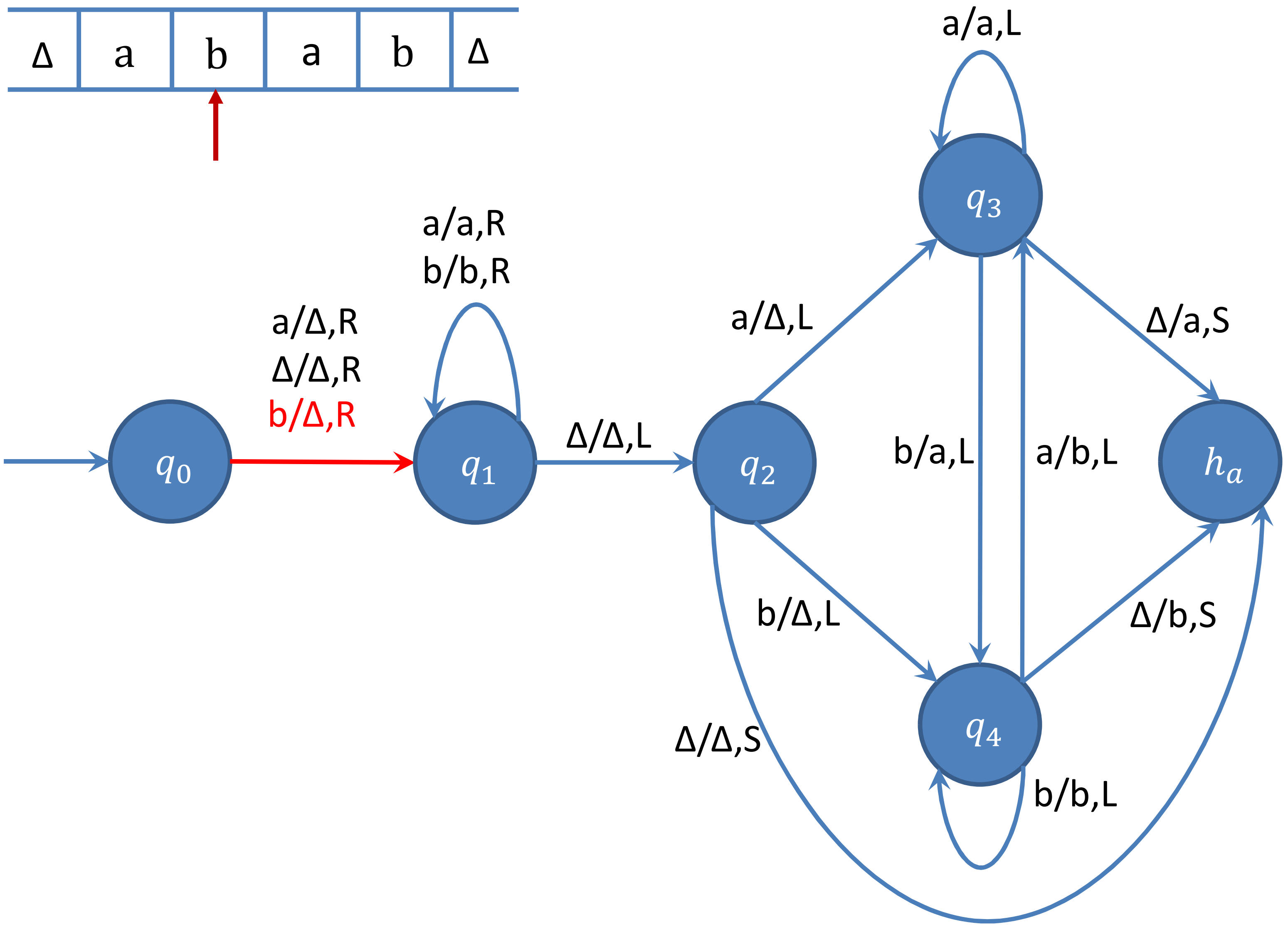


Design a Turing machine to delete a symbol

Δ	a	b	a	b	Δ

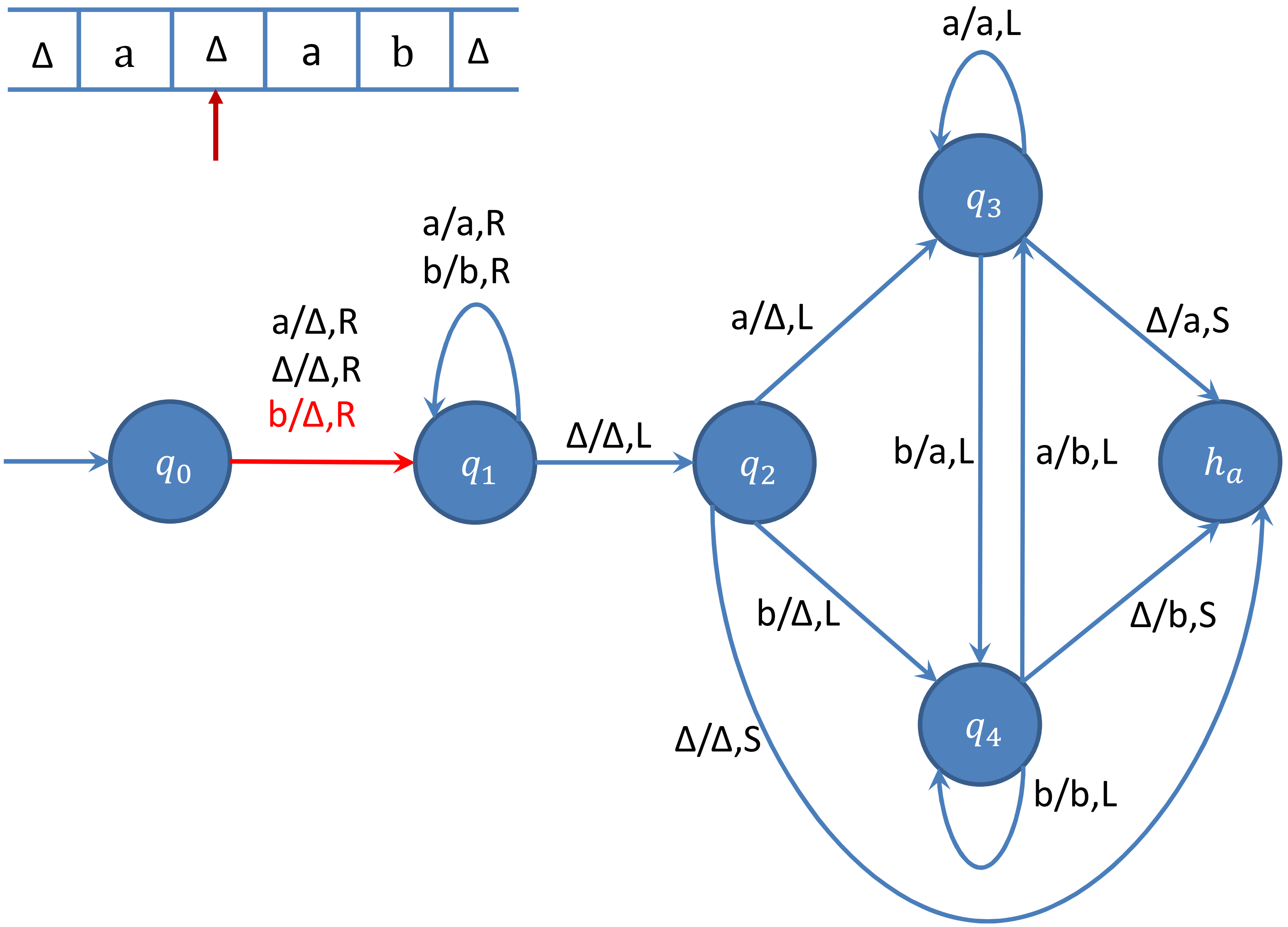


Design a Turing machine to delete a symbol



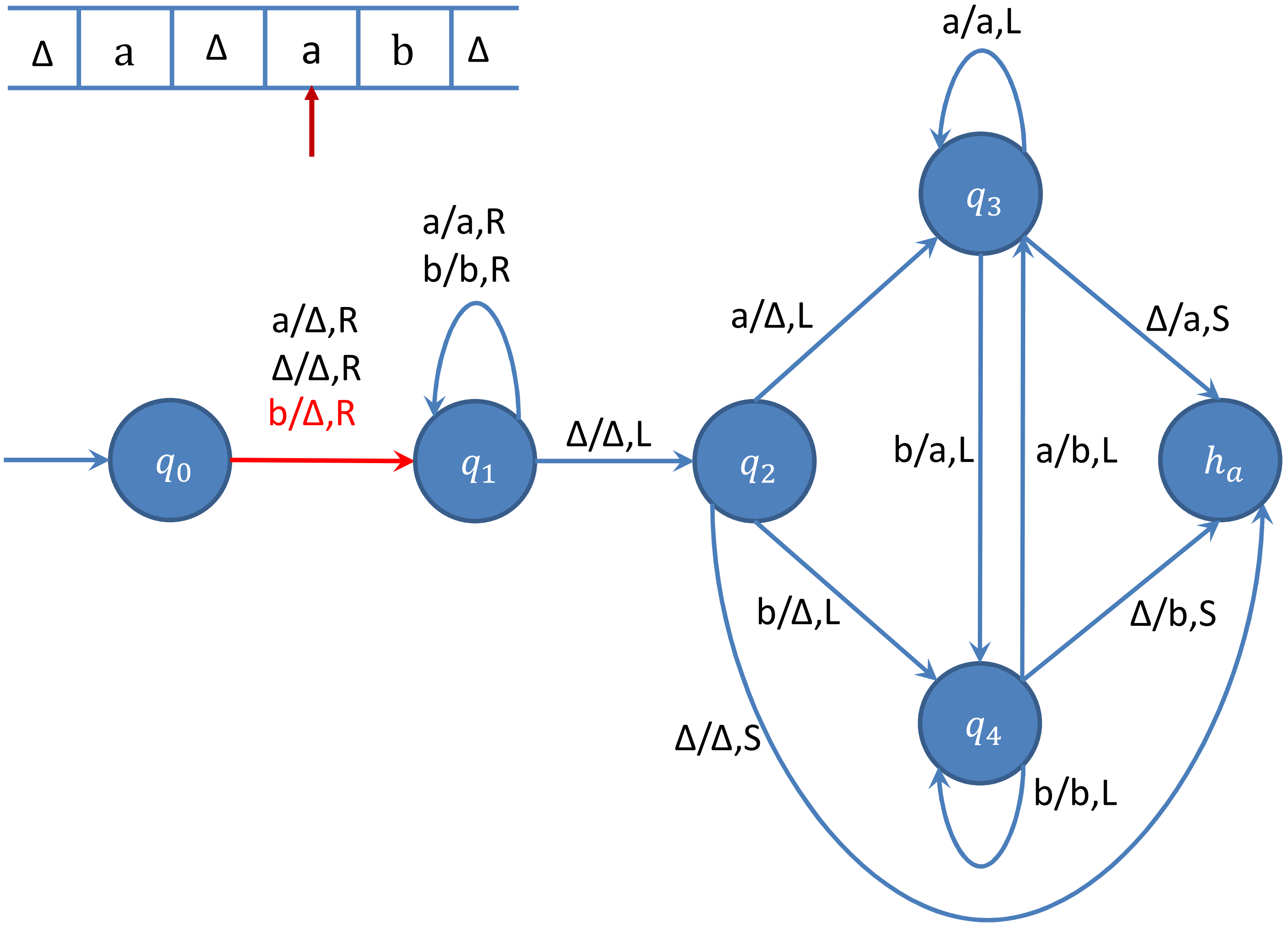
Δ	a	b	a	b	Δ

Design a Turing machine to delete a symbol



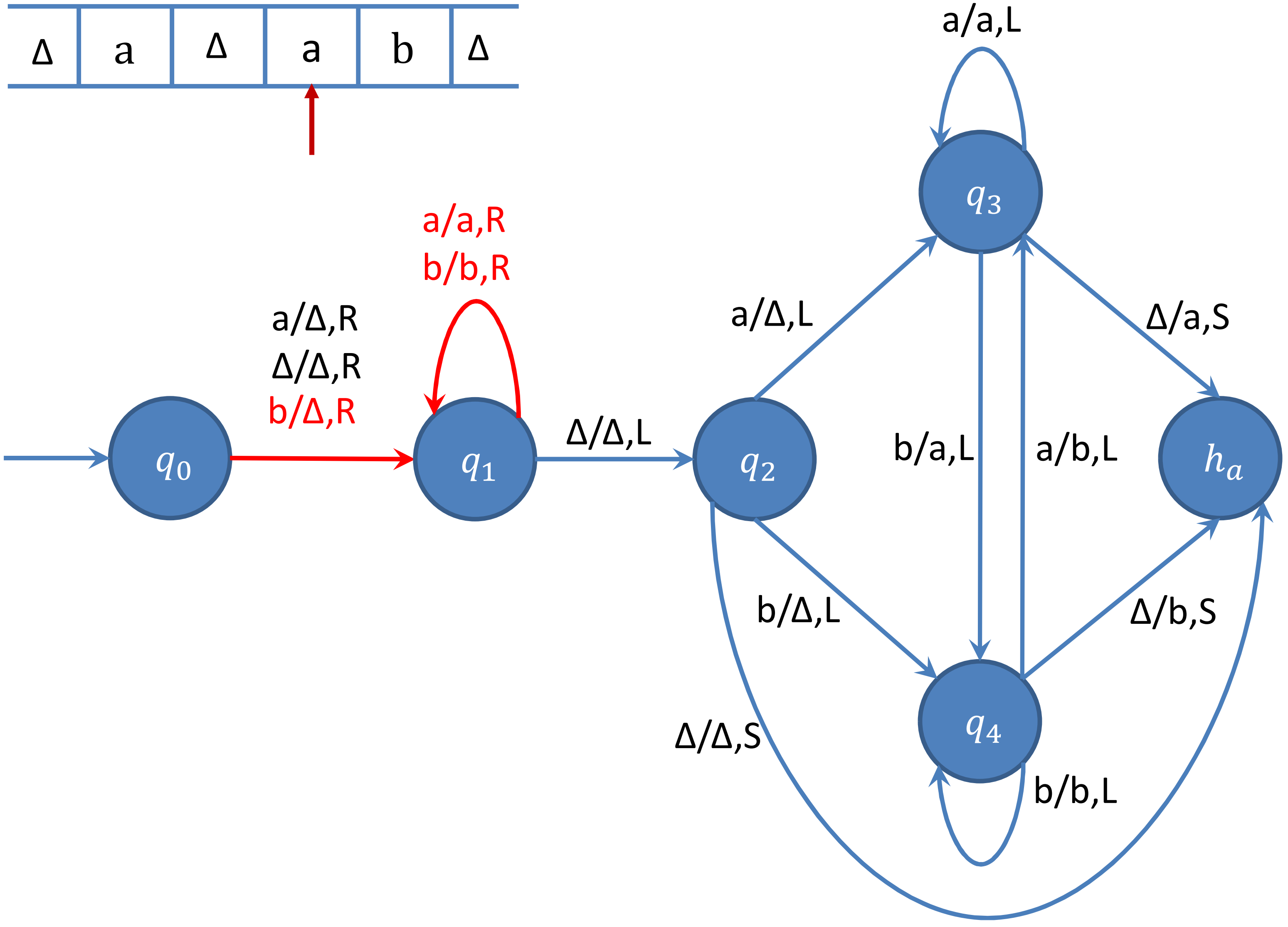
Δ	a	Δ	a	b	Δ

Design a Turing machine to delete a symbol



Δ	a	Δ	a	b	Δ

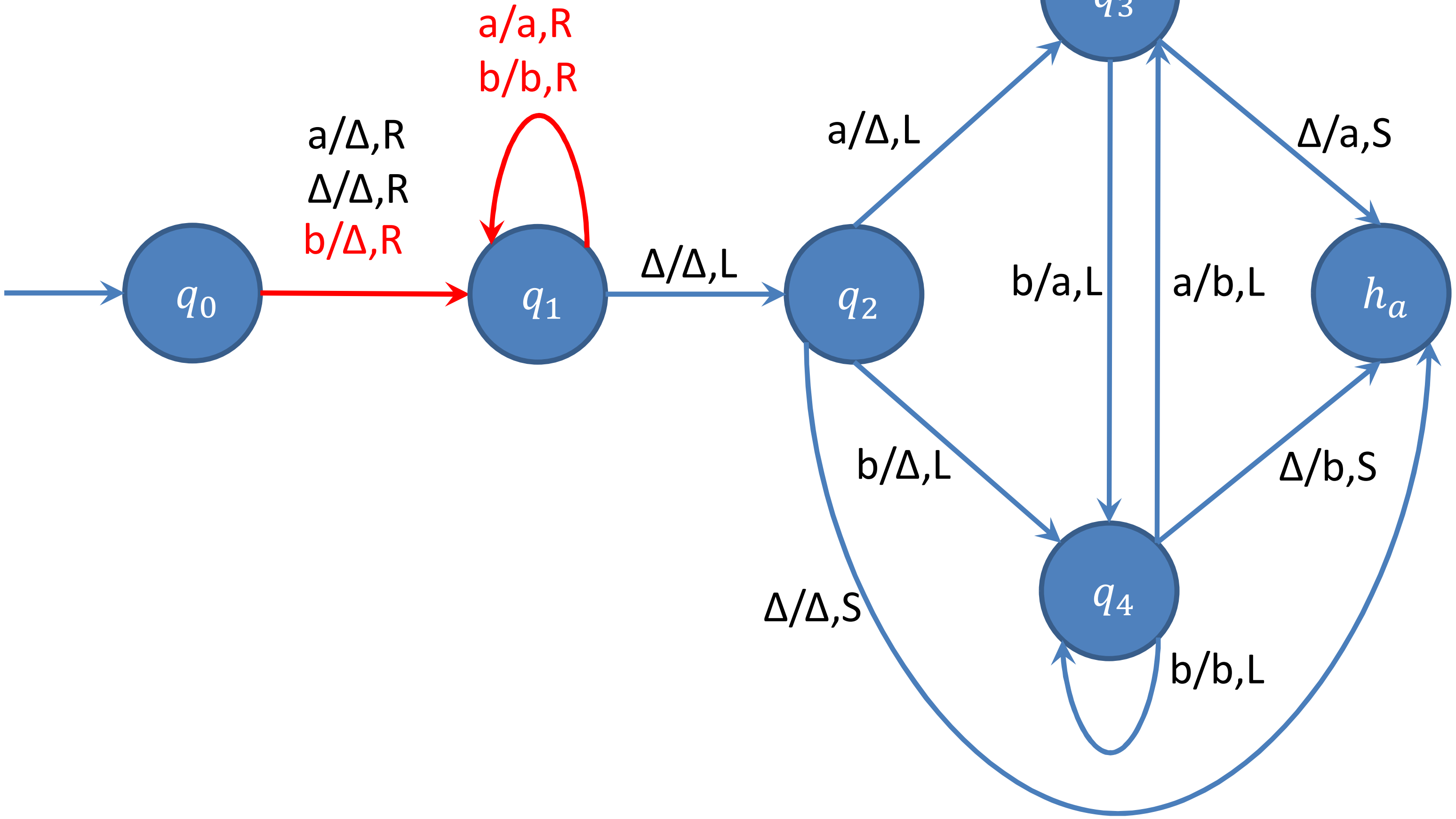
Design a Turing machine to delete a symbol



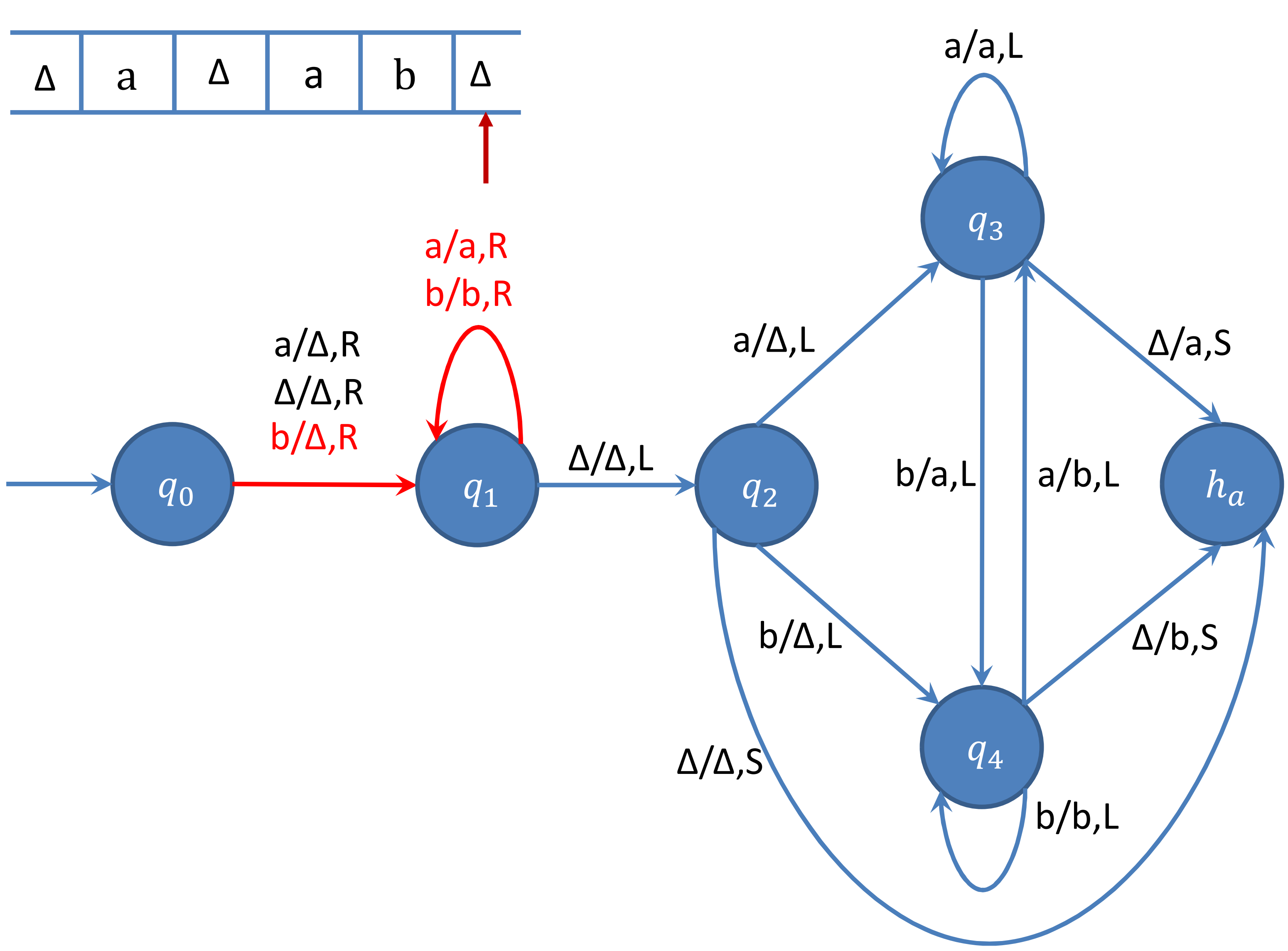
Δ	a	Δ	a	b	Δ

Design a Turing machine to delete a symbol

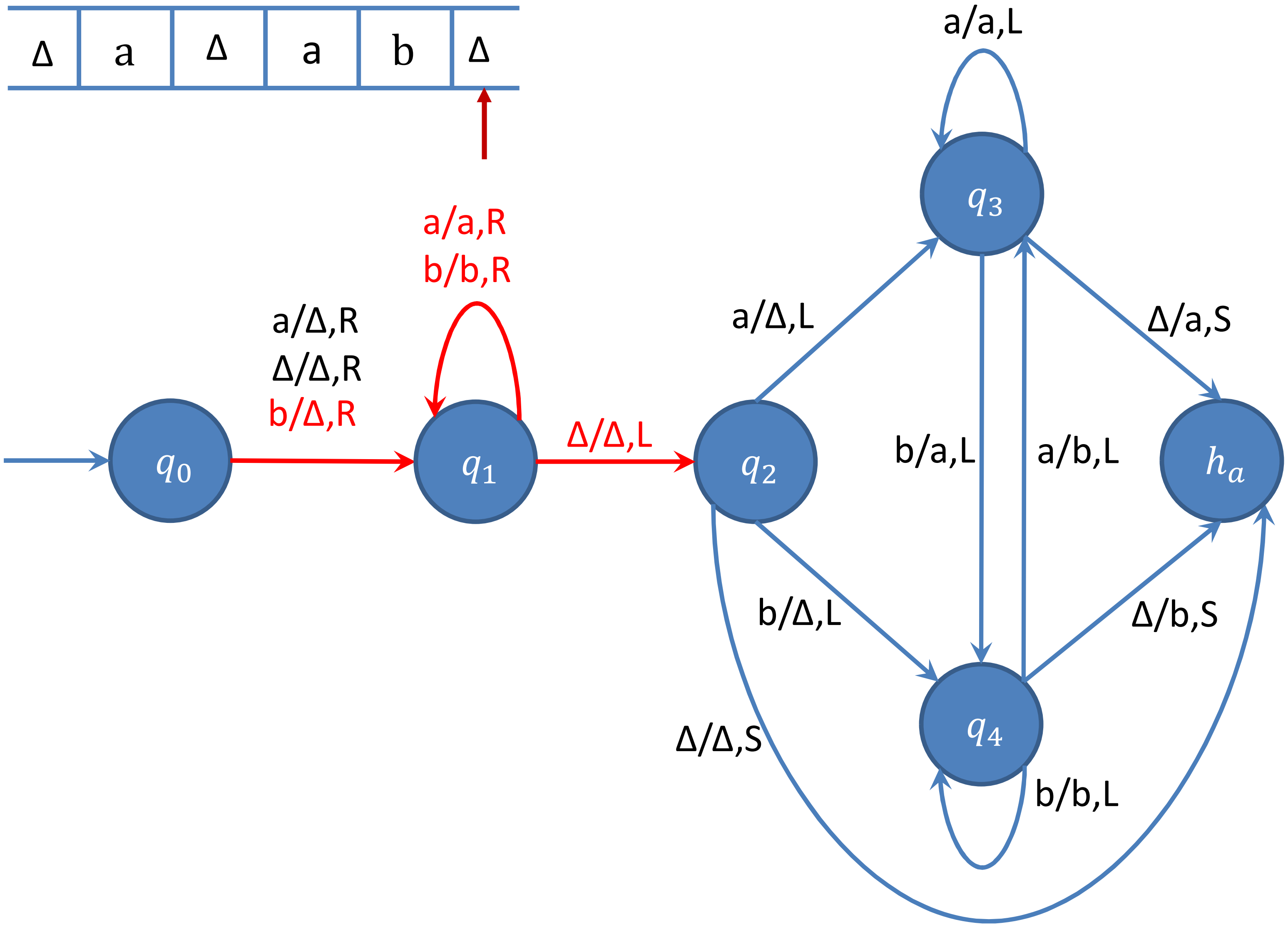
Δ	a	Δ	a	b	Δ



Design a Turing machine to delete a symbol

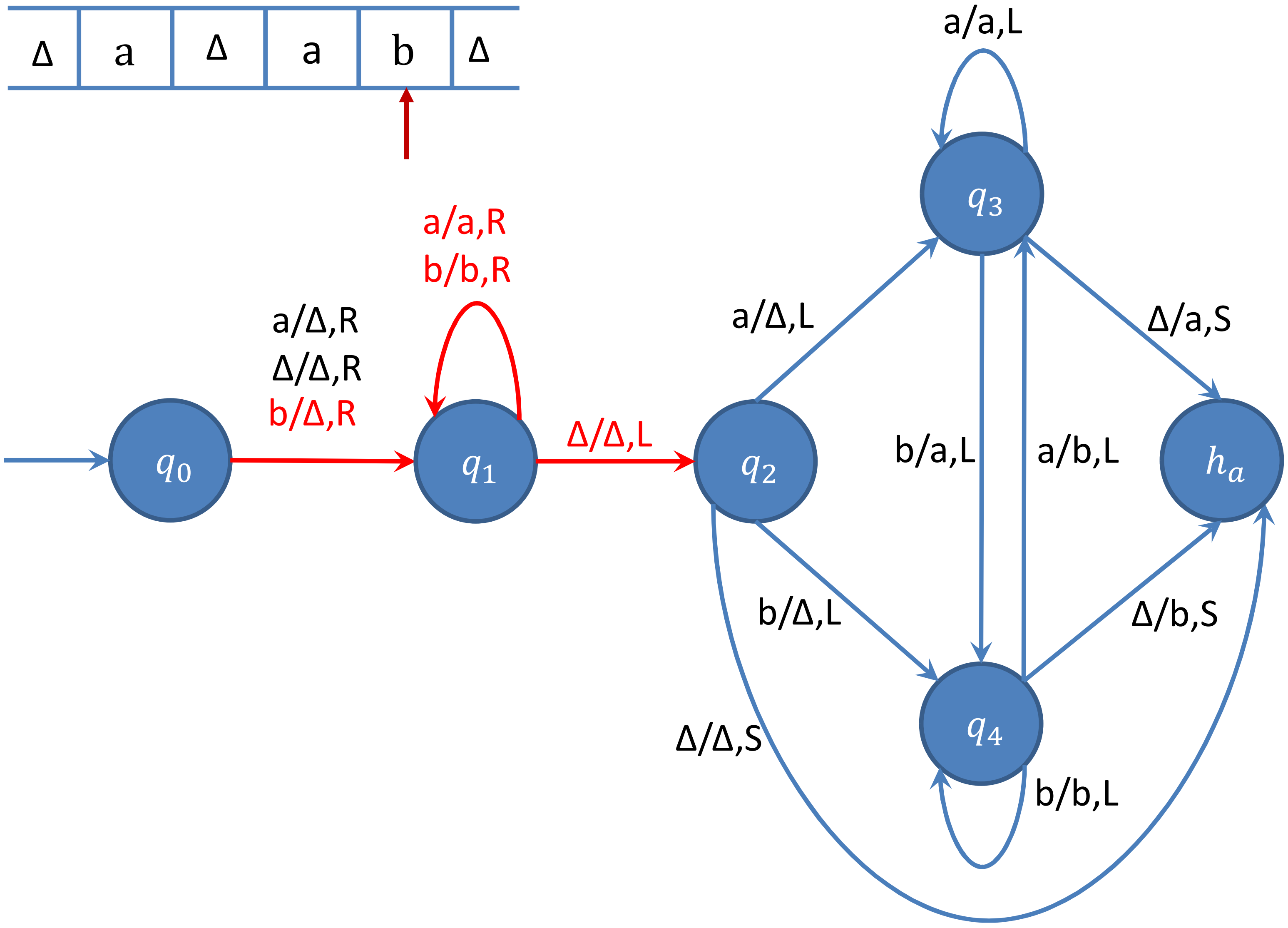


Design a Turing machine to delete a symbol

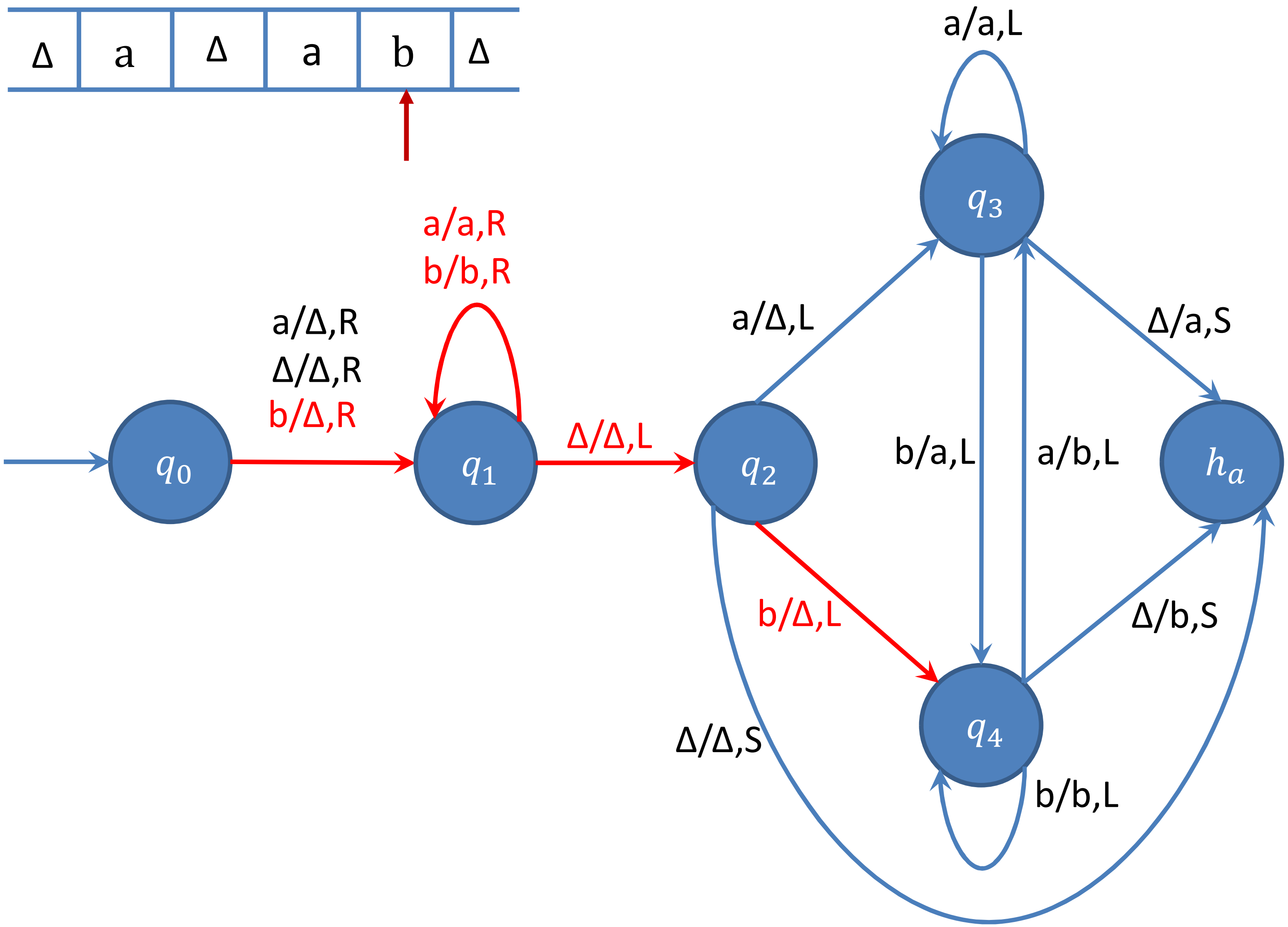


Δ	a	Δ	a	b	Δ

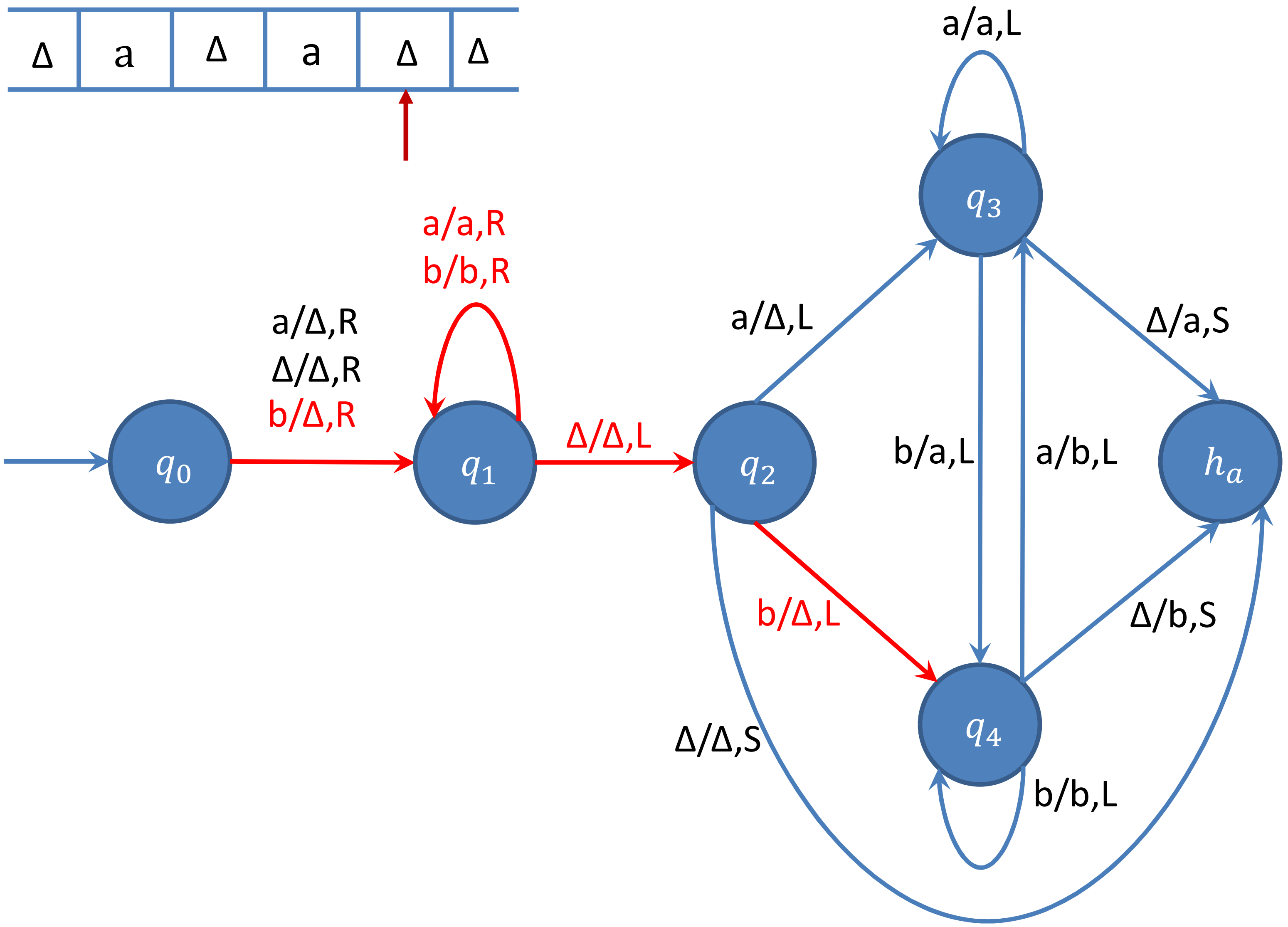
Design a Turing machine to delete a symbol



Design a Turing machine to delete a symbol

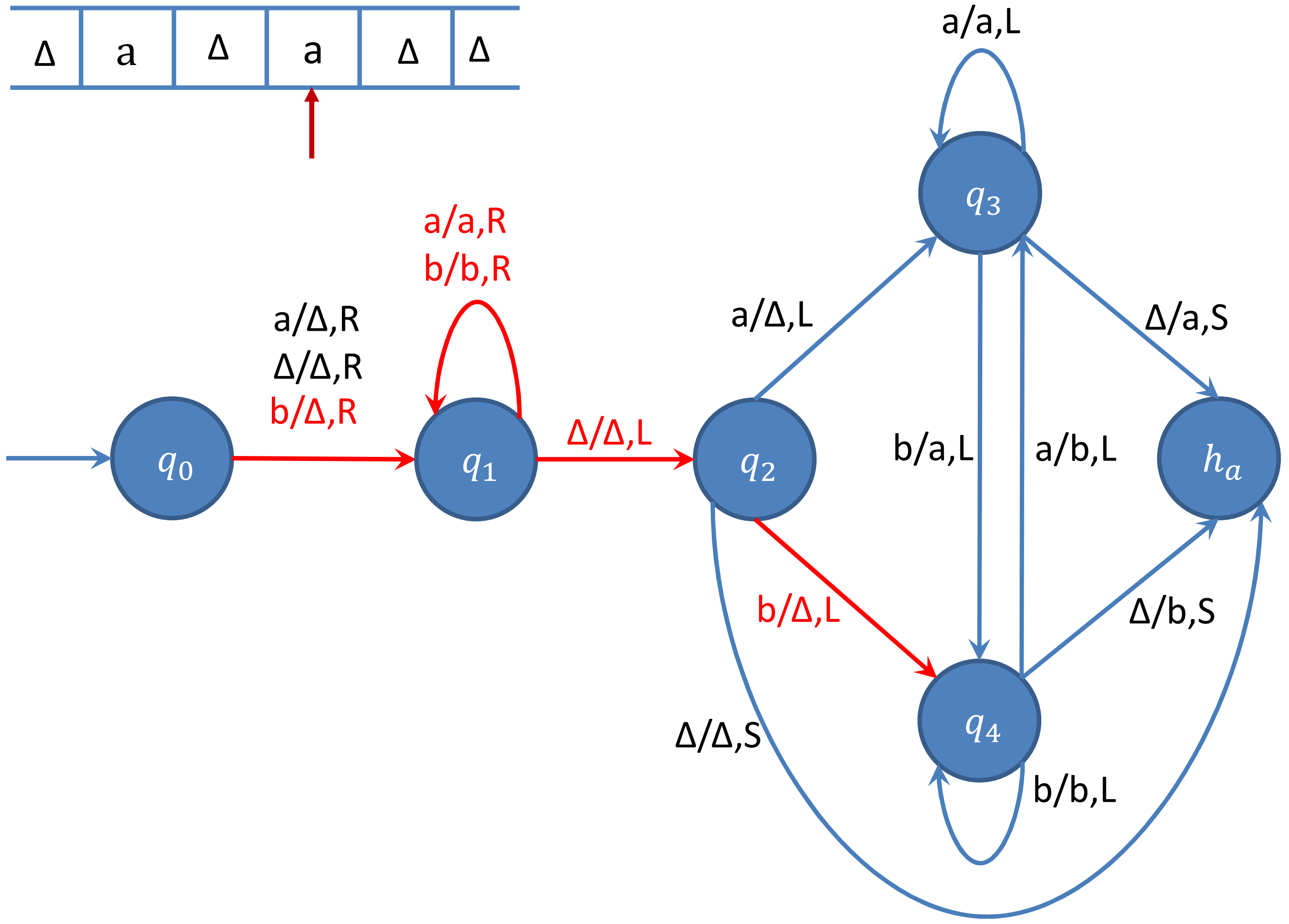


Design a Turing machine to delete a symbol

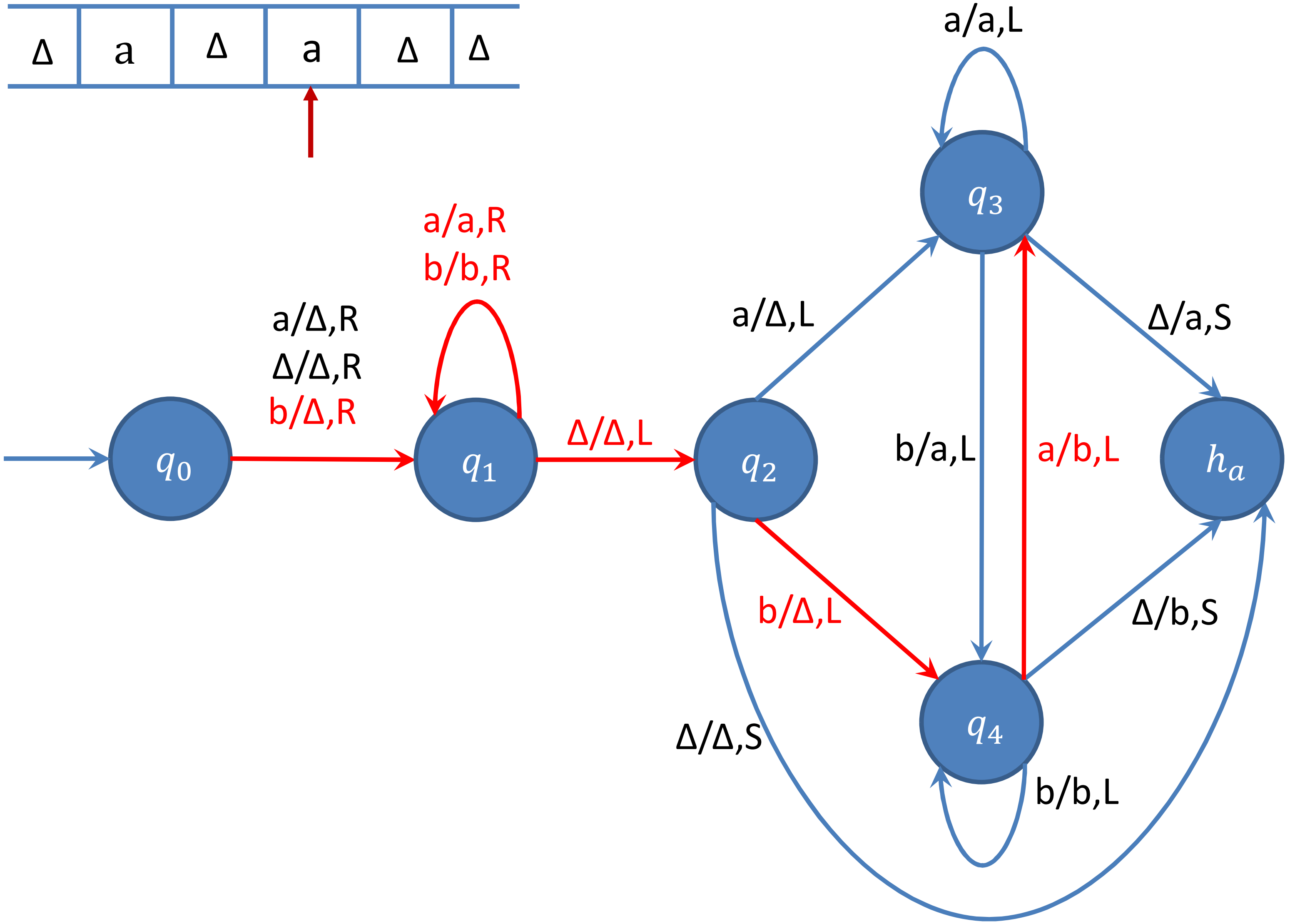


Design a Turing machine to delete a symbol

Δ	a	Δ	a	Δ	Δ

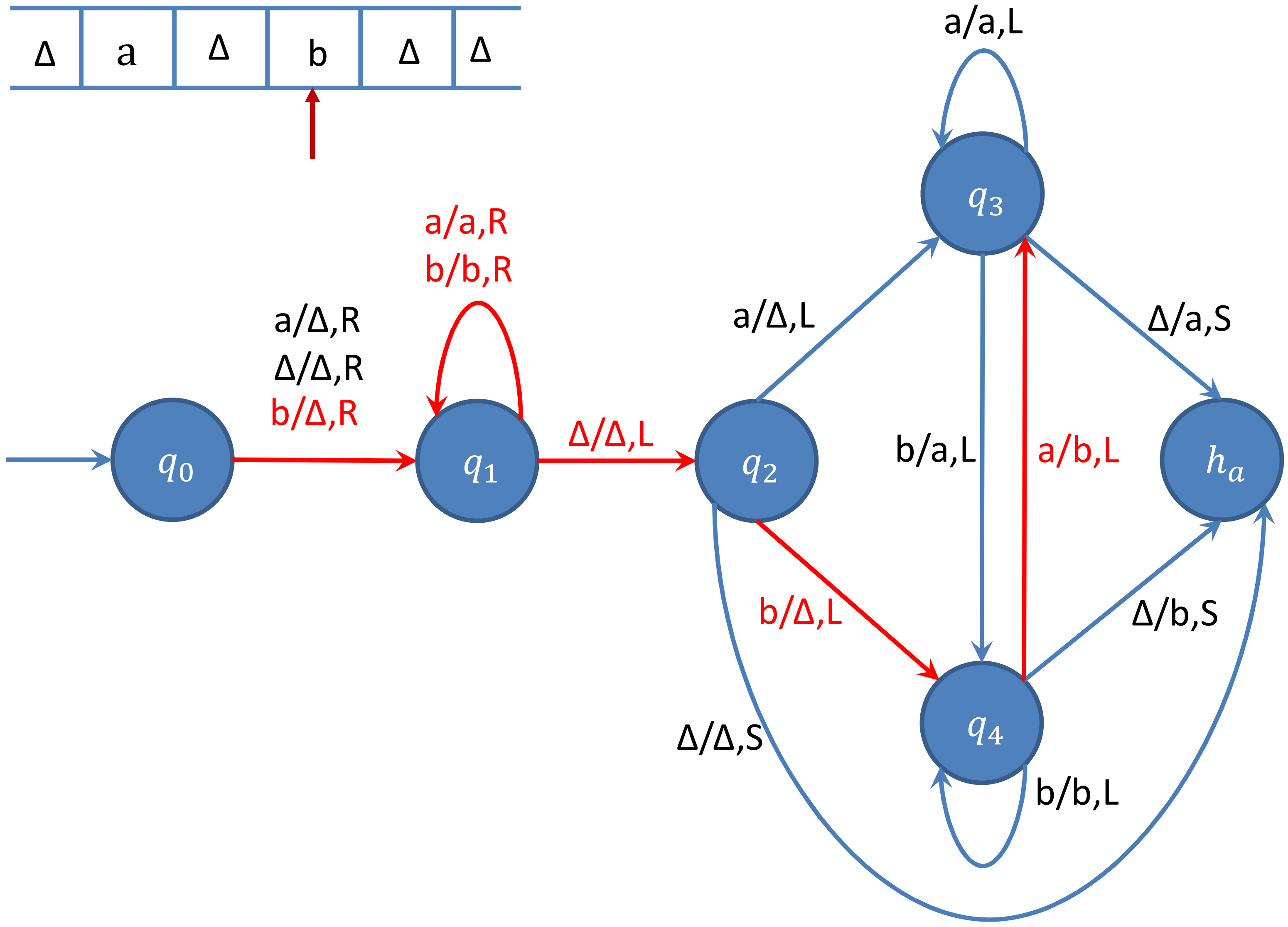


Design a Turing machine to delete a symbol

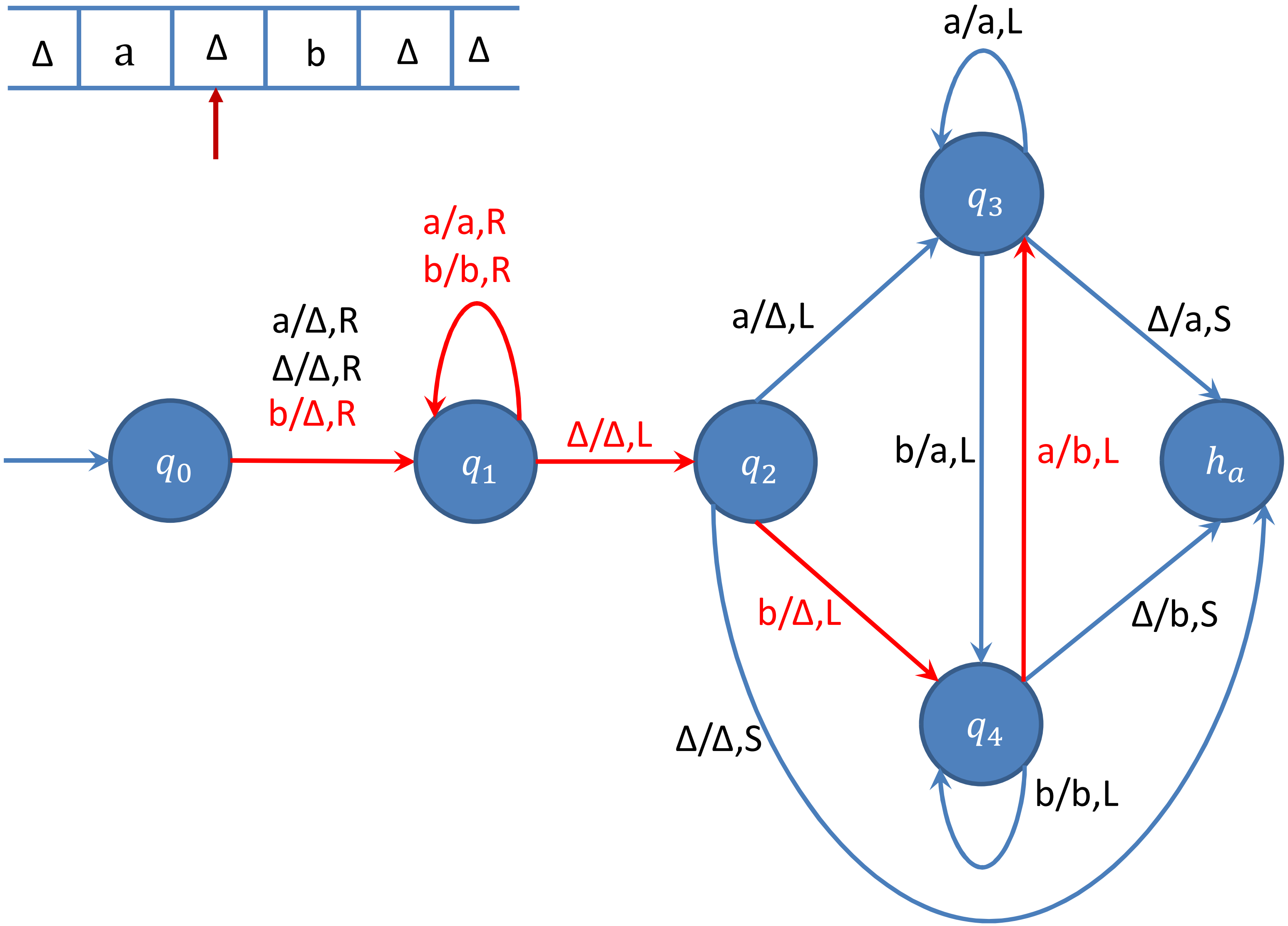


Δ	a	Δ	a	Δ	Δ

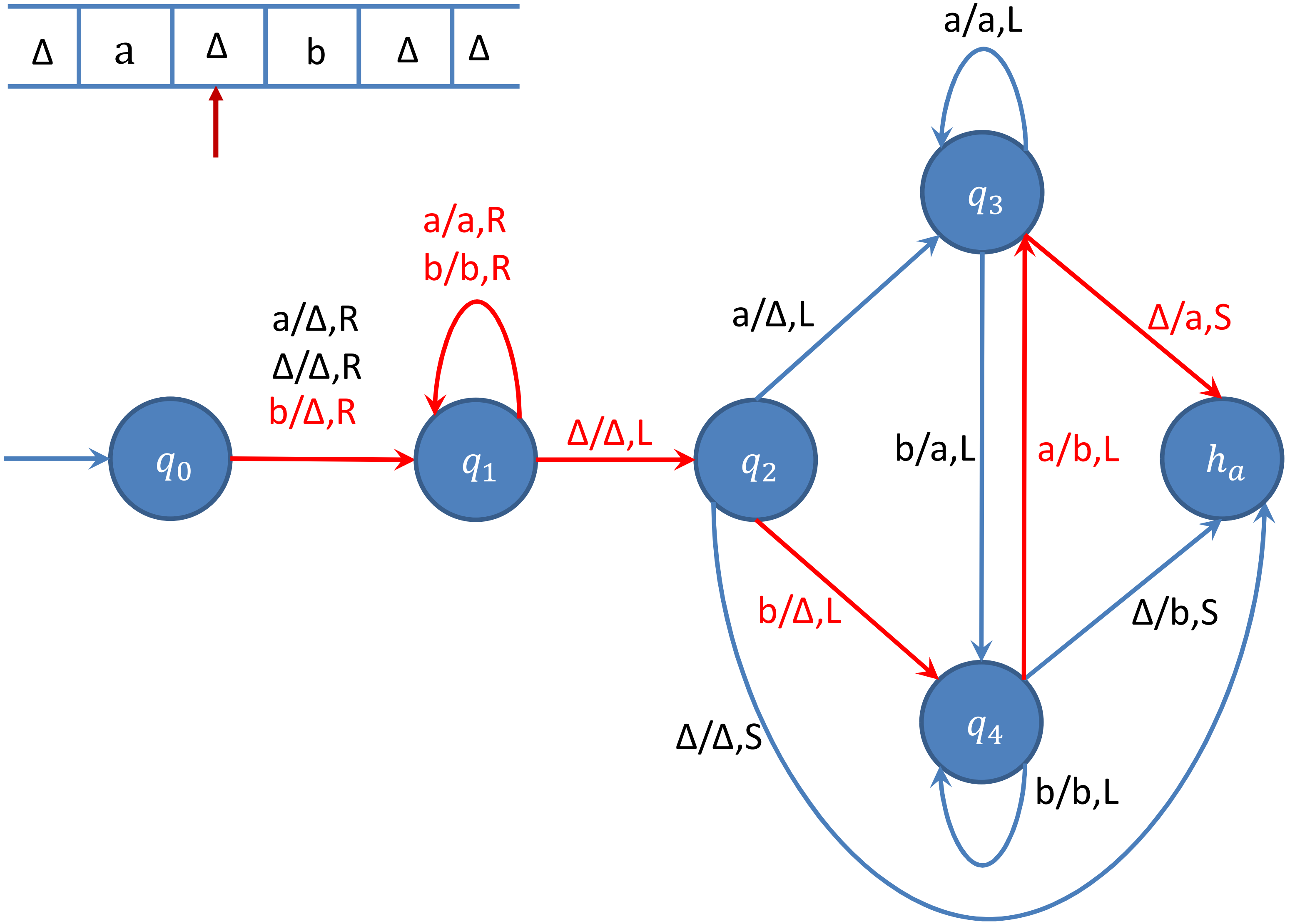
Design a Turing machine to delete a symbol



Design a Turing machine to delete a symbol

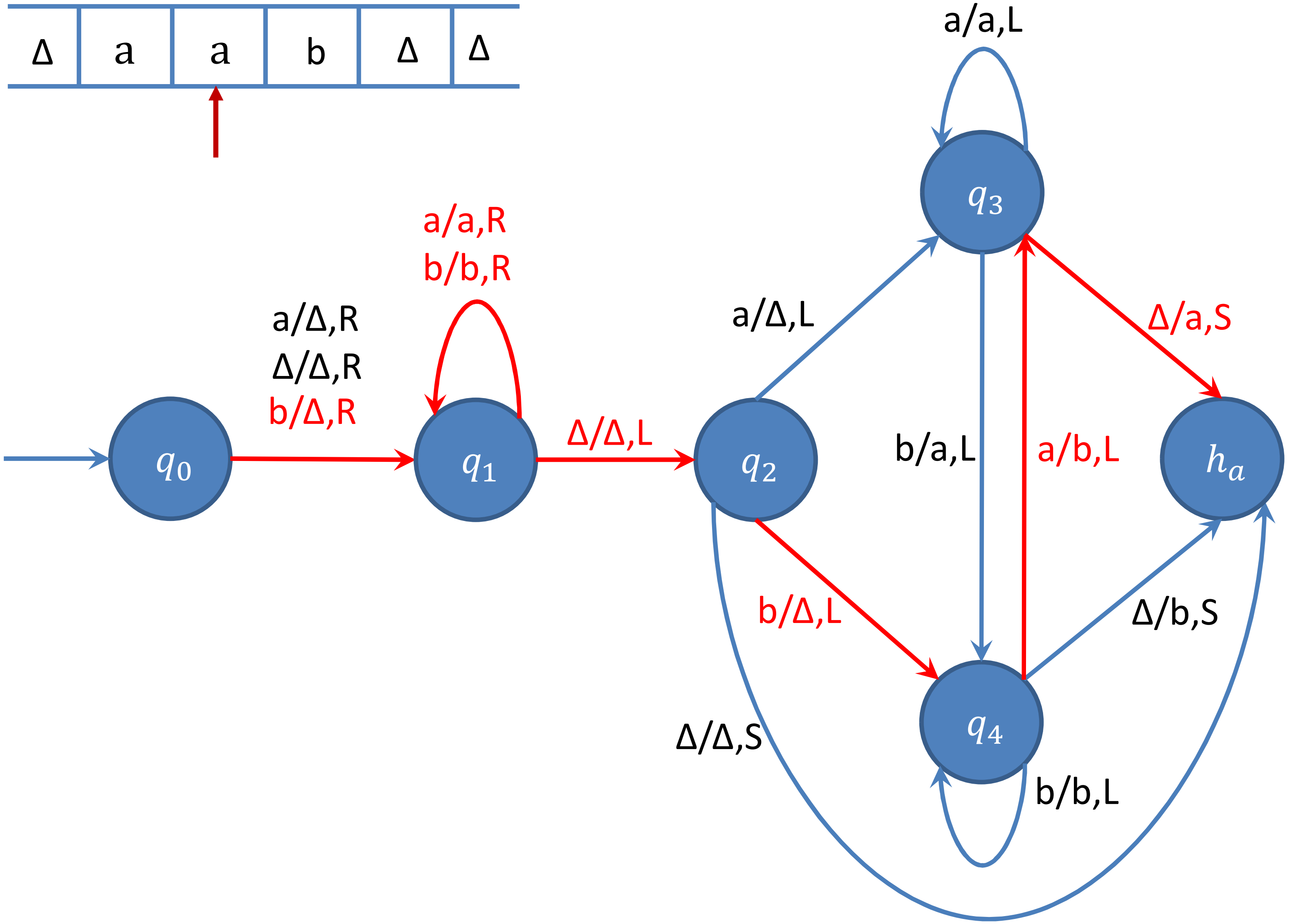


Design a Turing machine to delete a symbol



Δ	a	Δ	b	Δ	Δ

Design a Turing machine to delete a symbol



Δ	a	a	b	Δ	Δ

Turing machine to copy a string

Design a Turing machine to copy a string

Δ	a	b	Δ	Δ	Δ	Δ

Design a Turing machine to copy a string

String:

Δ	a	b	Δ	Δ	Δ	Δ

Design a Turing machine to copy a string

String:

Δ ab

Δ	a	b	Δ	Δ	Δ	Δ

Design a Turing machine to copy a string

String:

Δ ab **Δ** ab

Δ	a	b	Δ	Δ	Δ	Δ

Design a Turing machine to copy a string

String:

Δ ab **Δ** ab

Original

Δ	a	b	Δ	Δ	Δ	Δ

Design a Turing machine to copy a string

String:

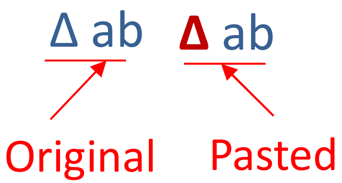
Δ ab Δ ab

Original Pasted

Δ	a	b	Δ	Δ	Δ	Δ

Design a Turing machine to copy a string

String:



Δ ab Δ ab

Original

Pasted

Δ bab

Δ	a	b	Δ	Δ	Δ	Δ

Design a Turing machine to copy a string

String:

Δ ab Δ ab

Original Pasted

Δ bab Δ bab

Δ	a	b	Δ	Δ	Δ	Δ

Design a Turing machine to copy a string

String:

Δ ab **Δ** ab

Original Pasted

Δ bab **Δ** bab

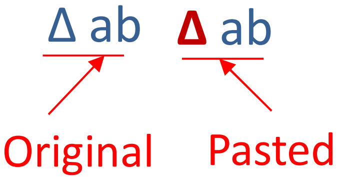
Original

Δ	a	b	Δ	Δ	Δ	Δ

Design a Turing machine to copy a string

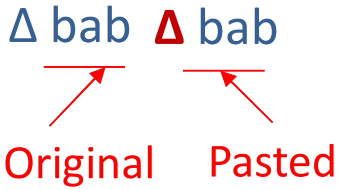
String:

Δ	a	b	Δ	Δ	Δ	Δ



Δ ab **Δ** ab

Original Pasted

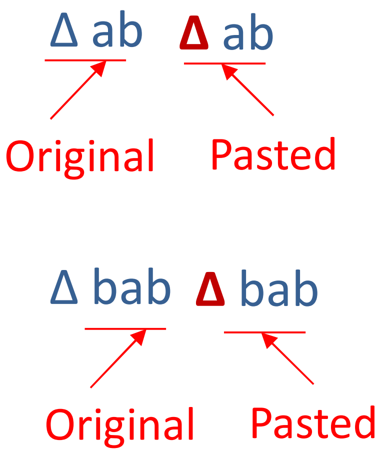


Δ bab **Δ** bab

Original Pasted

Design a Turing machine to copy a string

String:

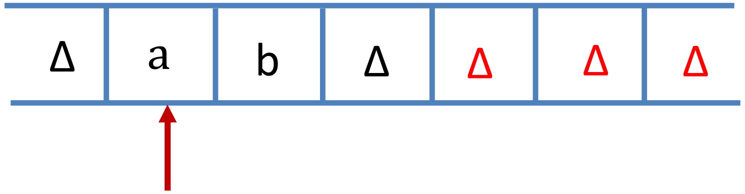


Δ ab Δ ab

Original Pasted

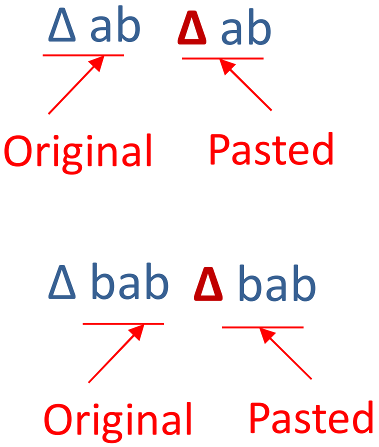
Δ bab Δ bab

Original Pasted



Design a Turing machine to copy a string

String:

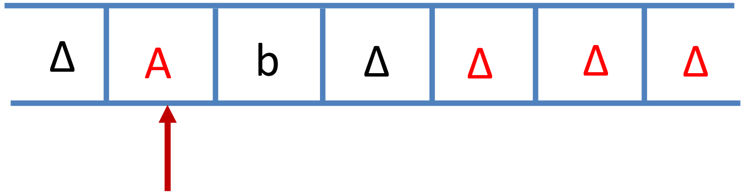


Δ ab **Δ** ab

Original Pasted

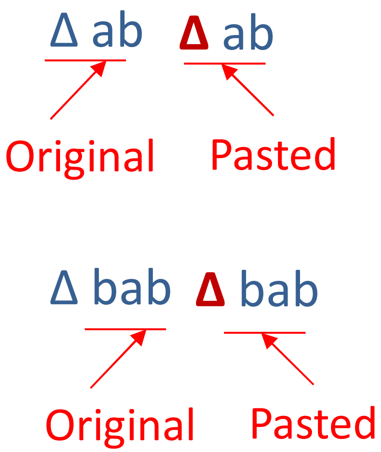
Δ bab **Δ** bab

Original Pasted



Design a Turing machine to copy a string

String:

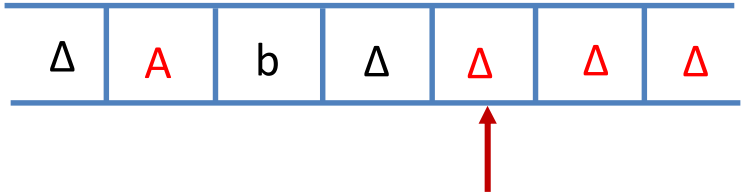


Δ ab **Δ** ab

Original Pasted

Δ bab **Δ** bab

Original Pasted



Design a Turing machine to copy a string

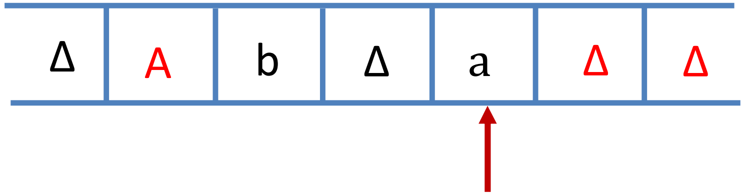
String:

Δ ab Δ ab

Original Pasted

Δ bab Δ bab

Original Pasted



Δ	A	b	Δ	a	Δ	Δ

Design a Turing machine to copy a string

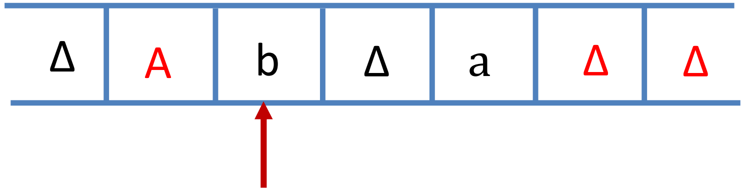
String:

Δ ab Δ ab

Original Pasted

Δ bab Δ bab

Original Pasted



Δ	A	b	Δ	a	Δ	Δ

Design a Turing machine to copy a string

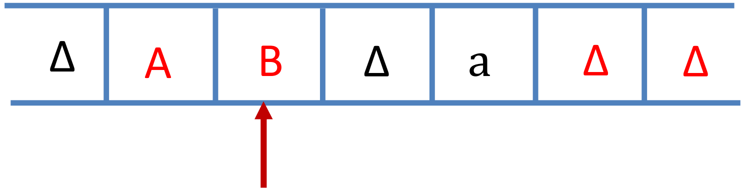
String:

Δ ab **Δ** ab

Original Pasted

Δ bab **Δ** bab

Original Pasted



Δ	A	B	Δ	a	Δ	Δ

Design a Turing machine to copy a string

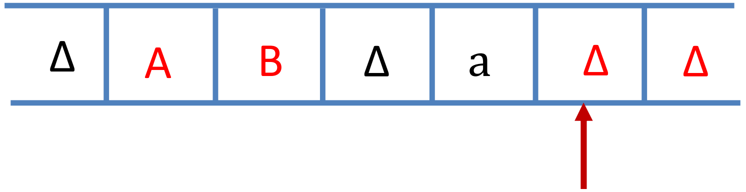
String:

Δ ab **Δ** ab

Original Pasted

Δ bab **Δ** bab

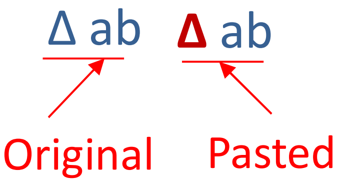
Original Pasted



Δ	A	B	Δ	a	Δ	Δ

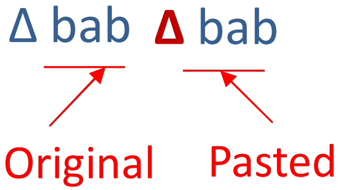
Design a Turing machine to copy a string

String:



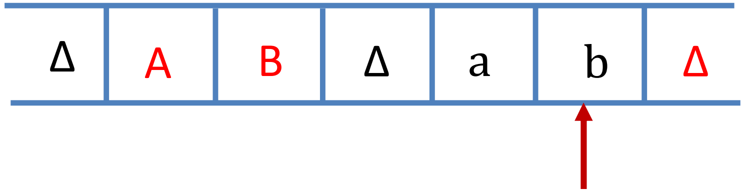
Δ ab Δ ab

Original Pasted



Δ bab Δ bab

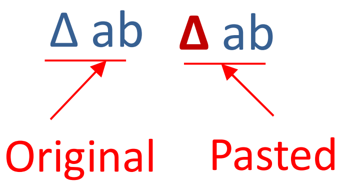
Original Pasted



Δ	A	B	Δ	a	b	Δ

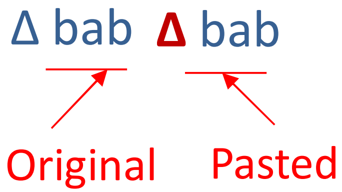
Design a Turing machine to copy a string

String:


Δ ab Δ ab

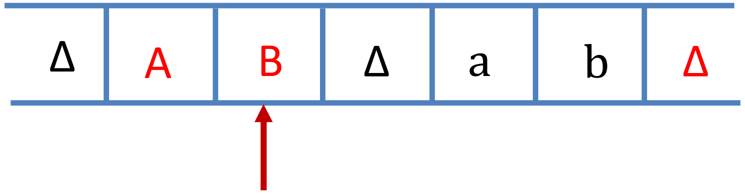
Original

Pasted


Δ bab Δ bab

Original

Pasted



Δ	A	B	Δ	a	b	Δ

Design a Turing machine to copy a string

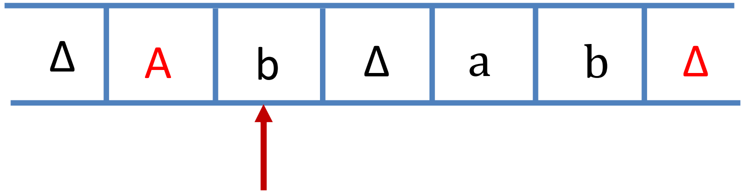
String:

Δ ab **Δ** ab

Original Pasted

Δ bab **Δ** bab

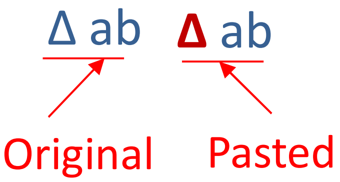
Original Pasted



Δ	A	b	Δ	a	b	Δ

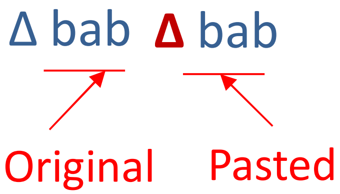
Design a Turing machine to copy a string

String:



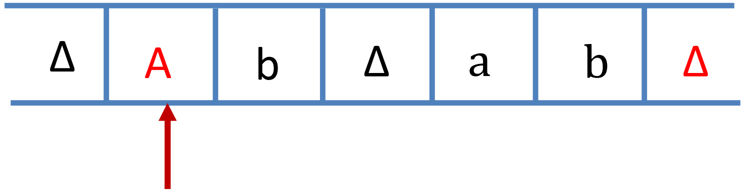
Δ ab Δ ab

Original Pasted



Δ bab Δ bab

Original Pasted



Δ	A	b	Δ	a	b	Δ

Design a Turing machine to copy a string

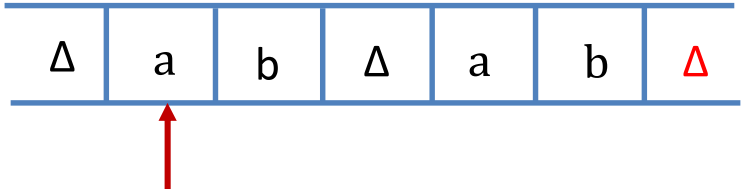
String:

Δ ab **Δ** ab

Original Pasted

Δ bab **Δ** bab

Original Pasted



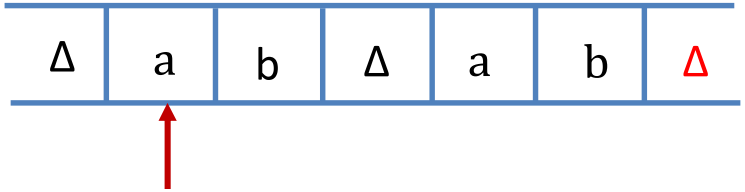
Δ	a	b	Δ	a	b	Δ

Design a Turing machine to copy a string

String:

Δ ab **Δ** ab

Original Pasted



Δ	a	b	Δ	a	b	Δ

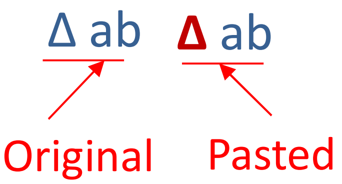
Δ bab **Δ** bab

Original Pasted

Δ	a	b	b	a	Δ	Δ	Δ	Δ	Δ	Δ

Design a Turing machine to copy a string

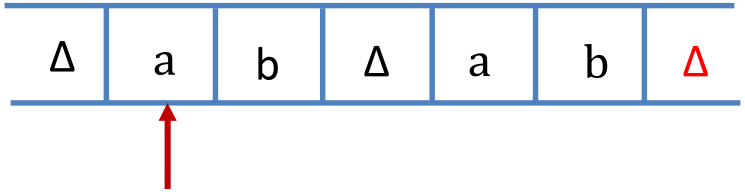
String:



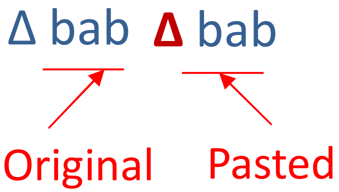
Δ ab **Δ** ab

Original

Pasted



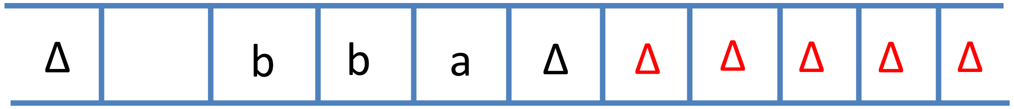
Δ	a	b	Δ	a	b	Δ



Δ bab **Δ** bab

Original

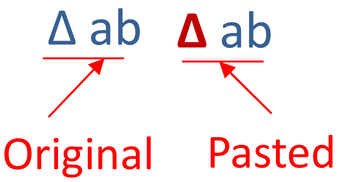
Pasted



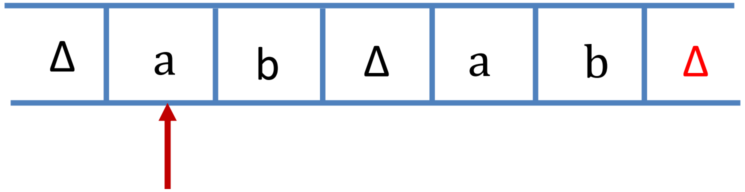
Δ		b	b	a	Δ	Δ	Δ	Δ	Δ	Δ

Design a Turing machine to copy a string

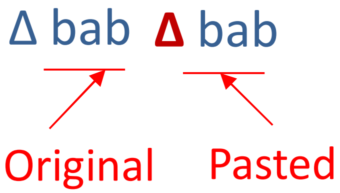
String:


Δ ab Δ ab

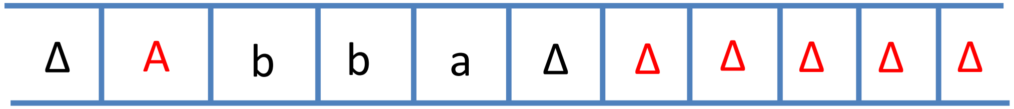
Original Pasted



Δ	a	b	Δ	a	b	Δ


Δ bab Δ bab

Original Pasted



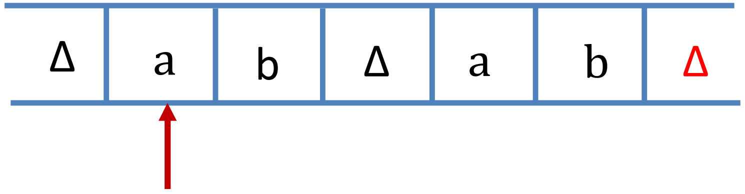
Δ	A	b	b	a	Δ	Δ	Δ	Δ	Δ	Δ

Design a Turing machine to copy a string

String:

Δ ab **Δ** ab

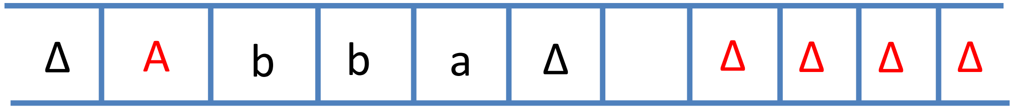
Original Pasted



Δ	a	b	Δ	a	b	Δ

Δ bab **Δ** bab

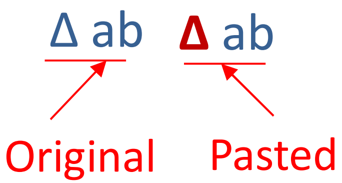
Original Pasted



Δ	A	b	b	a	Δ		Δ	Δ	Δ	Δ

Design a Turing machine to copy a string

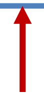
String:

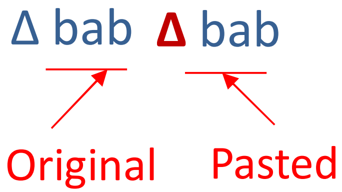


Δ ab **Δ** ab

Original Pasted

Δ	a	b	Δ	a	b	Δ





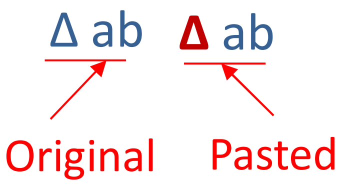
Δ bab **Δ** bab

Original Pasted

Δ	A	b	b	a	Δ	a	Δ	Δ	Δ	Δ

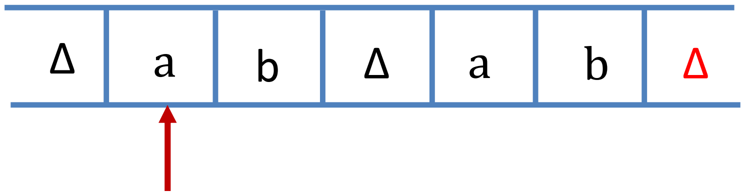
Design a Turing machine to copy a string

String:

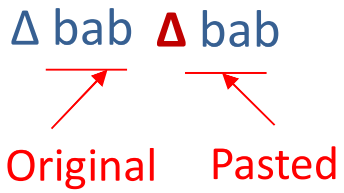


Δ ab Δ ab

Original Pasted



Δ	a	b	Δ	a	b	Δ



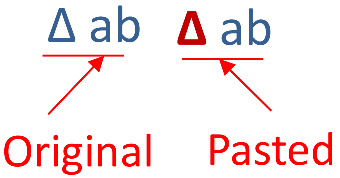
Δ bab Δ bab

Original Pasted

Δ	A		b	a	Δ	a	Δ	Δ	Δ	Δ

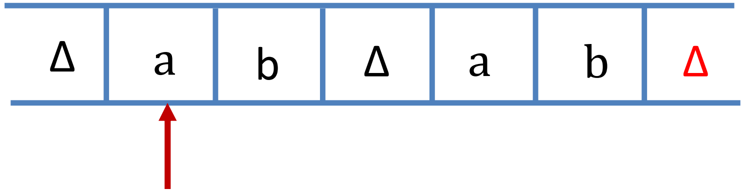
Design a Turing machine to copy a string

String:

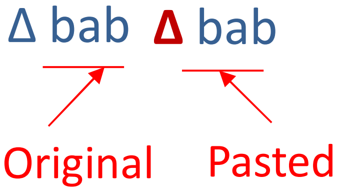


Δ ab **Δ** ab

Original Pasted

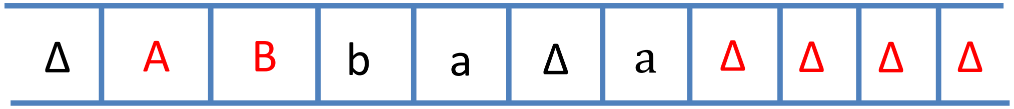


Δ	a	b	Δ	a	b	Δ



Δ bab **Δ** bab

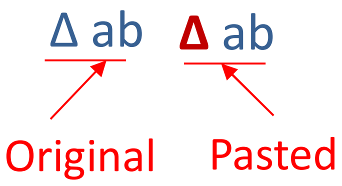
Original Pasted



Δ	A	B	b	a	Δ	a	Δ	Δ	Δ	Δ

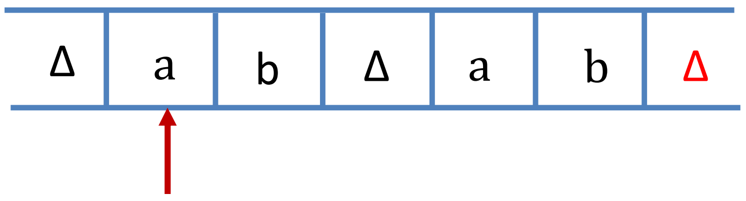
Design a Turing machine to copy a string

String:

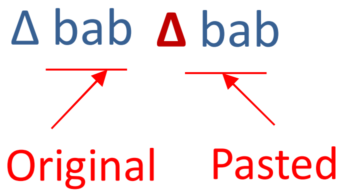


Δ ab Δ ab

Original Pasted



Δ	a	b	Δ	a	b	Δ



Δ bab Δ bab

Original Pasted

Δ	A	B	b	a	Δ	a		Δ	Δ	Δ

Design a Turing machine to copy a string

String:

Δ ab **Δ** ab

Original Pasted

Δ	a	b	Δ	a	b	Δ

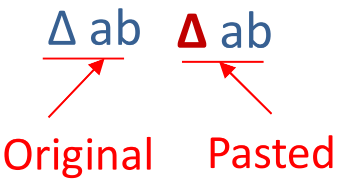
Δ bab **Δ** bab

Original Pasted

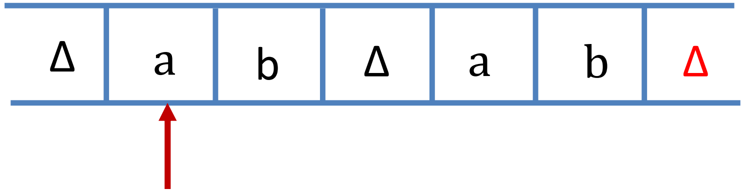
Δ	A	B	b	a	Δ	a	b	Δ	Δ	Δ

Design a Turing machine to copy a string

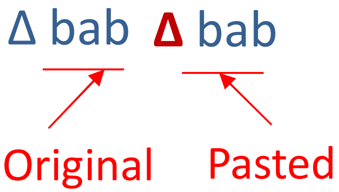
String:


Δ ab **Δ** ab

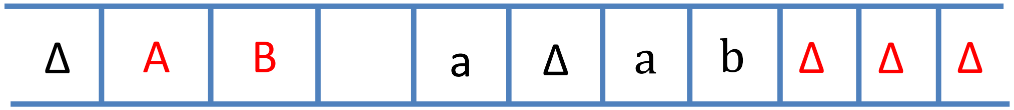
Original Pasted



Δ	a	b	Δ	a	b	Δ


Δ bab **Δ** bab

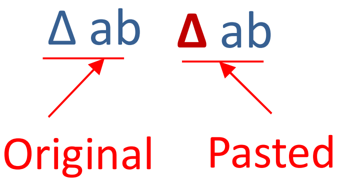
Original Pasted



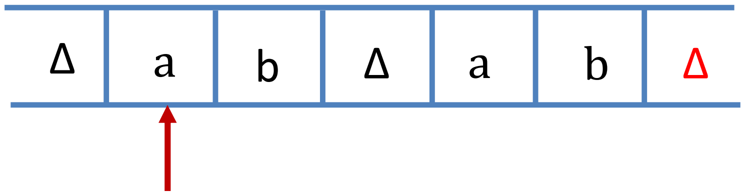
Δ	A	B		a	Δ	a	b	Δ	Δ	Δ

Design a Turing machine to copy a string

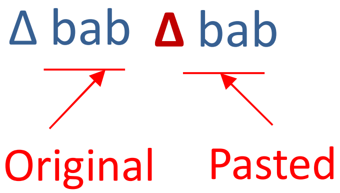
String:


Δ ab Δ ab

Original Pasted



Δ	a	b	Δ	a	b	Δ

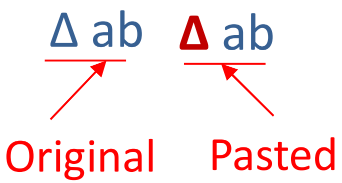

Δ bab Δ bab

Original Pasted

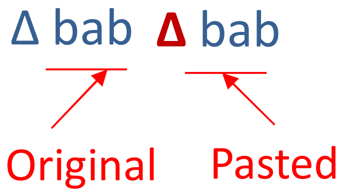
Δ	A	B	B	a	Δ	a	b	Δ	Δ	Δ

Design a Turing machine to copy a string

String:



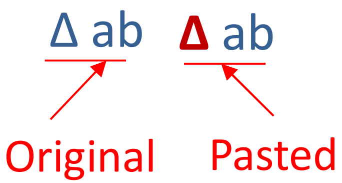
Δ	a	b	Δ	a	b	Δ



Δ	A	B	B	a	Δ	a	b		Δ	Δ

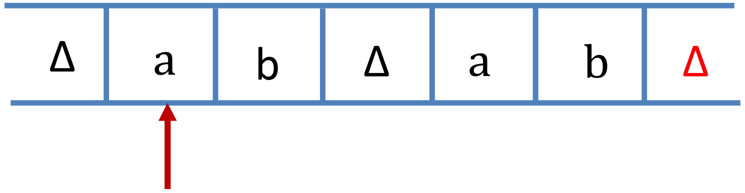
Design a Turing machine to copy a string

String:

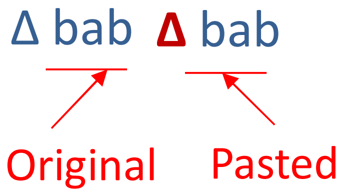


Δ ab **Δ** ab

Original Pasted



Δ	a	b	Δ	a	b	Δ



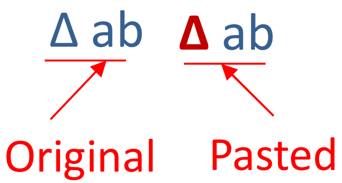
Δ bab **Δ** bab

Original Pasted

Δ	A	B	B	a	Δ	a	b	b	Δ	Δ

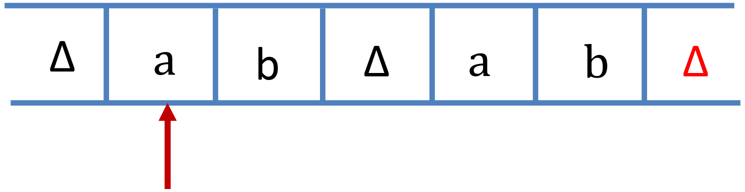
Design a Turing machine to copy a string

String:

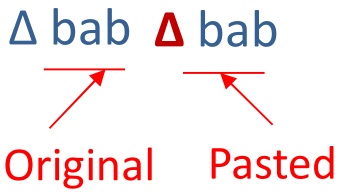


Δ ab **Δ** ab

Original Pasted

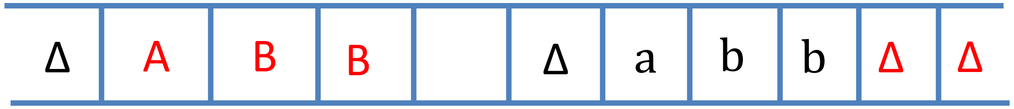


Δ	a	b	Δ	a	b	Δ



Δ bab **Δ** bab

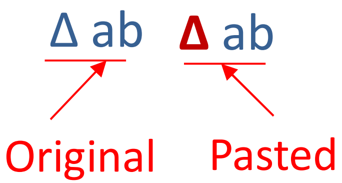
Original Pasted



Δ	A	B	B		Δ	a	b	b	Δ	Δ

Design a Turing machine to copy a string

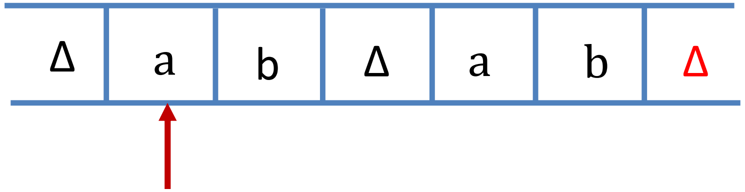
String:



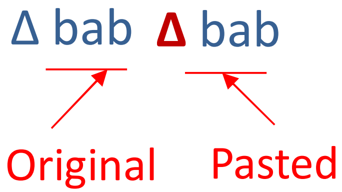
Δ ab **Δ** ab

Original

Pasted



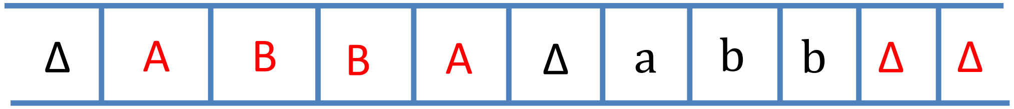
Δ	a	b	Δ	a	b	Δ



Δ bab **Δ** bab

Original

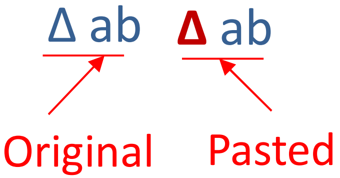
Pasted



Δ	A	B	B	A	Δ	a	b	b	Δ	Δ

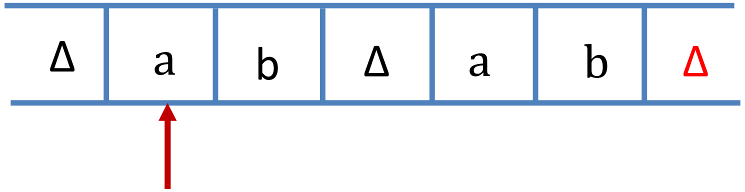
Design a Turing machine to copy a string

String:

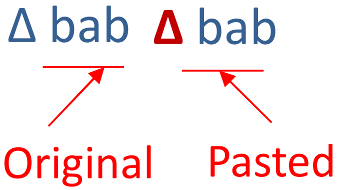


Δ ab **Δ** ab

Original Pasted



Δ	a	b	Δ	a	b	Δ



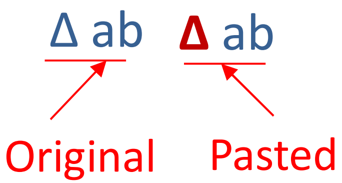
Δ bab **Δ** bab

Original Pasted

Δ	A	B	B	A	Δ	a	b	b		Δ

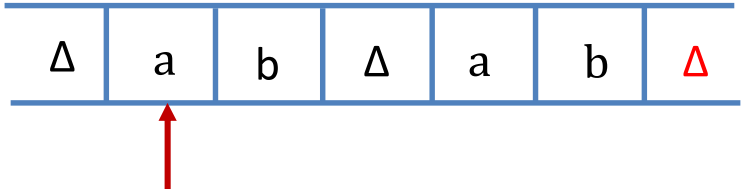
Design a Turing machine to copy a string

String:

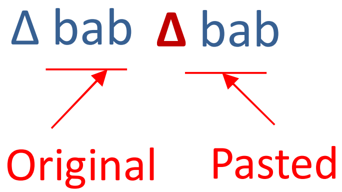


Δ ab **Δ** ab

Original Pasted



Δ	a	b	Δ	a	b	Δ



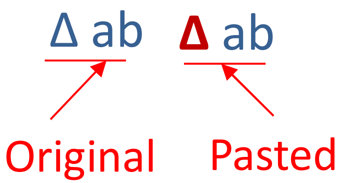
Δ bab **Δ** bab

Original Pasted

Δ	A	B	B	A	Δ	a	b	b	a	Δ

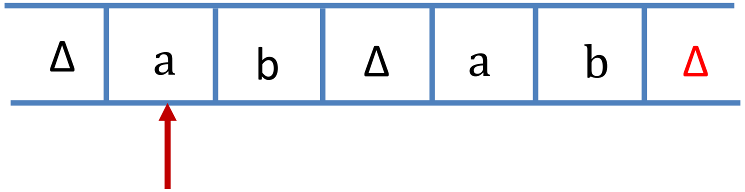
Design a Turing machine to copy a string

String:

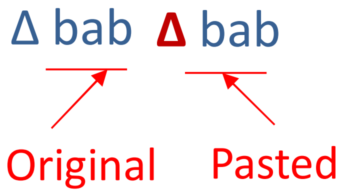


Δ ab **Δ** ab

Original Pasted

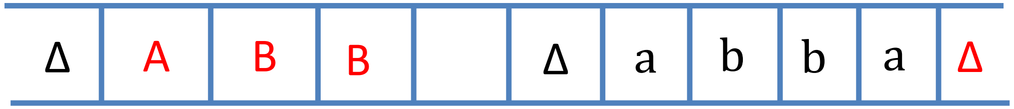


Δ	a	b	Δ	a	b	Δ



Δ bab **Δ** bab

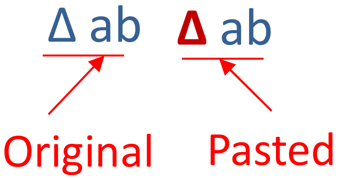
Original Pasted



Δ	A	B	B		Δ	a	b	b	a	Δ

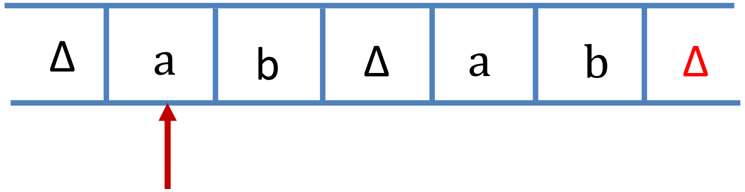
Design a Turing machine to copy a string

String:

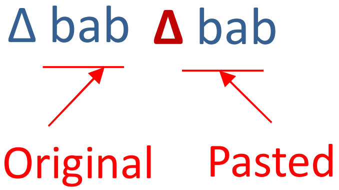


Δ ab Δ ab

Original Pasted

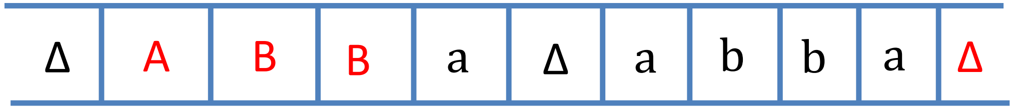


Δ	a	b	Δ	a	b	Δ



Δ bab Δ bab

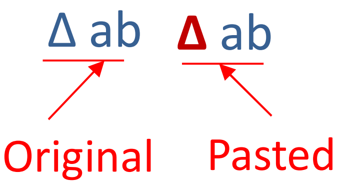
Original Pasted



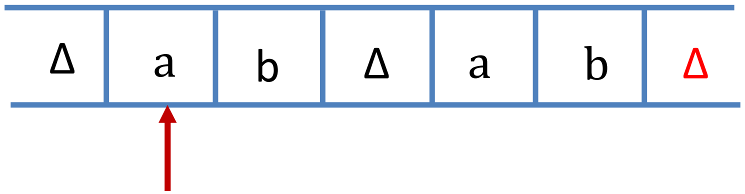
Δ	A	B	B	a	Δ	a	b	b	a	Δ

Design a Turing machine to copy a string

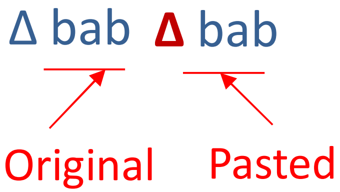
String:


Δ ab **Δ** ab

Original Pasted



Δ	a	b	Δ	a	b	Δ


Δ bab **Δ** bab

Original Pasted

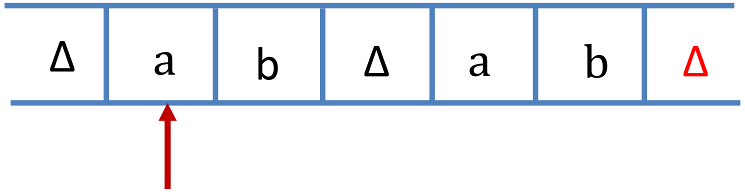
Δ	A	B		a	Δ	a	b	b	a	Δ

Design a Turing machine to copy a string

String:

Δ ab **Δ** ab

Original Pasted



Δ	a	b	Δ	a	b	Δ

Δ bab **Δ** bab

Original Pasted

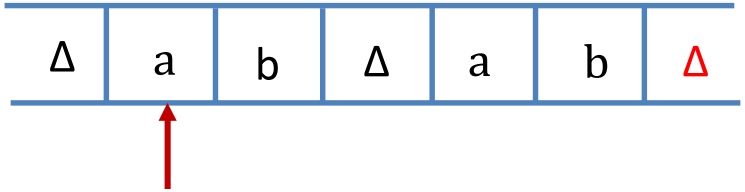
Δ	A	B	b	a	Δ	a	b	b	a	Δ

Design a Turing machine to copy a string

String:

Δ ab **Δ** ab

Original Pasted



Δ	a	b	Δ	a	b	Δ

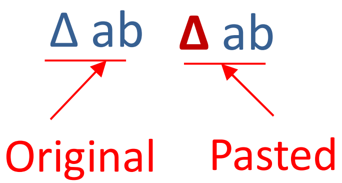
Δ bab **Δ** bab

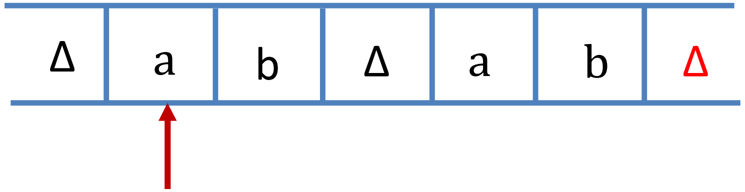
Original Pasted

Δ	A		b	a	Δ	a	b	b	a	Δ

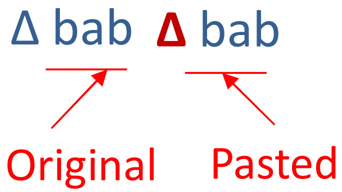
Design a Turing machine to copy a string

String:





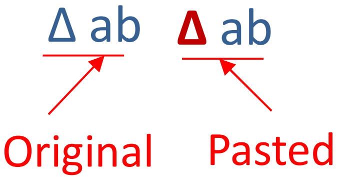
Δ	a	b	Δ	a	b	Δ



Δ	A	b	b	a	Δ	a	b	b	a	Δ

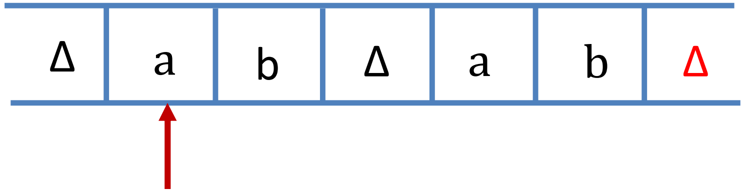
Design a Turing machine to copy a string

String:

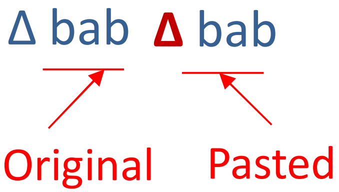

Δ ab Δ ab

Original

Pasted

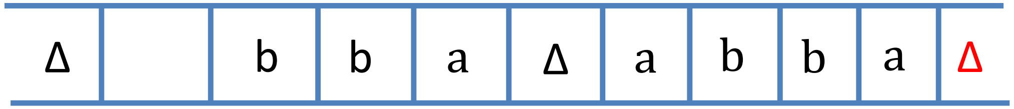


Δ	a	b	Δ	a	b	Δ


Δ bab Δ bab

Original

Pasted



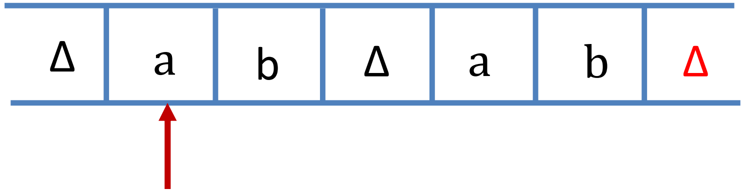
Δ		b	b	a	Δ	a	b	b	a	Δ

Design a Turing machine to copy a string

String:

Δ ab **Δ** ab

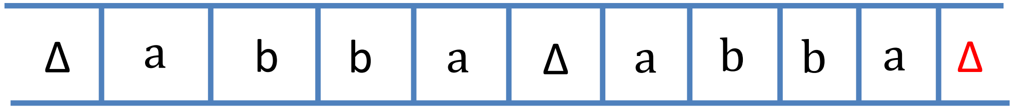
Original Pasted



Δ	a	b	Δ	a	b	Δ

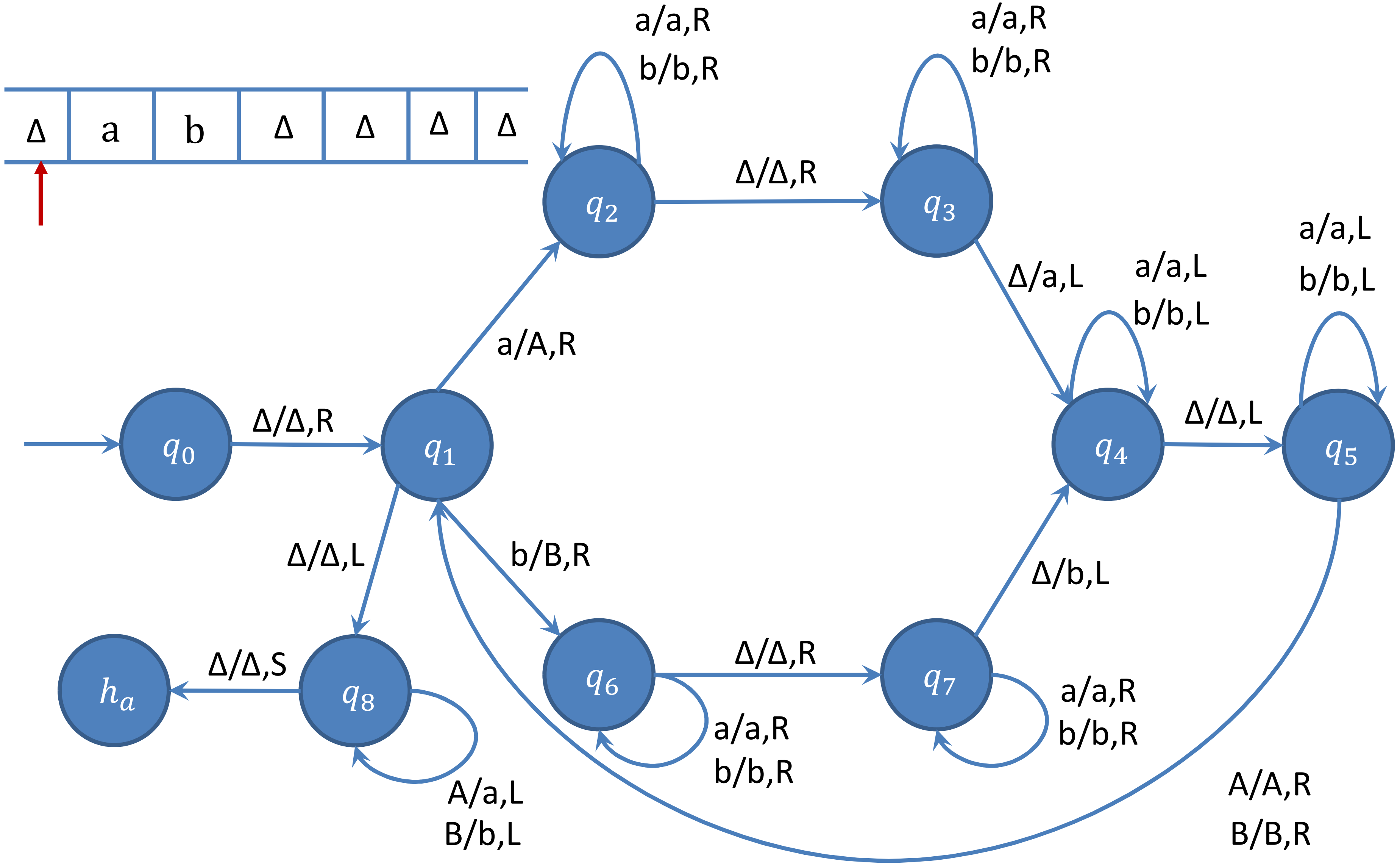
Δ bab **Δ** bab

Original Pasted

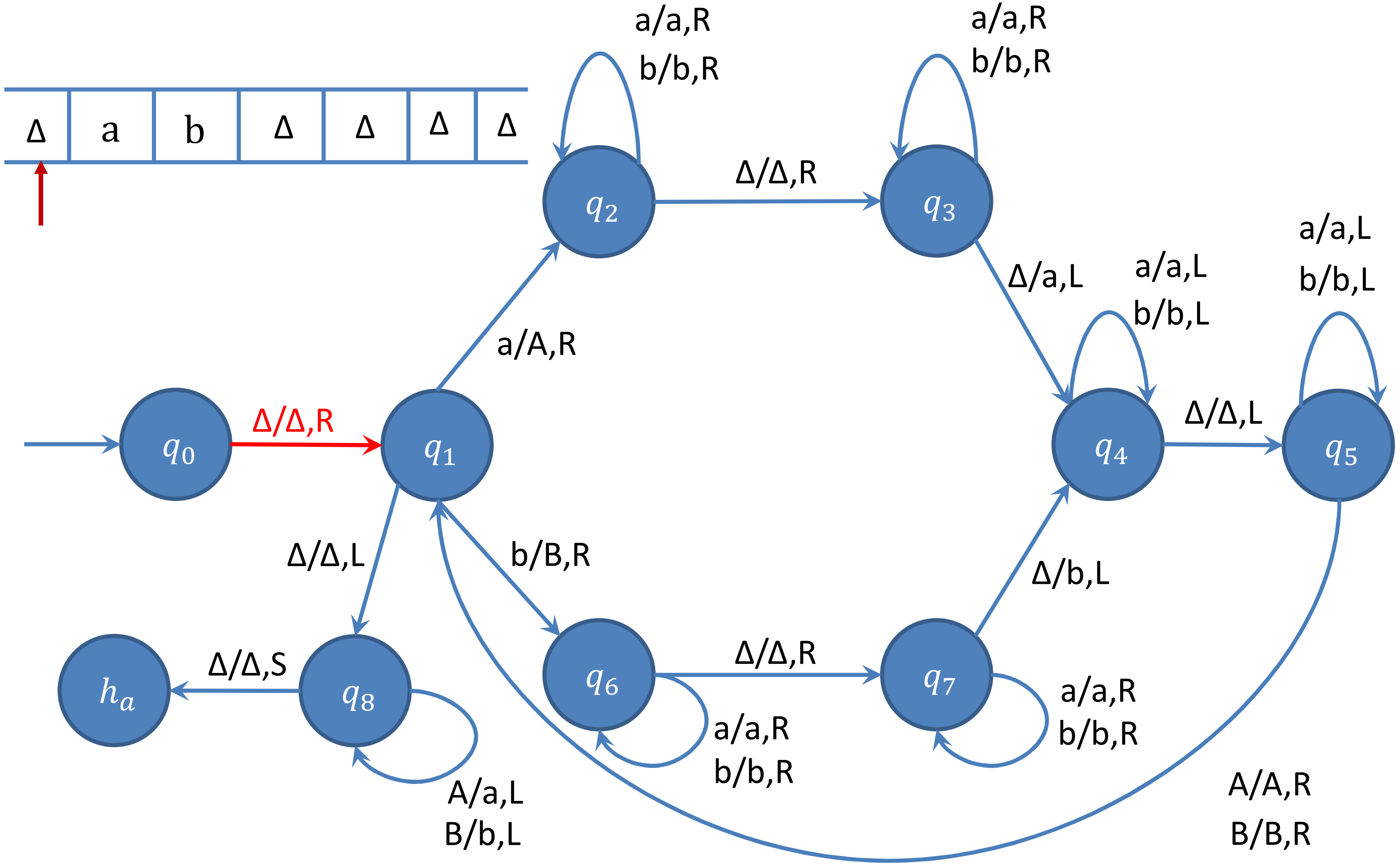


Δ	a	b	b	a	Δ	a	b	b	a	Δ

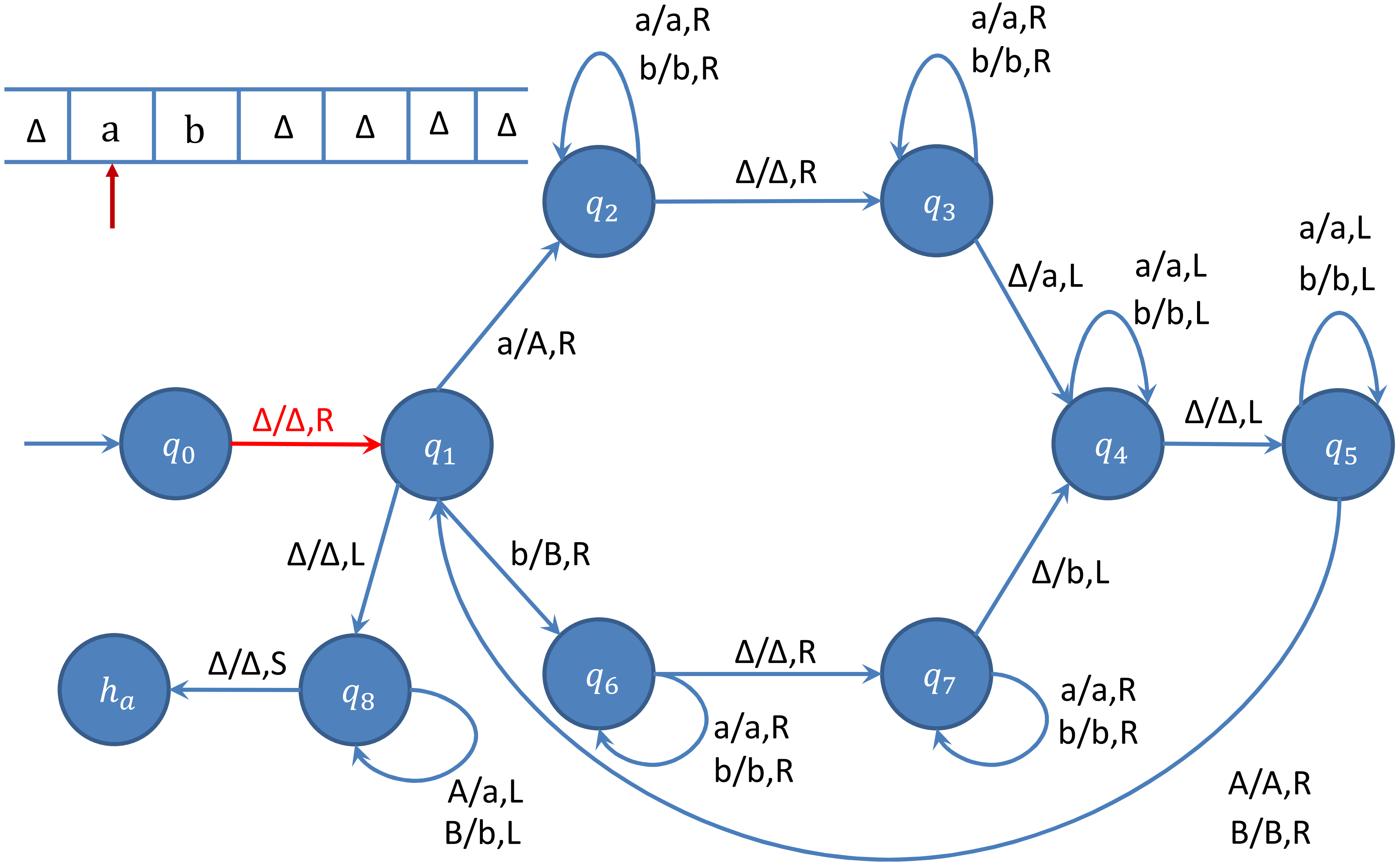
Design a Turing machine to copy a string



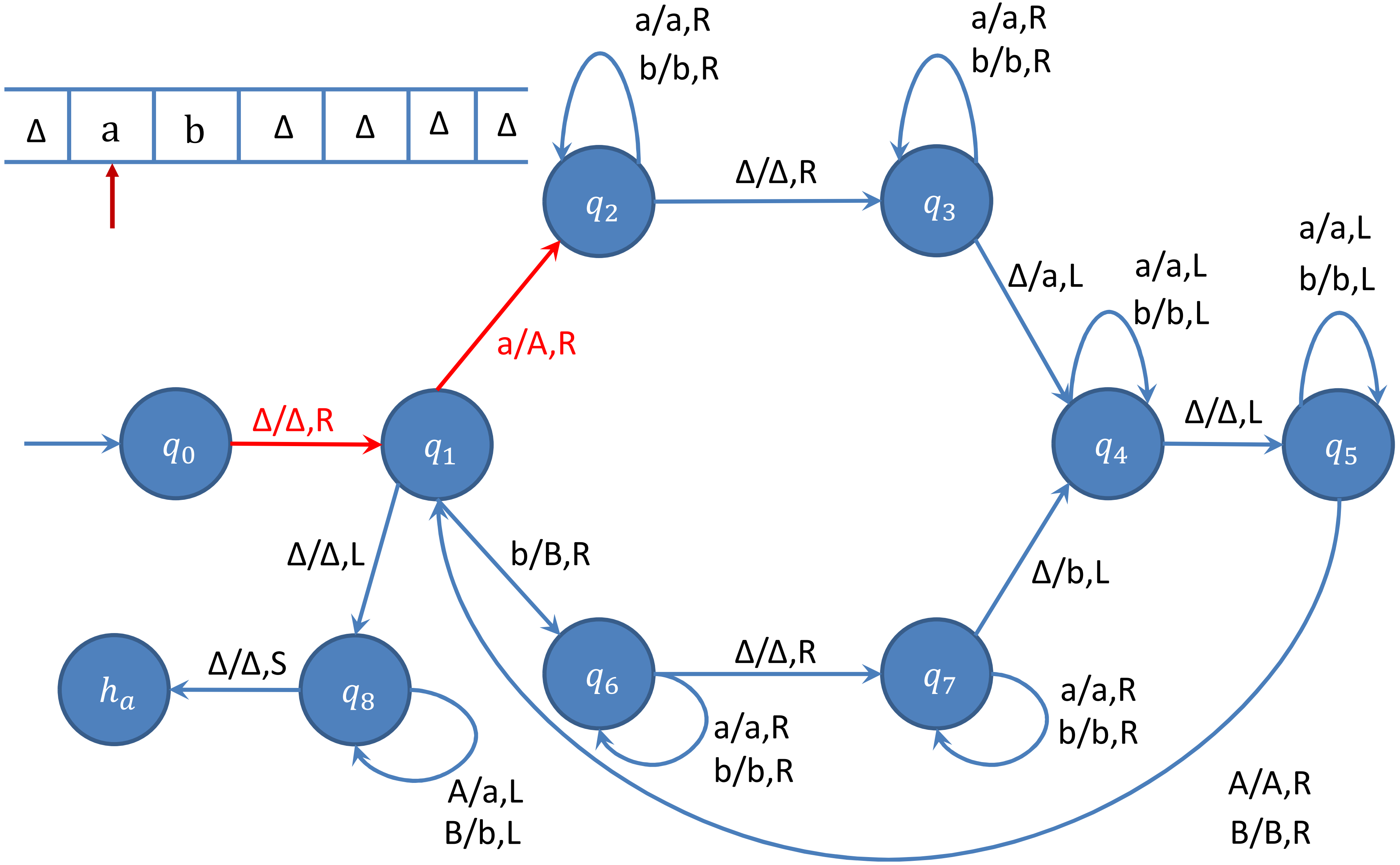
Design a Turing machine to copy a string



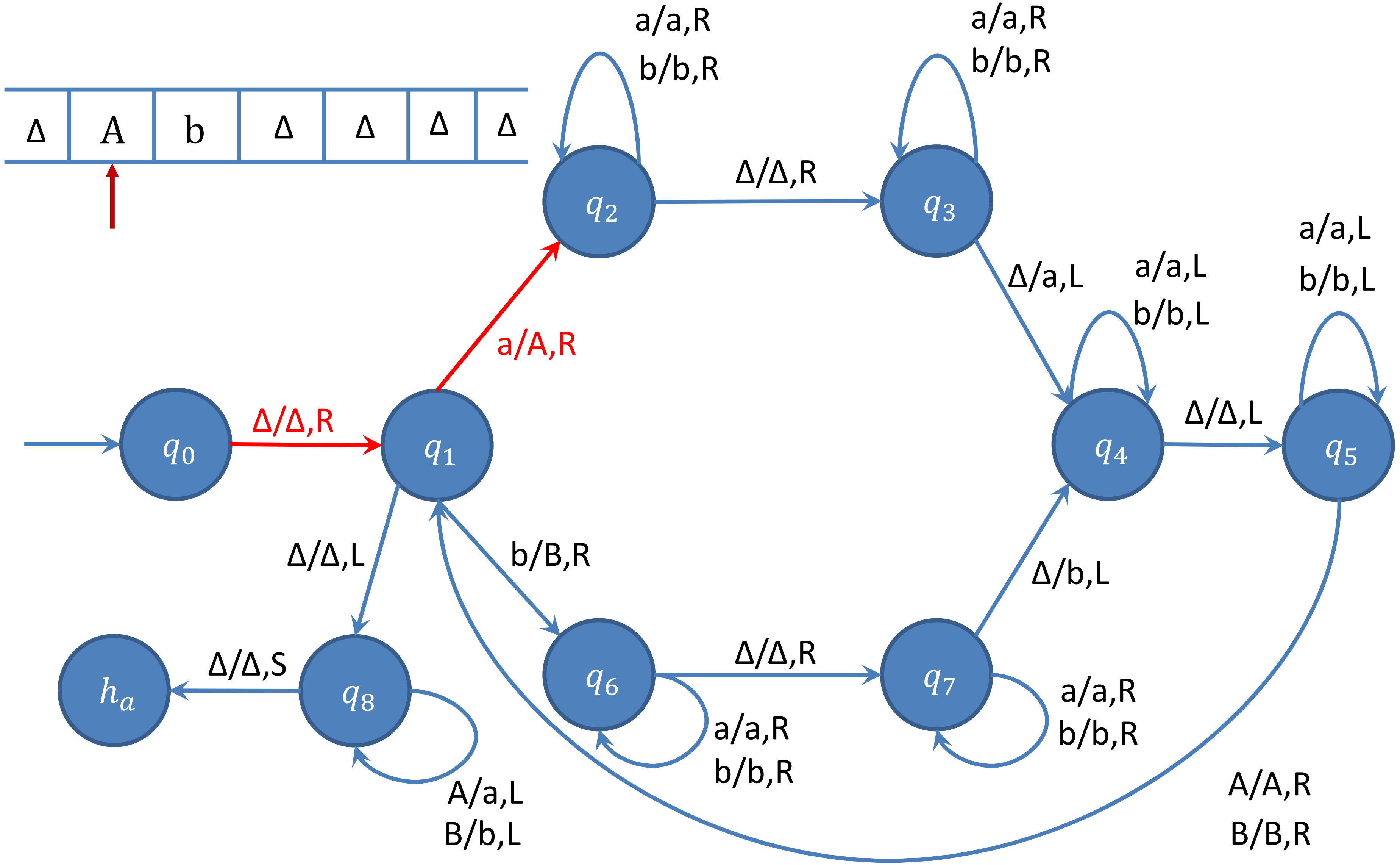
Design a Turing machine to copy a string



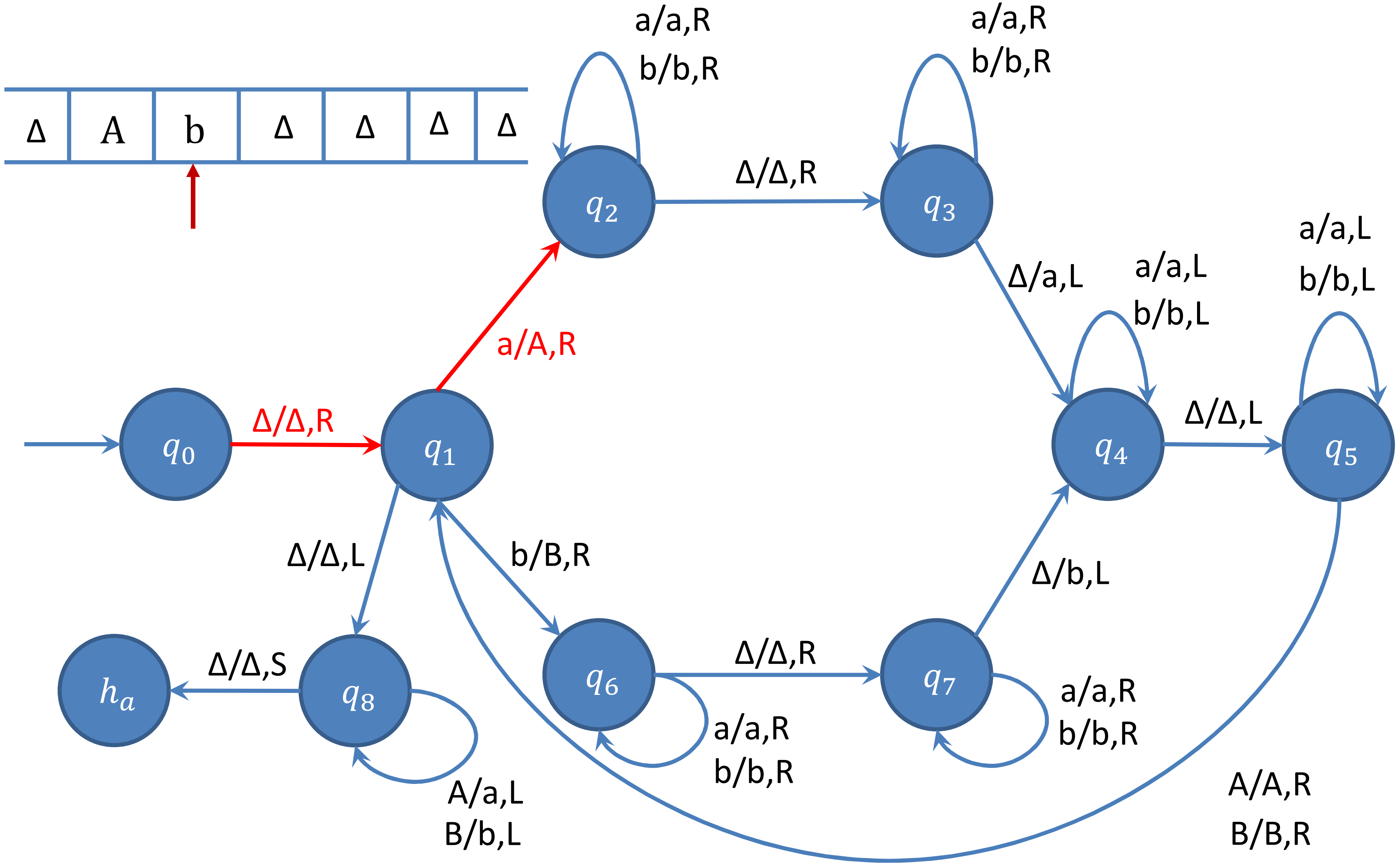
Design a Turing machine to copy a string



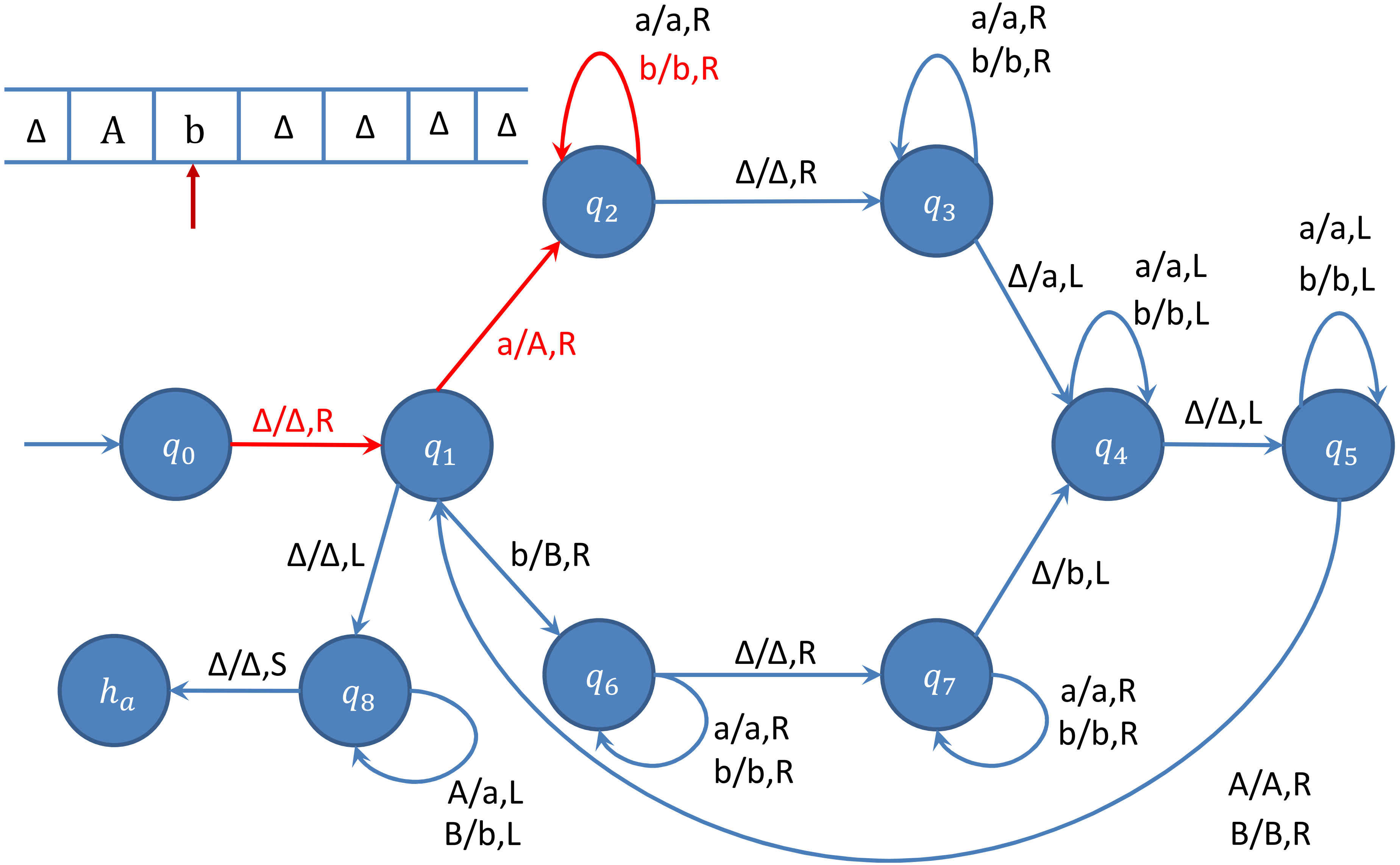
Design a Turing machine to copy a string



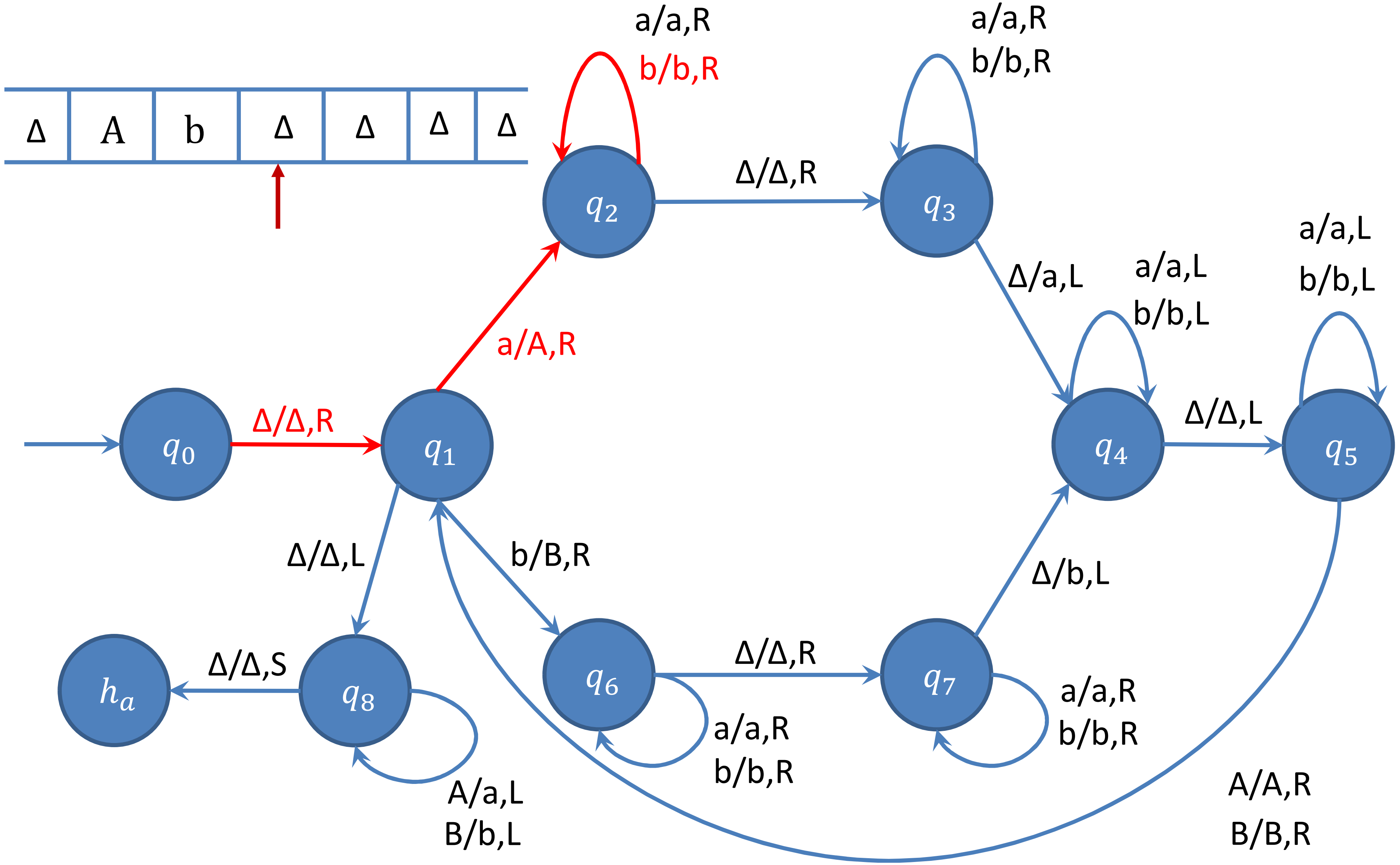
Design a Turing machine to copy a string



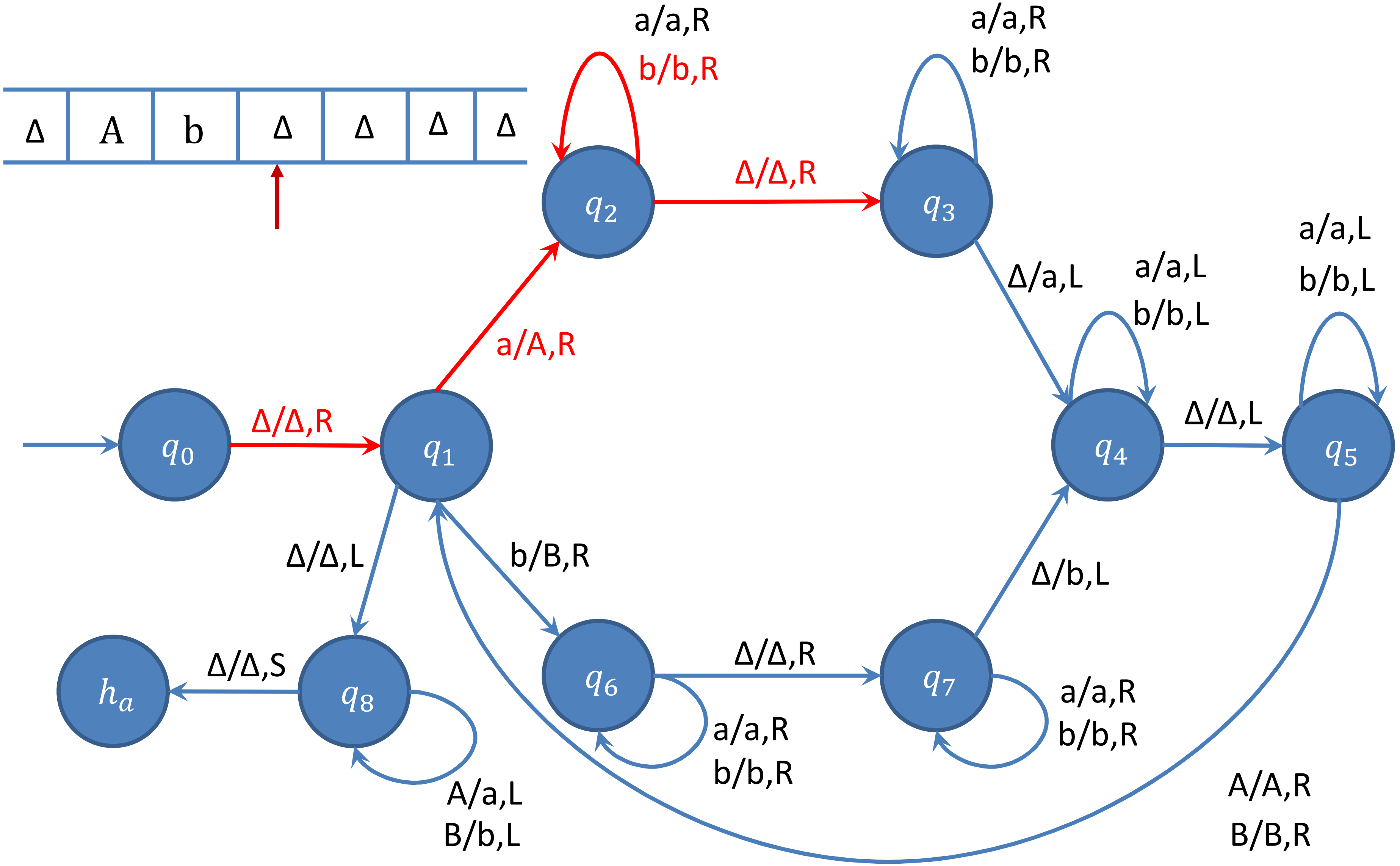
Design a Turing machine to copy a string



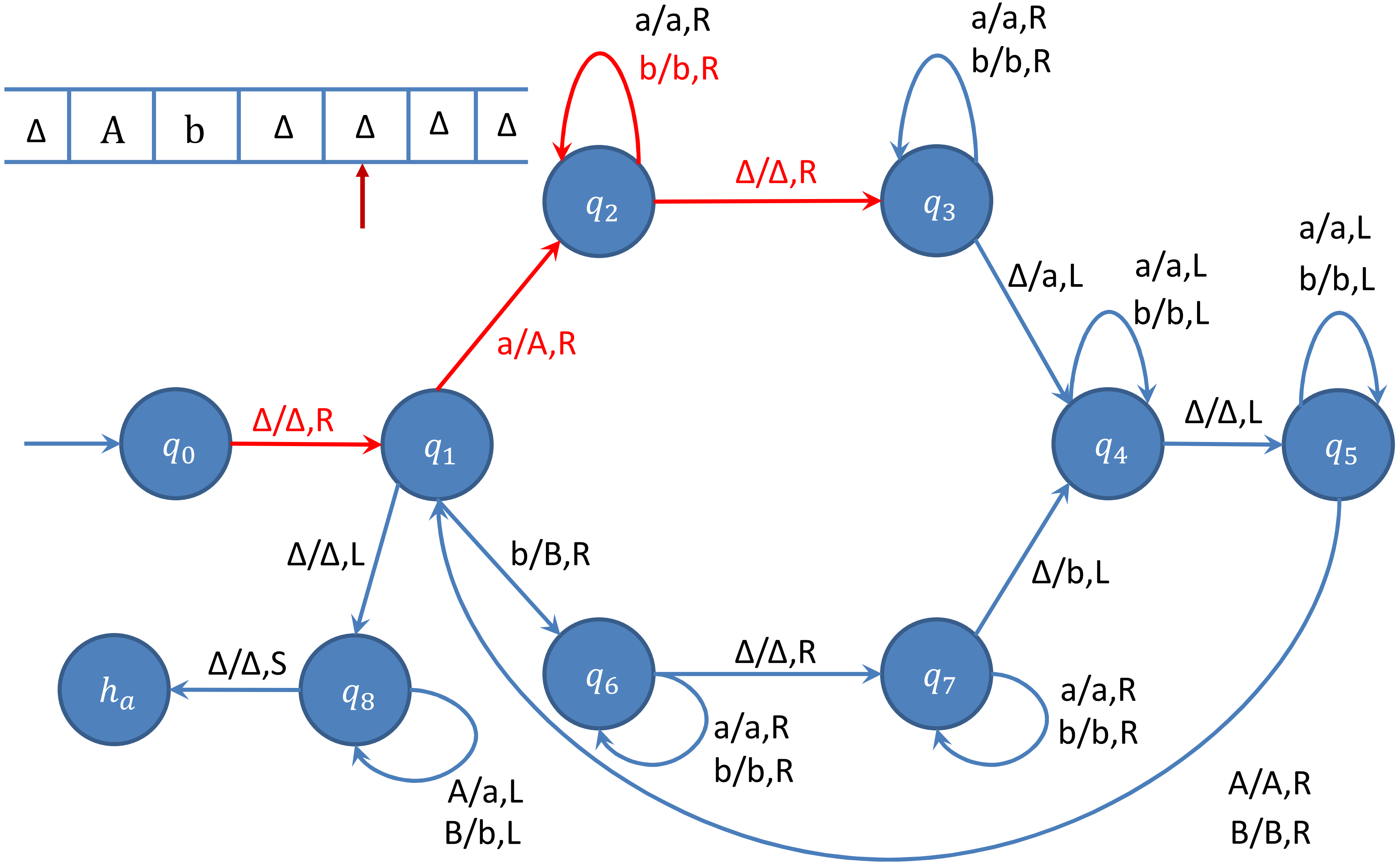
Design a Turing machine to copy a string



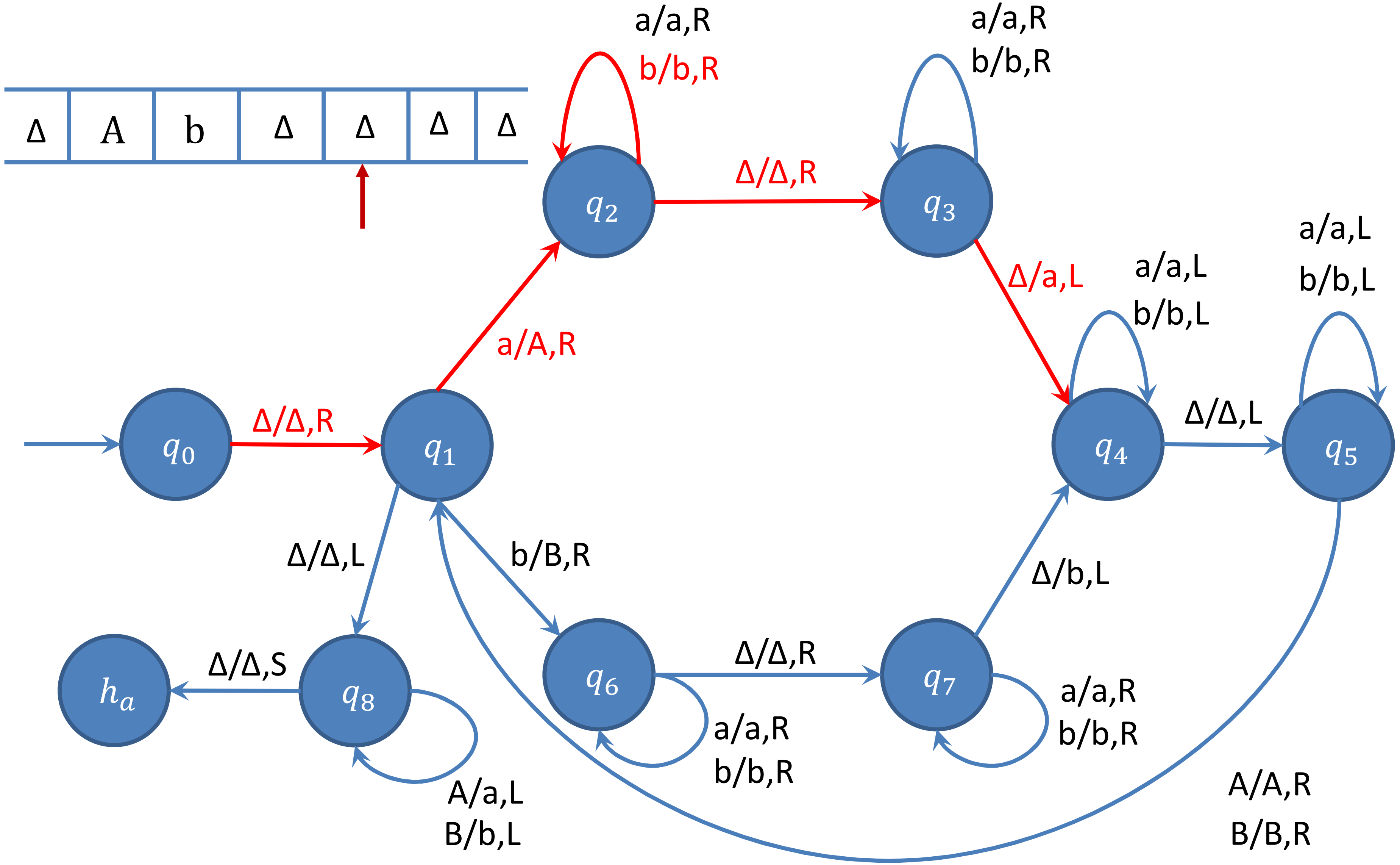
Design a Turing machine to copy a string



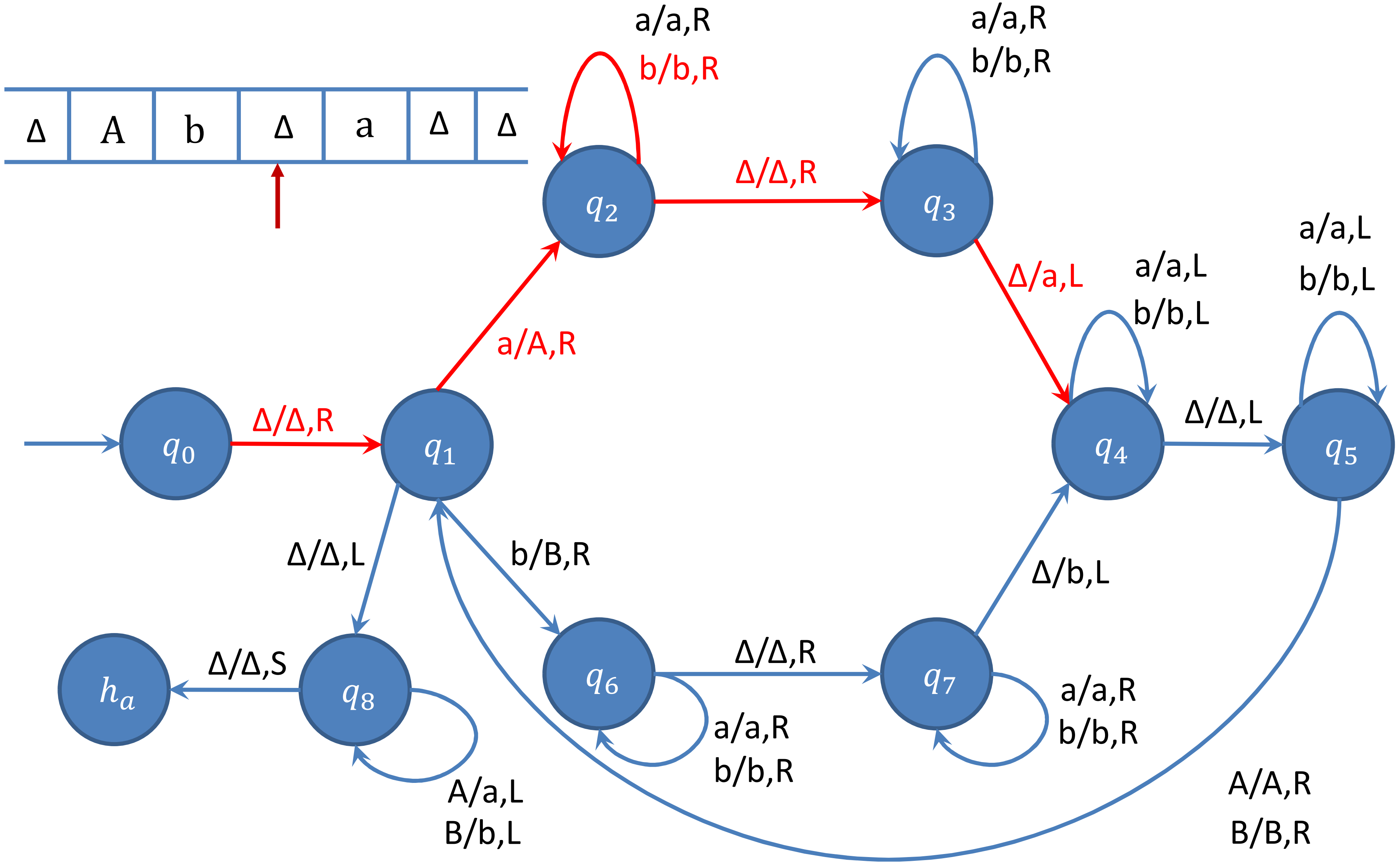
Design a Turing machine to copy a string



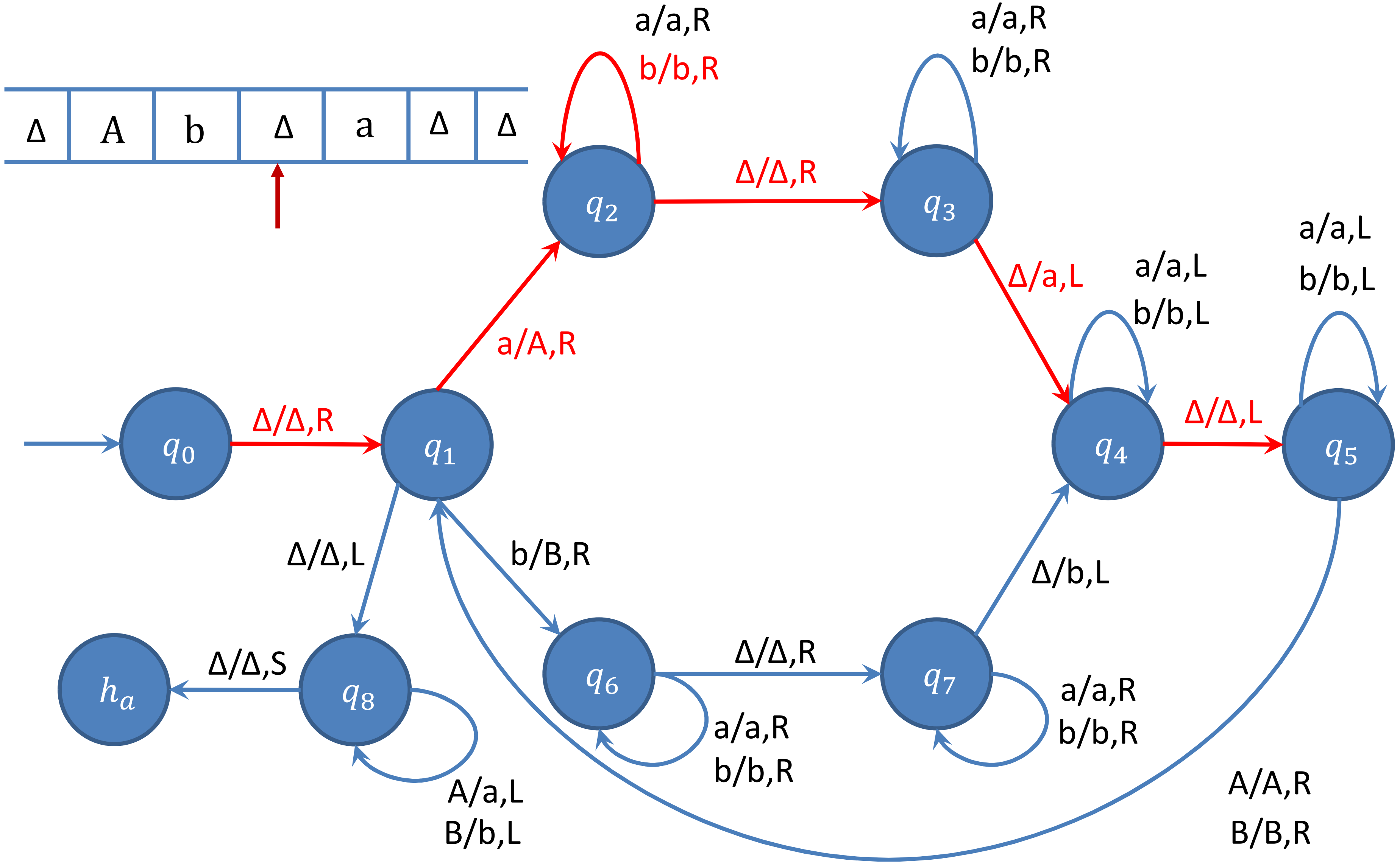
Design a Turing machine to copy a string



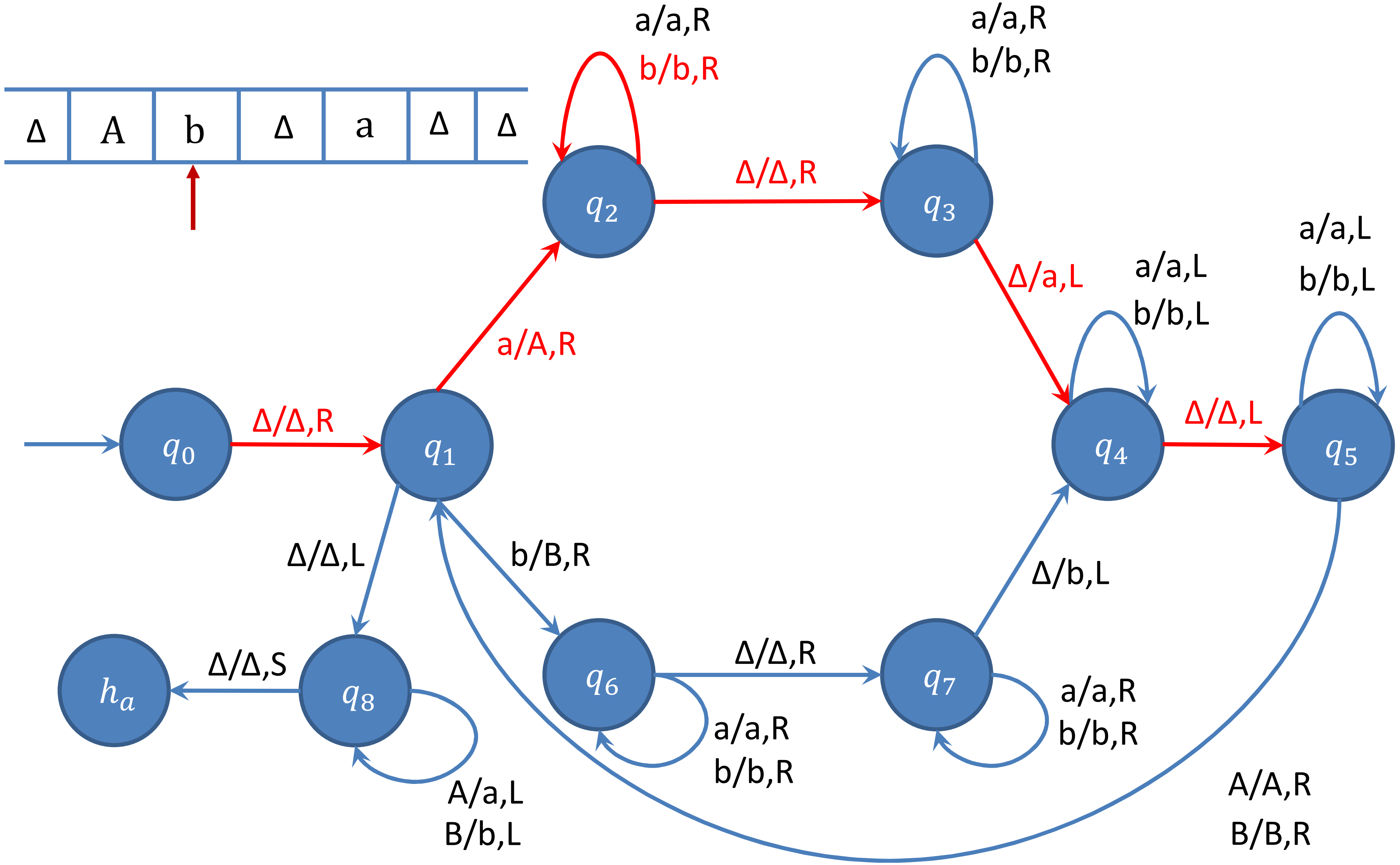
Design a Turing machine to copy a string



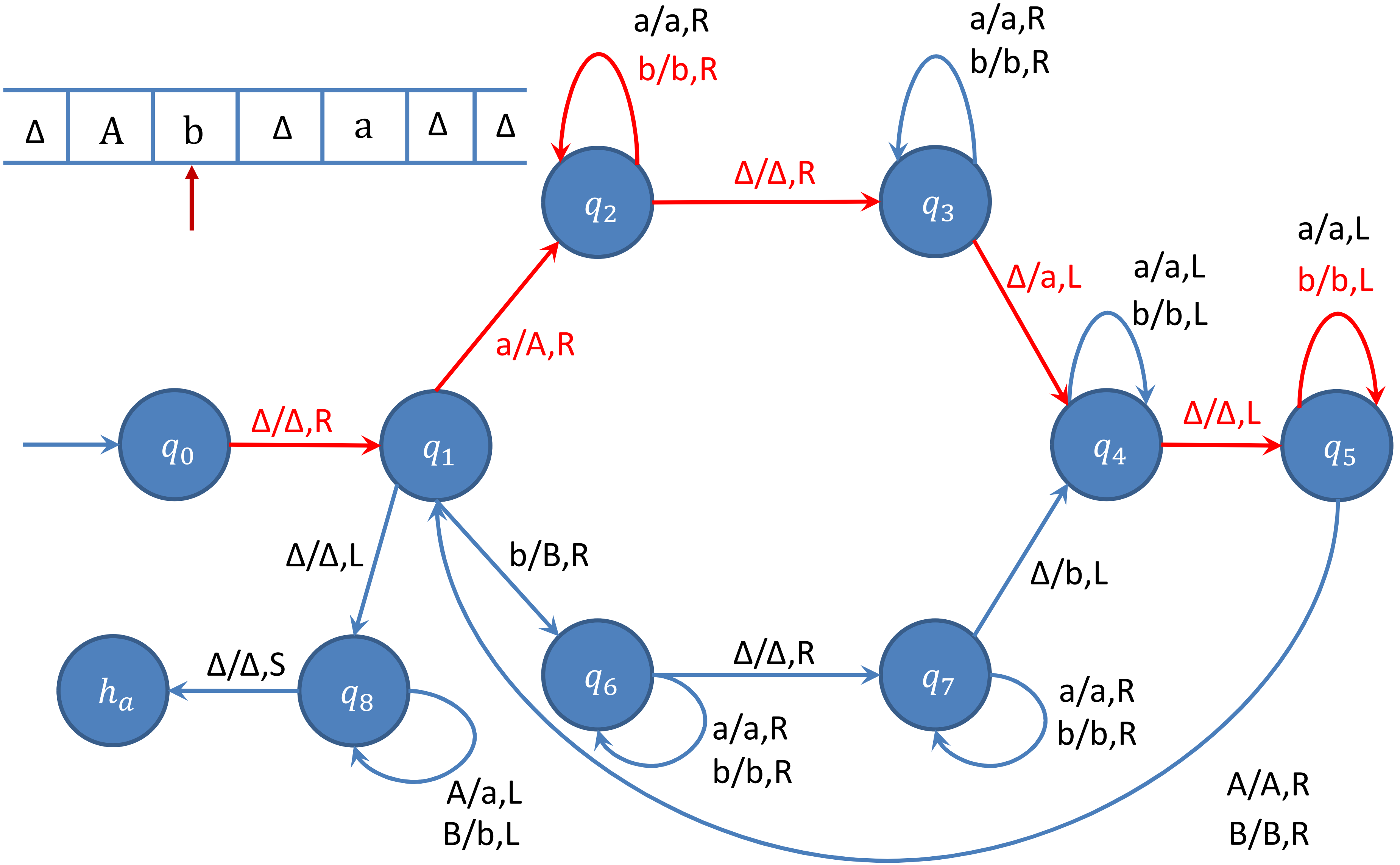
Design a Turing machine to copy a string



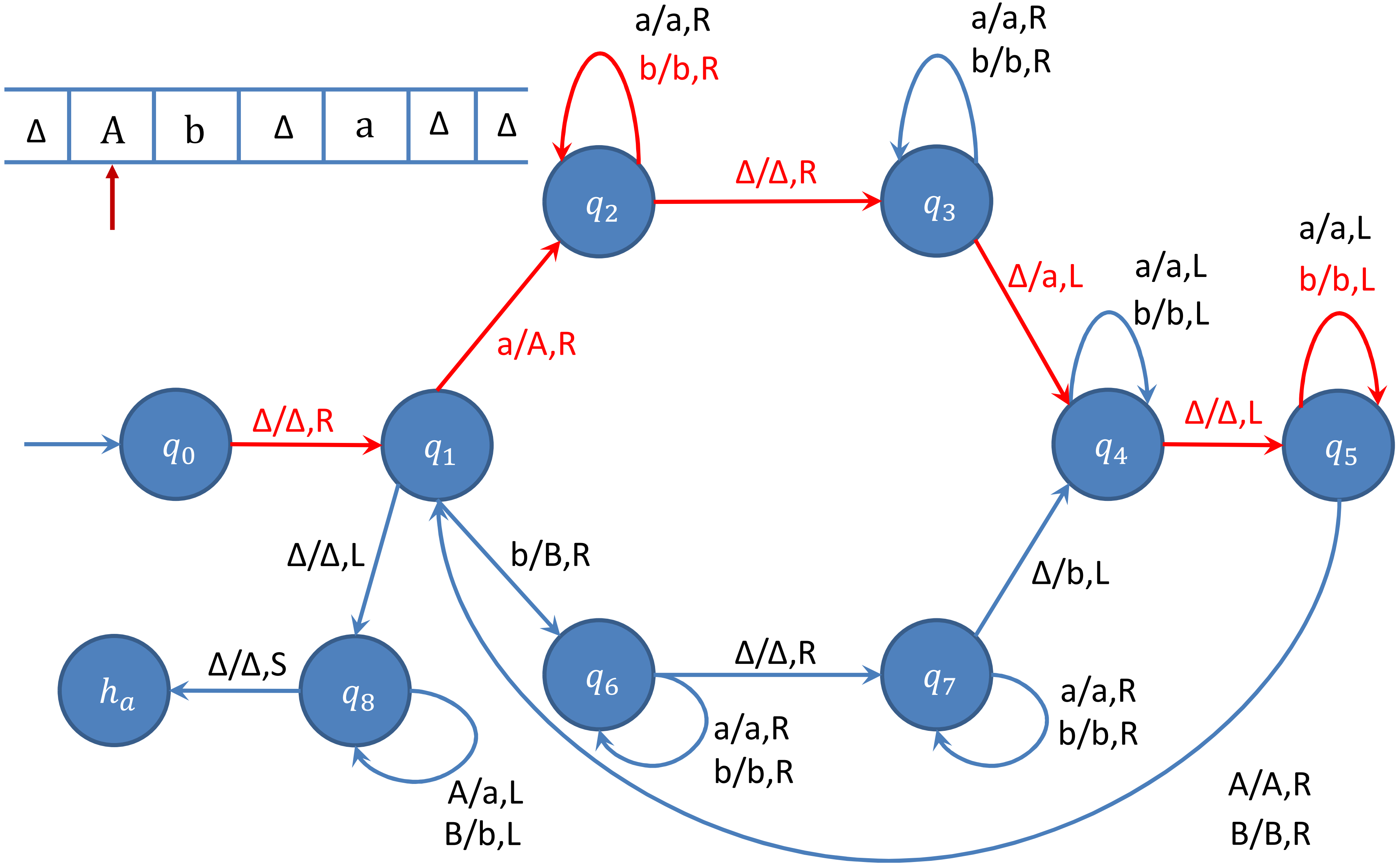
Design a Turing machine to copy a string



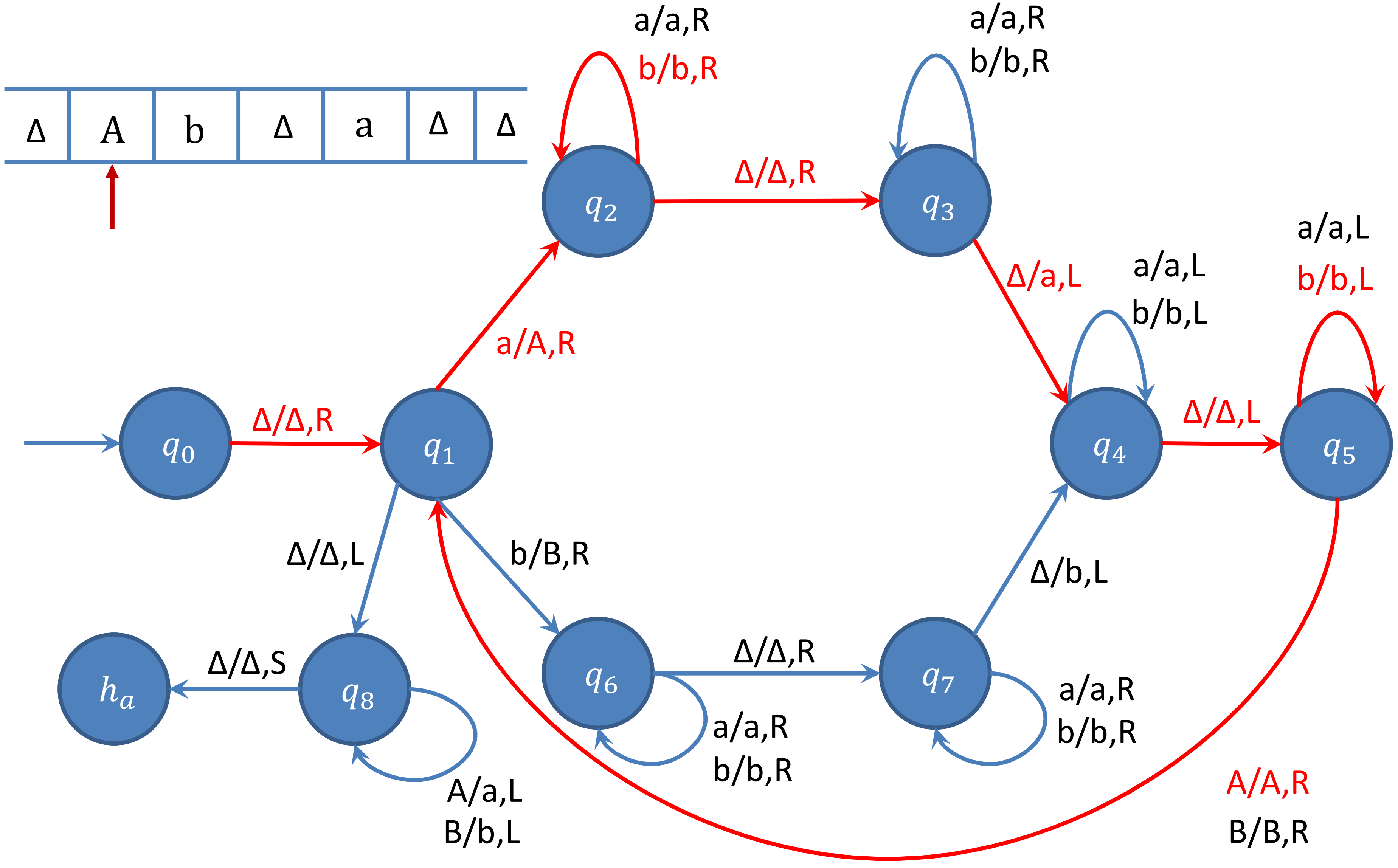
Design a Turing machine to copy a string



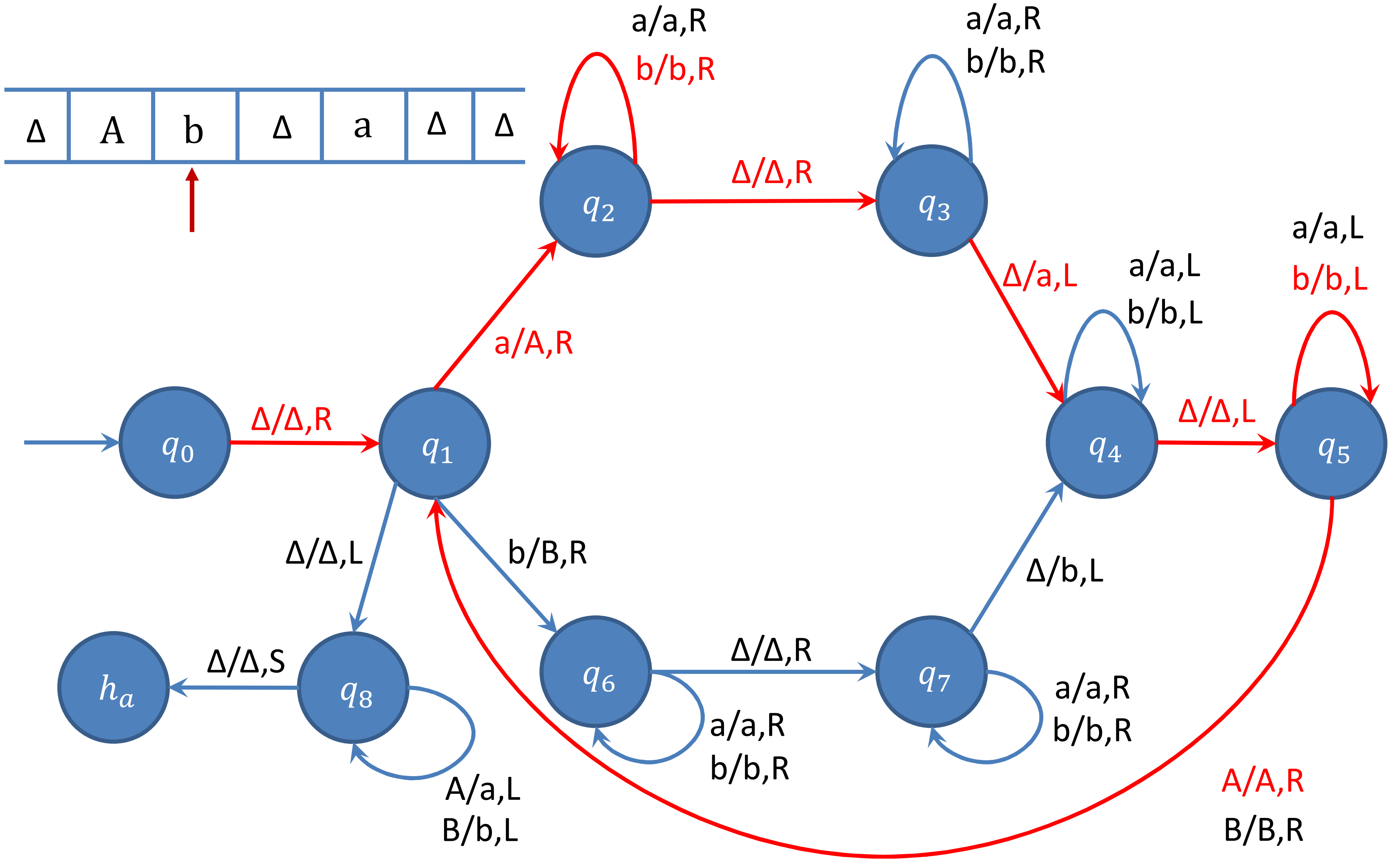
Design a Turing machine to copy a string



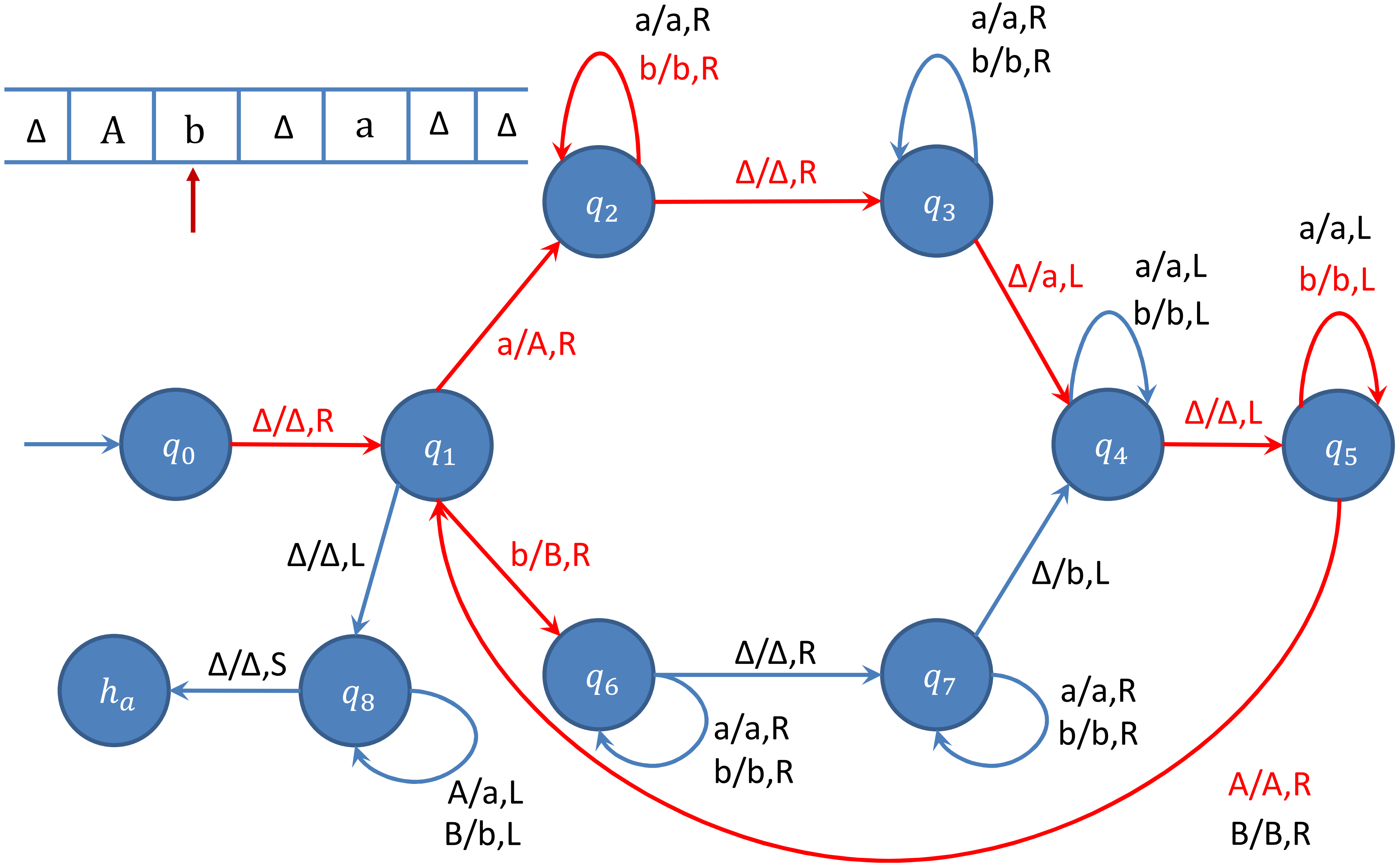
Design a Turing machine to copy a string



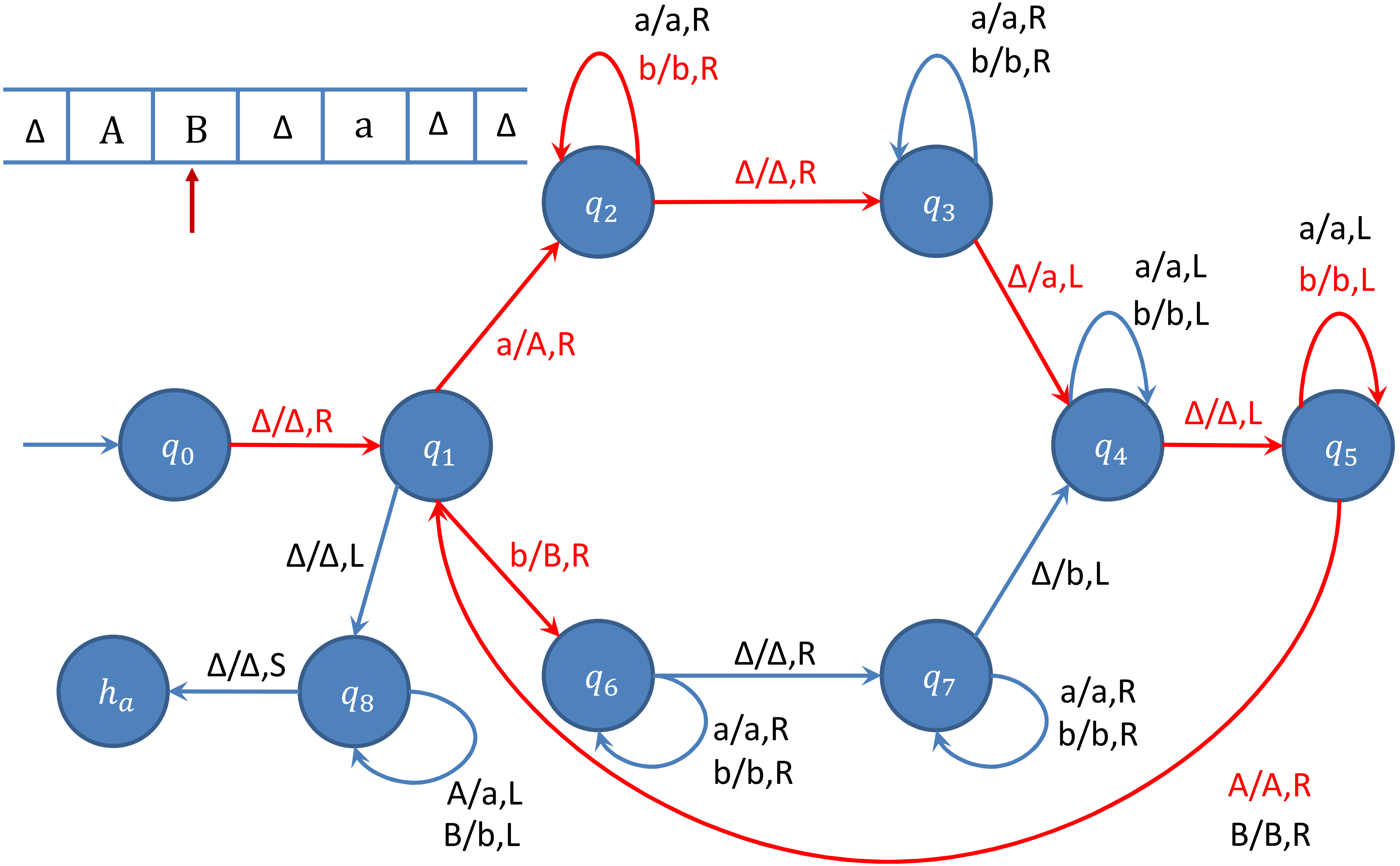
Design a Turing machine to copy a string



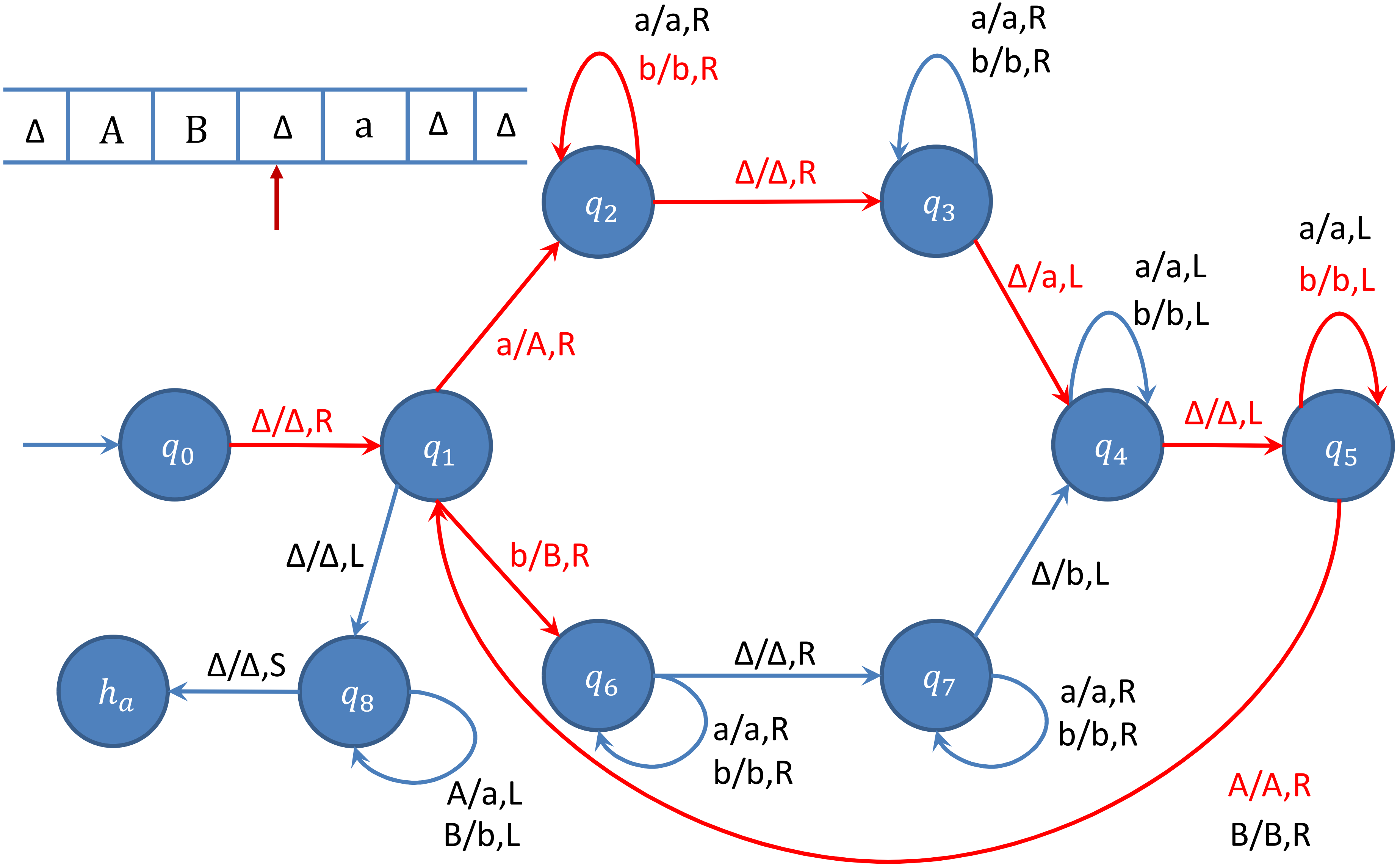
Design a Turing machine to copy a string



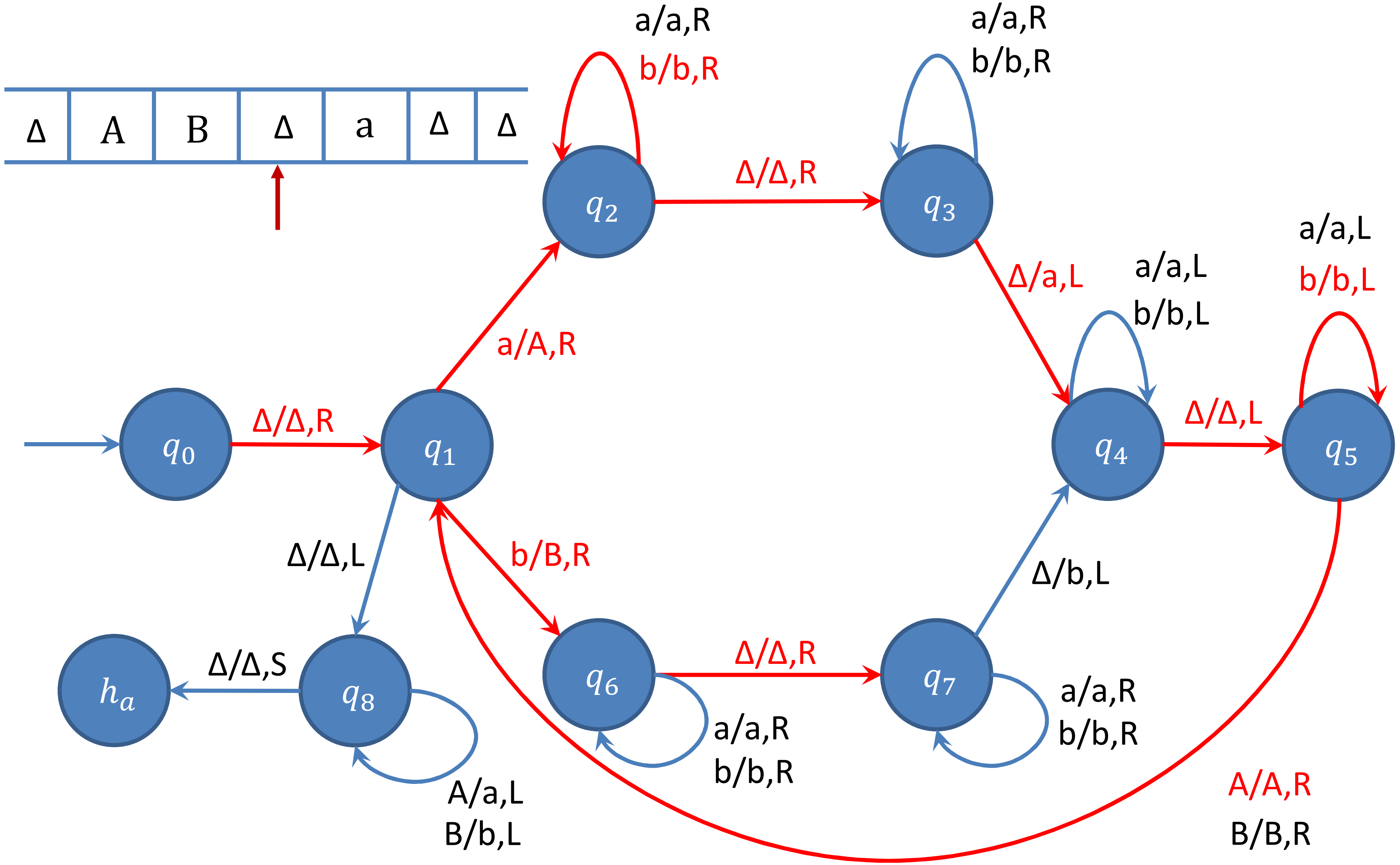
Design a Turing machine to copy a string



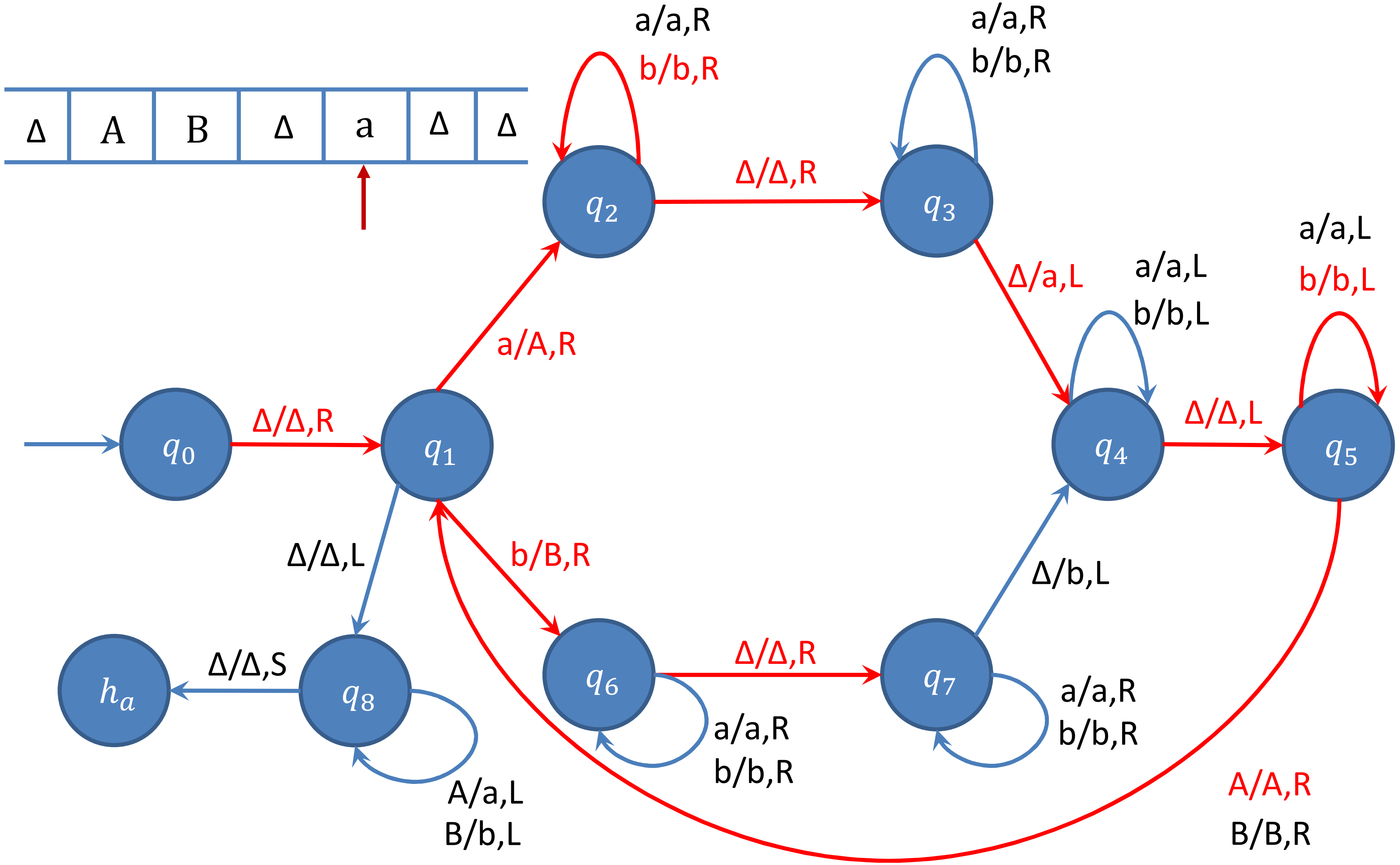
Design a Turing machine to copy a string



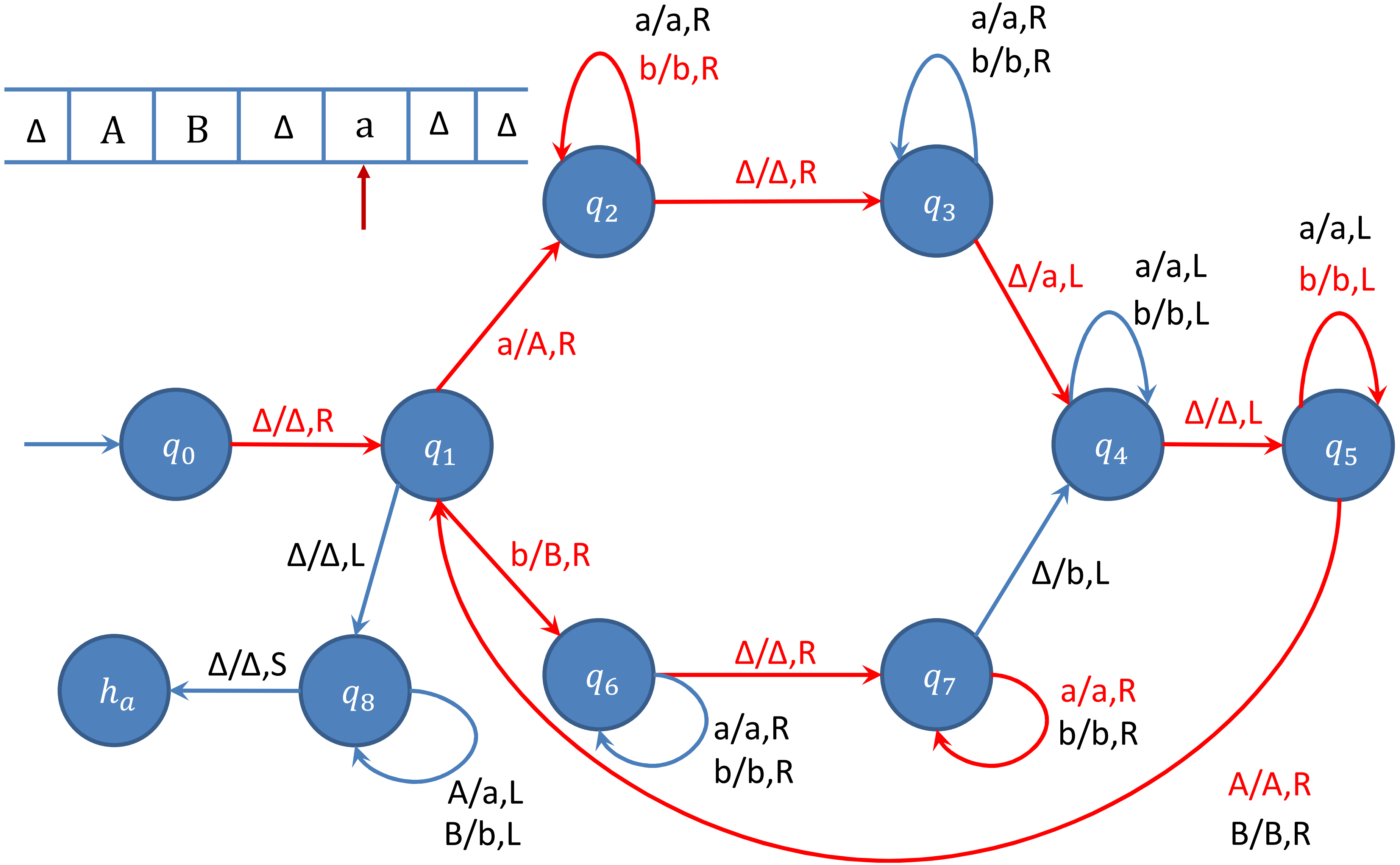
Design a Turing machine to copy a string



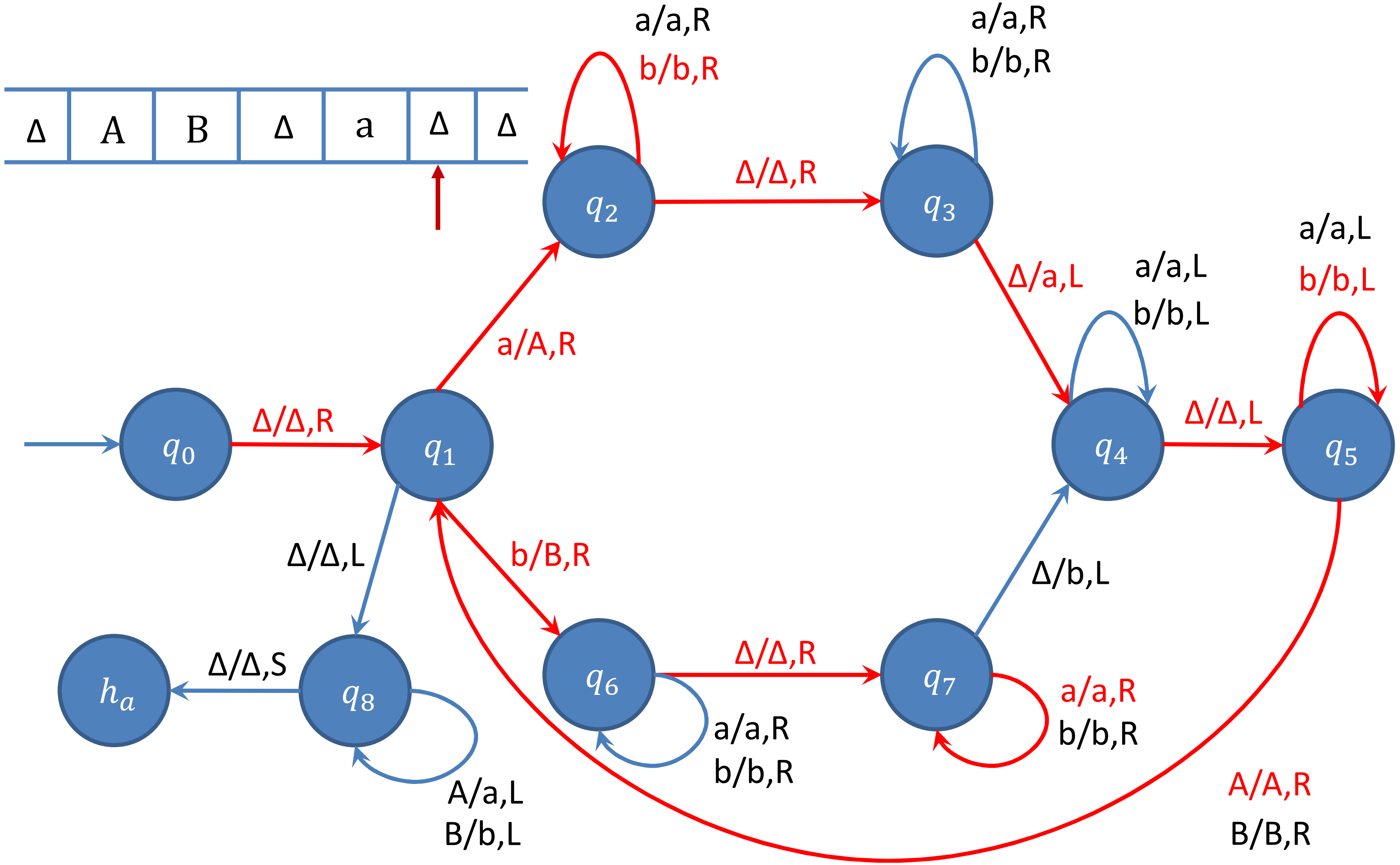
Design a Turing machine to copy a string



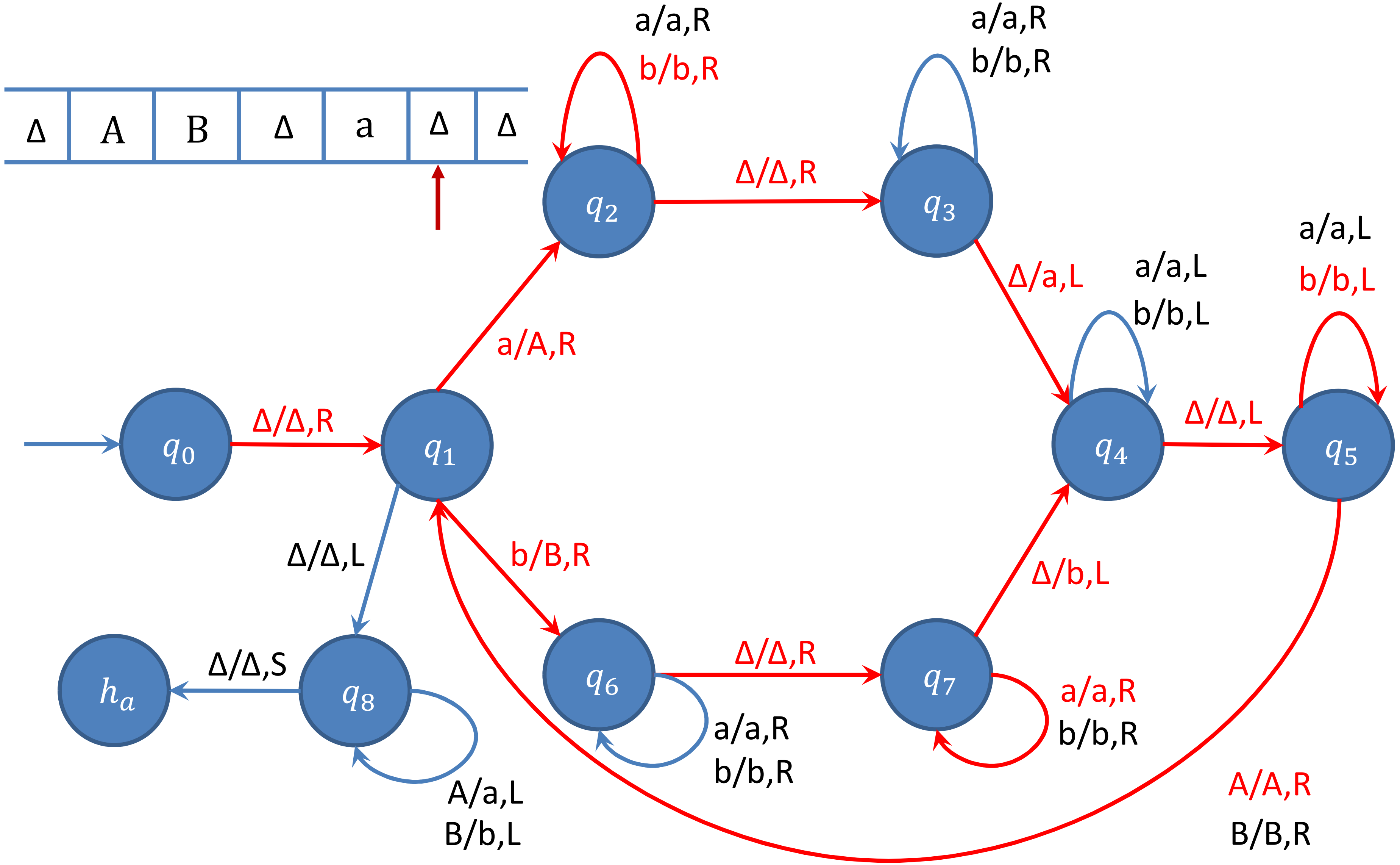
Design a Turing machine to copy a string



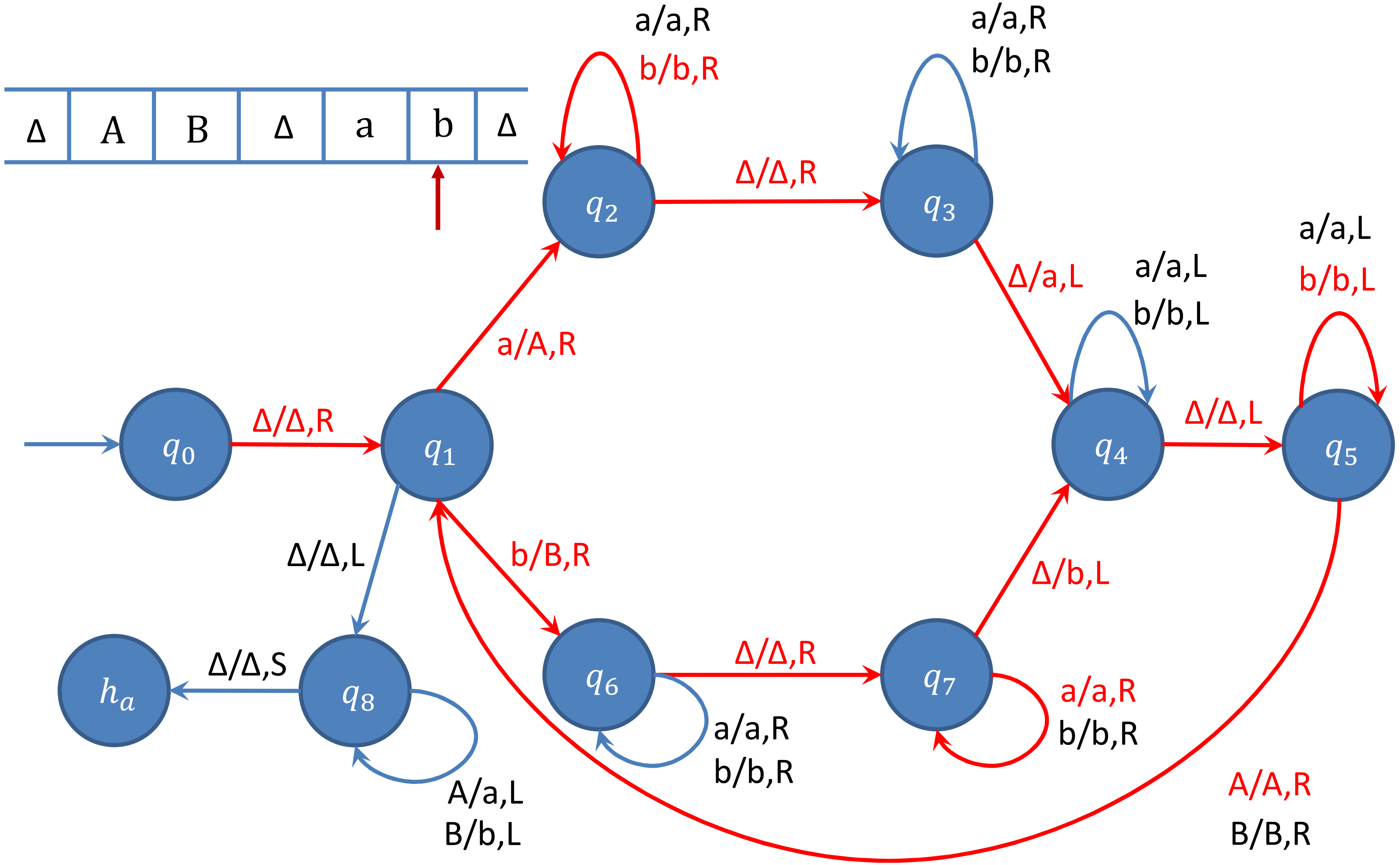
Design a Turing machine to copy a string



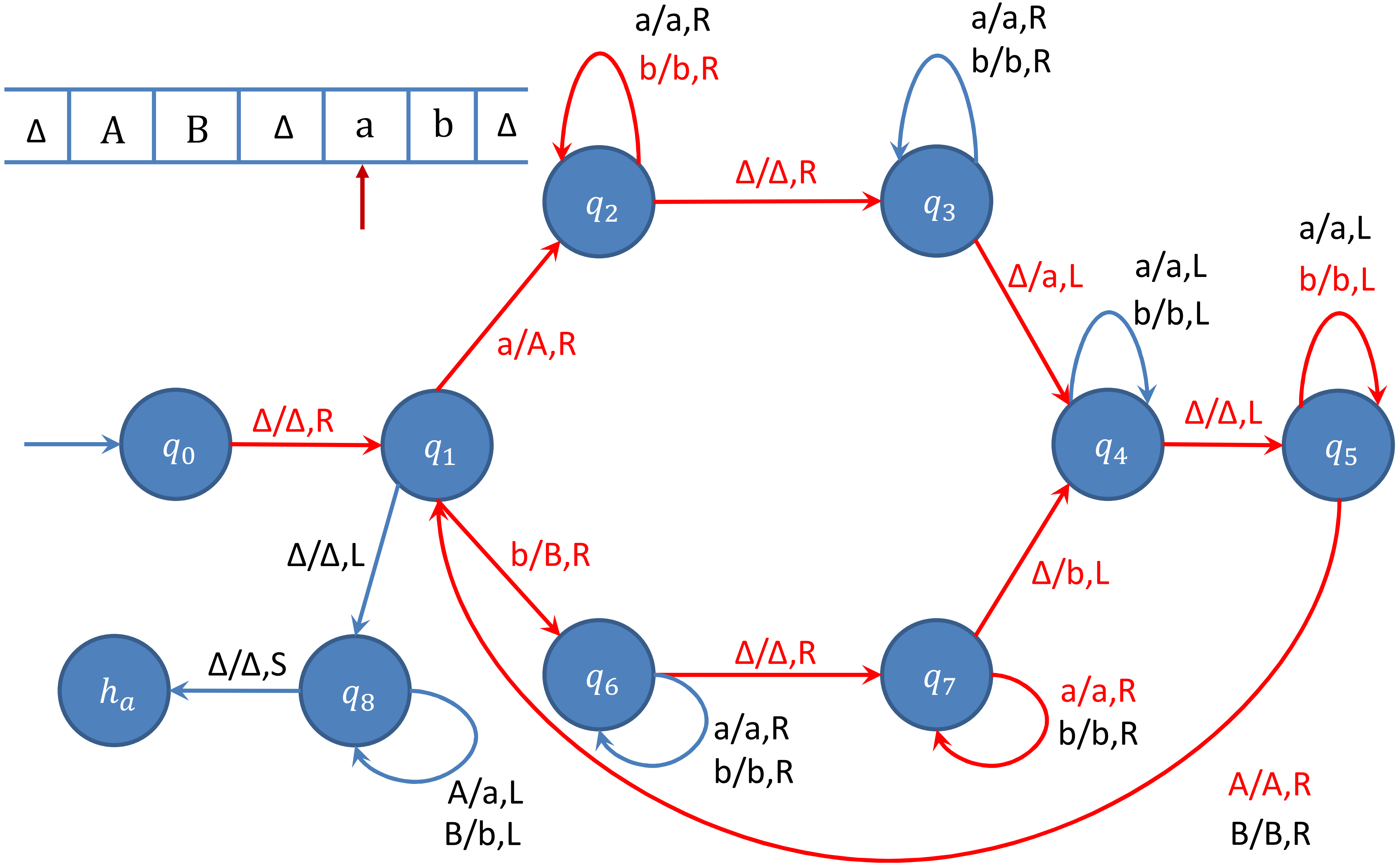
Design a Turing machine to copy a string



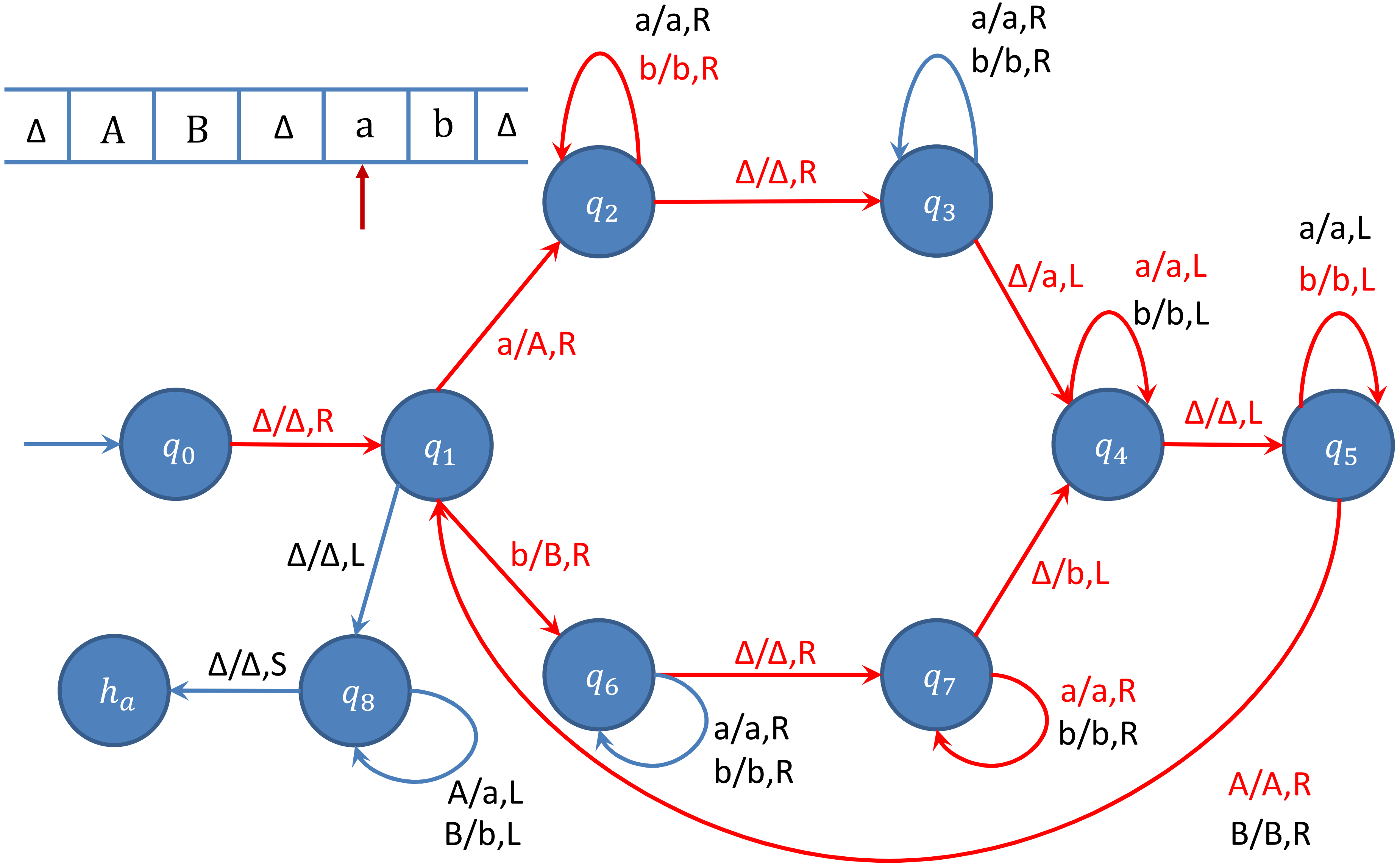
Design a Turing machine to copy a string



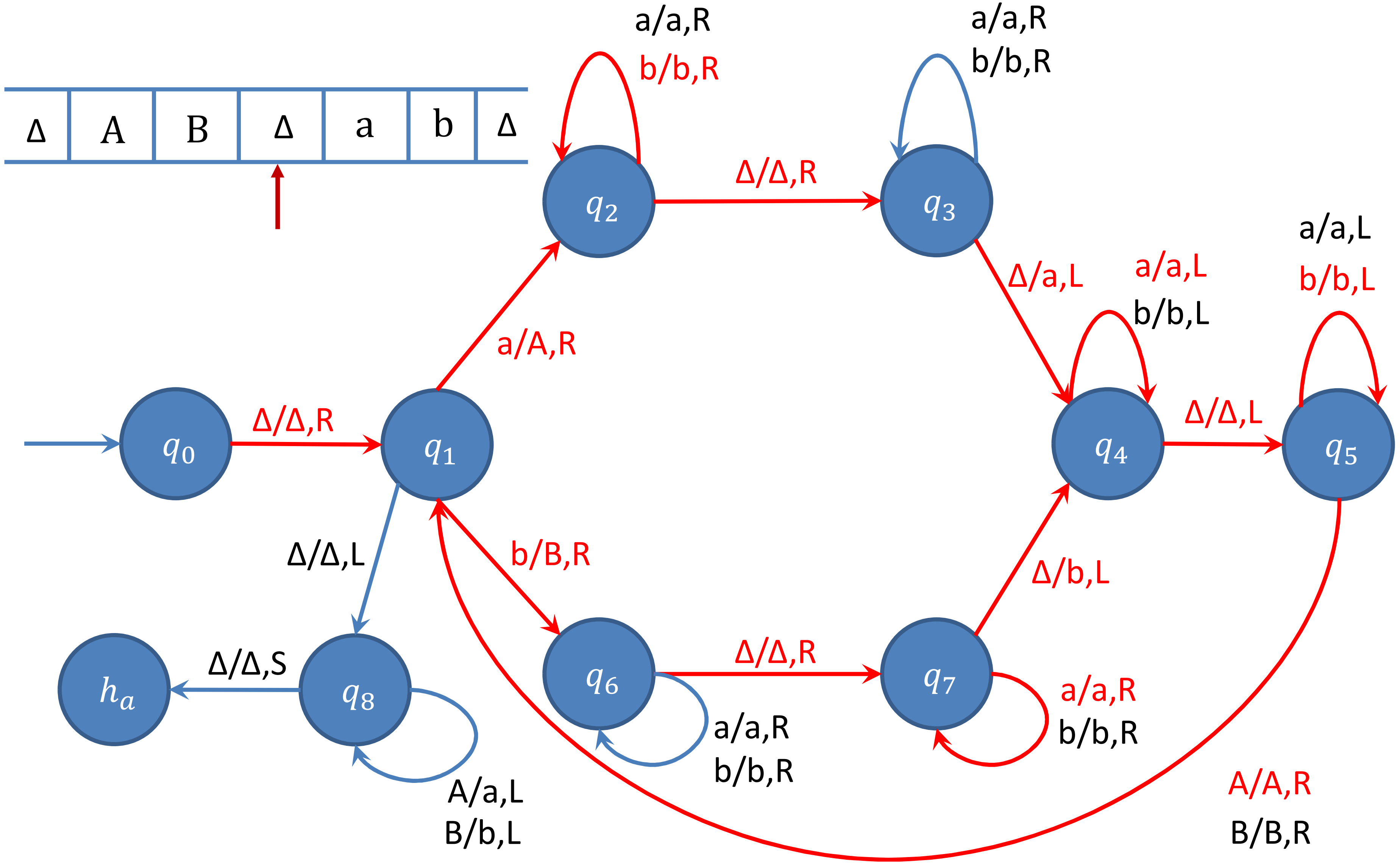
Design a Turing machine to copy a string



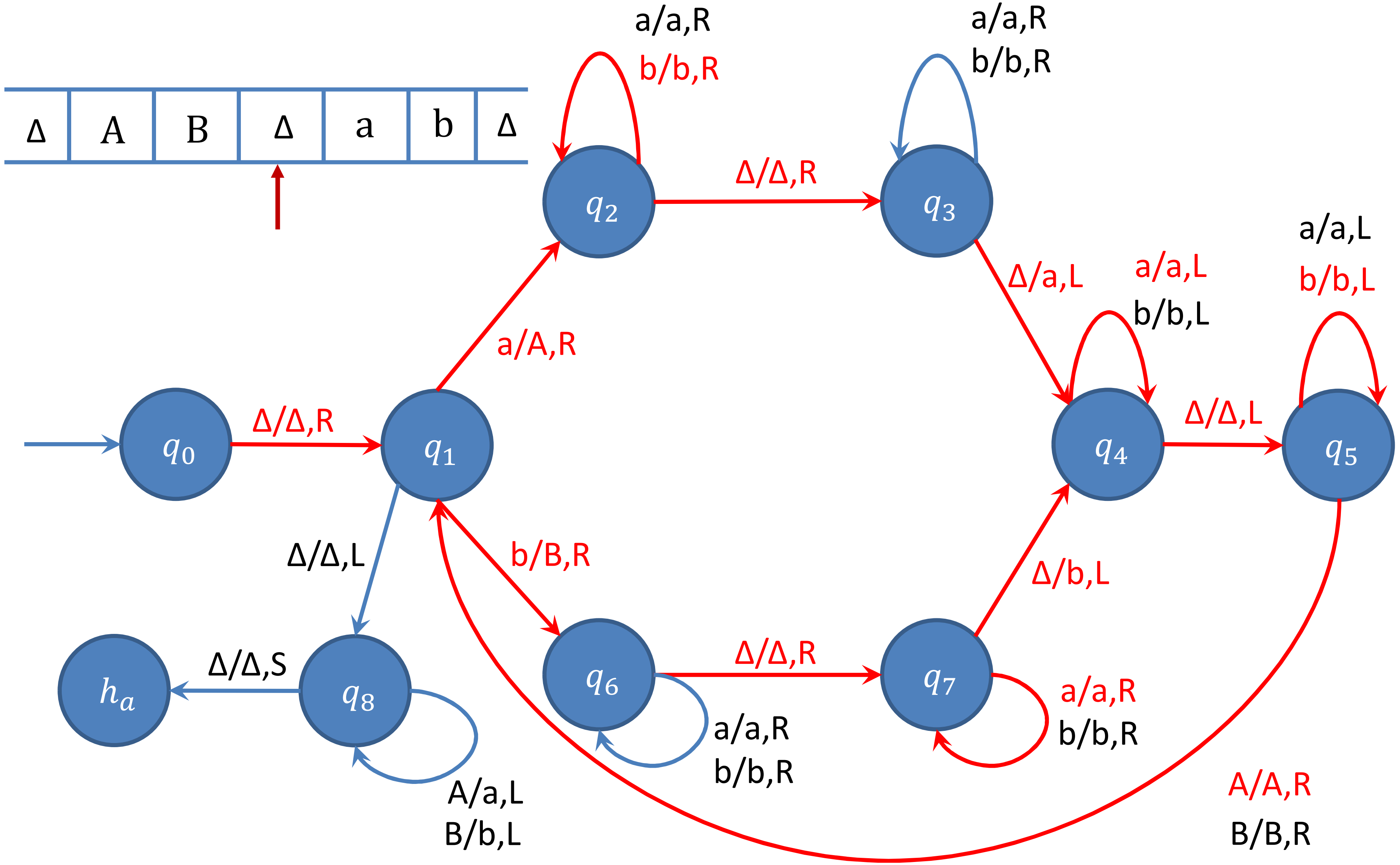
Design a Turing machine to copy a string



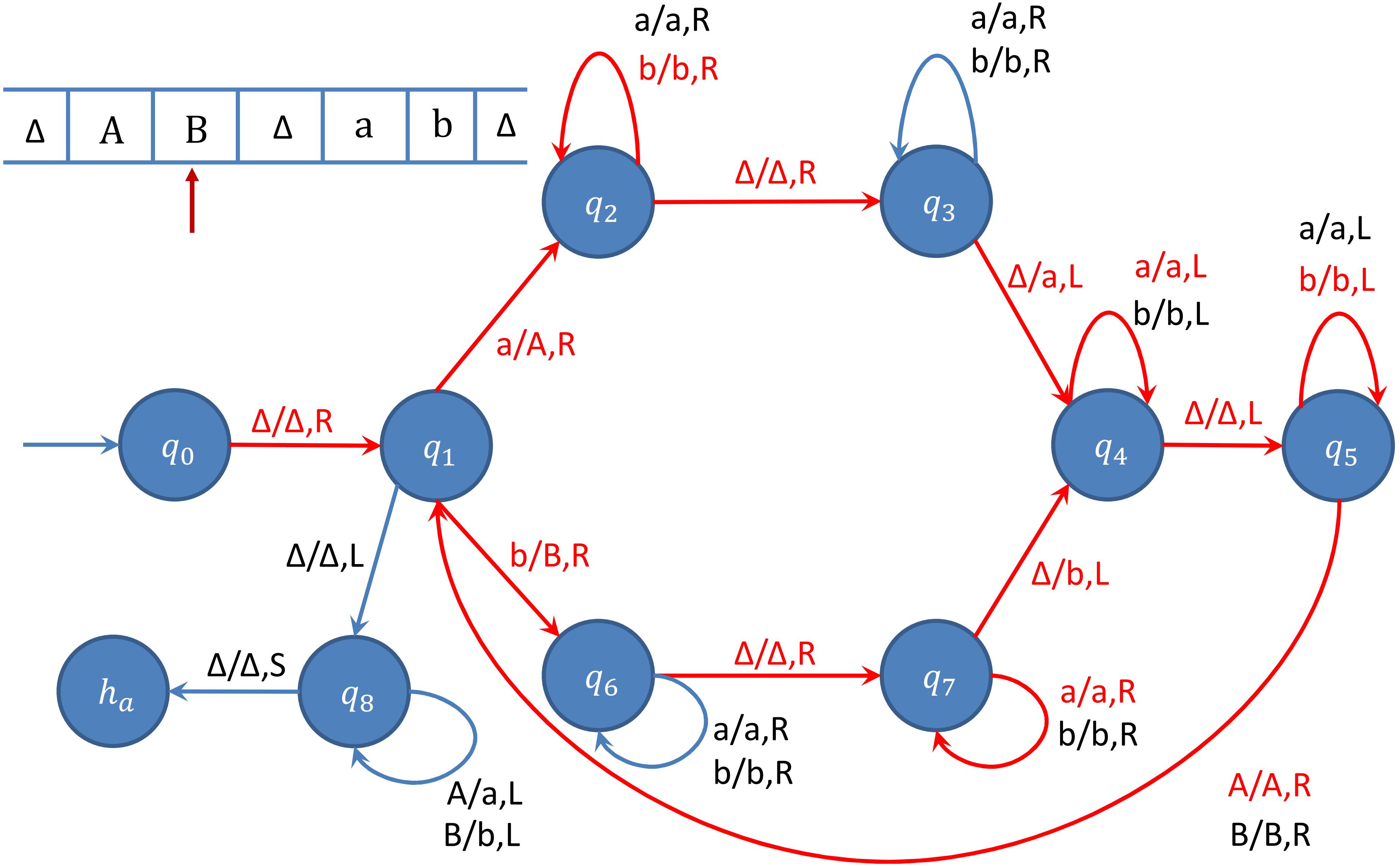
Design a Turing machine to copy a string



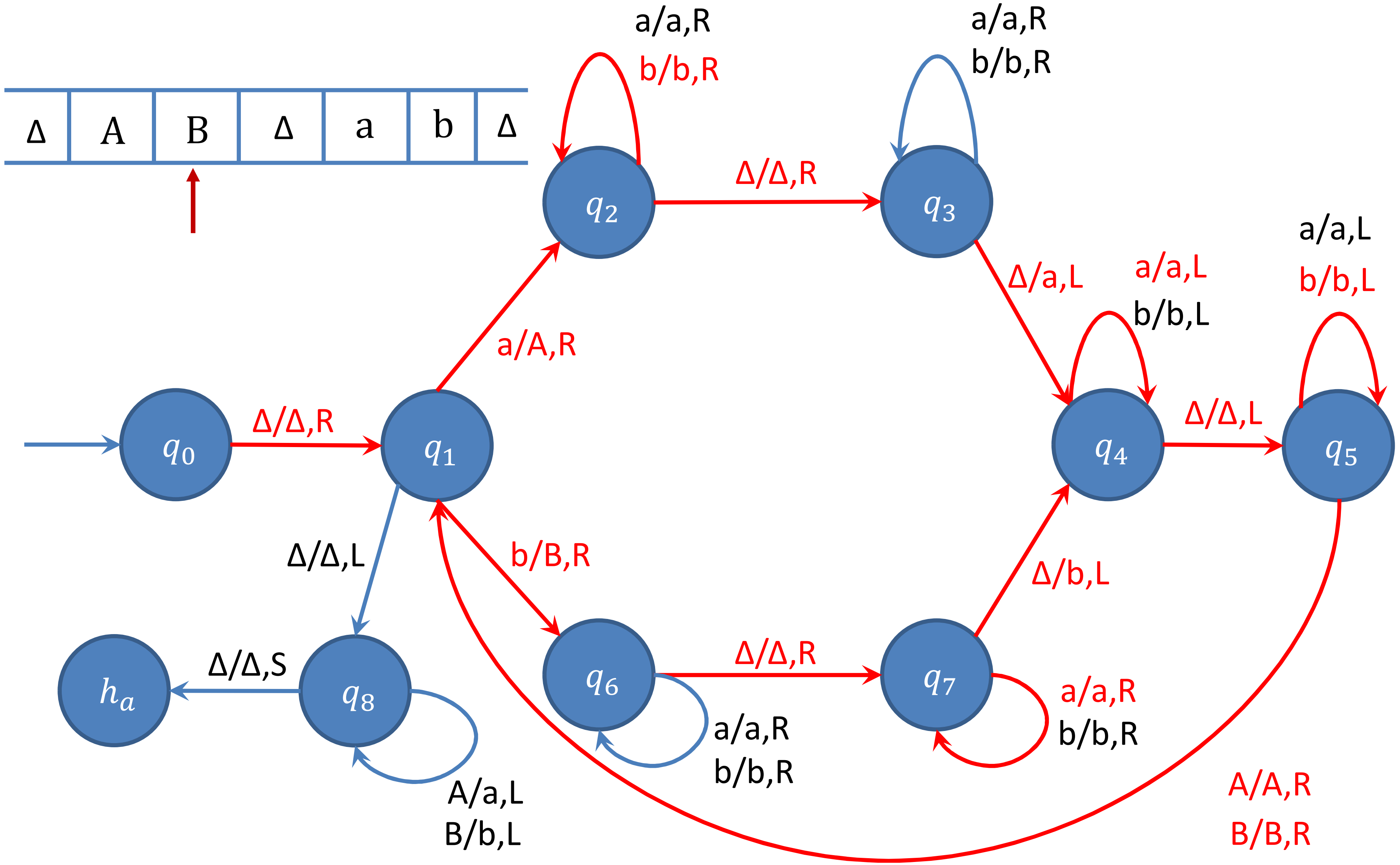
Design a Turing machine to copy a string



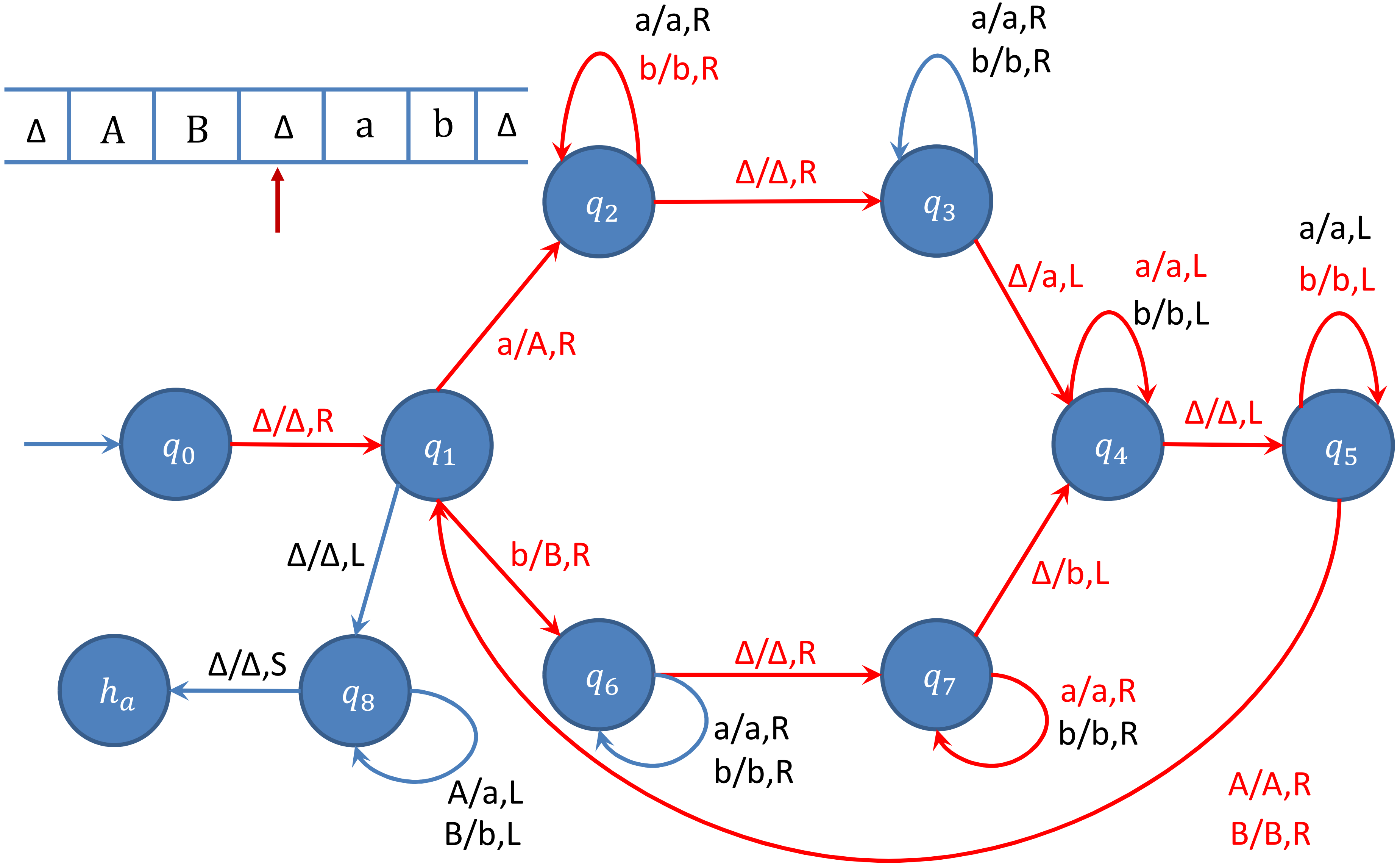
Design a Turing machine to copy a string



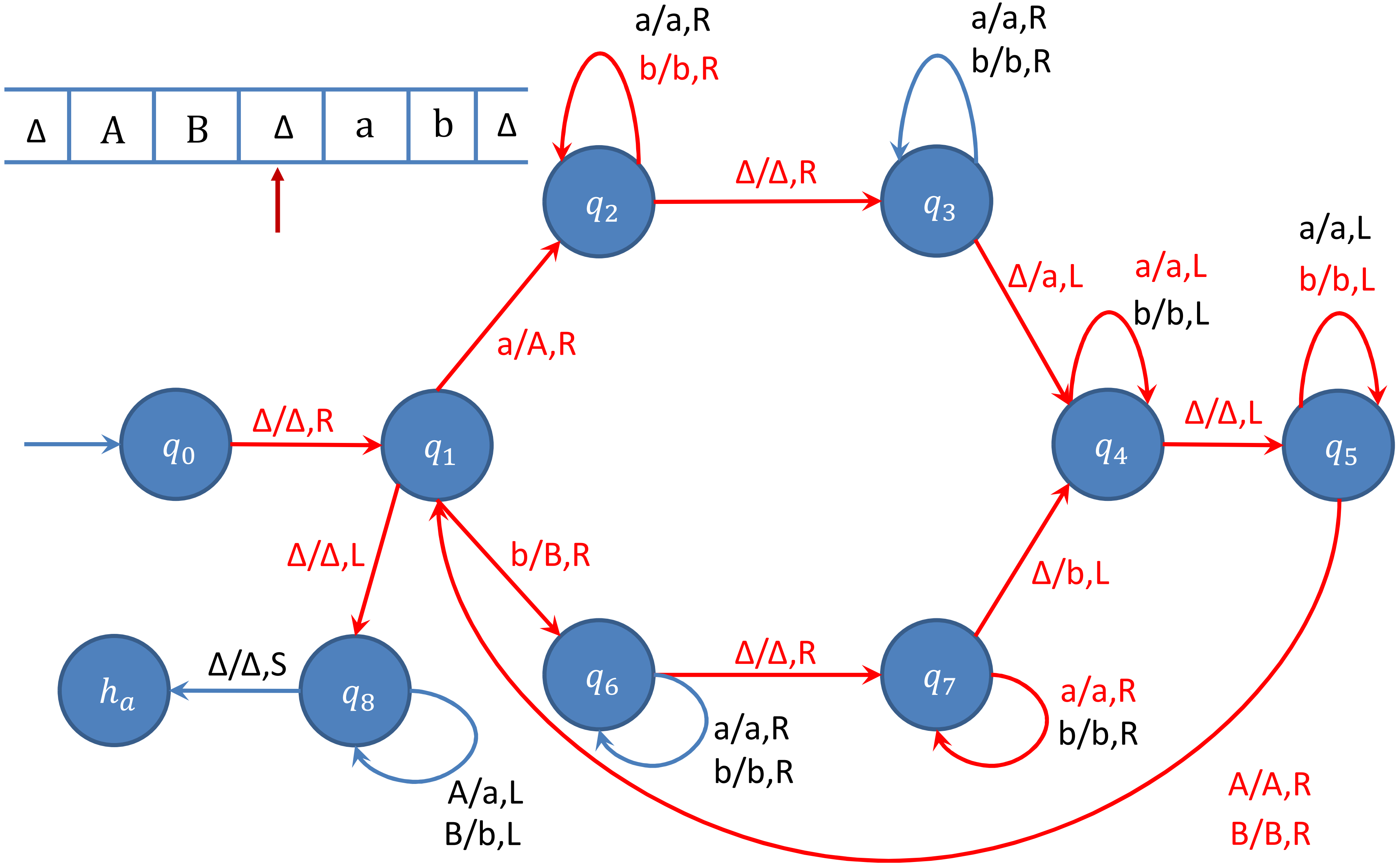
Design a Turing machine to copy a string



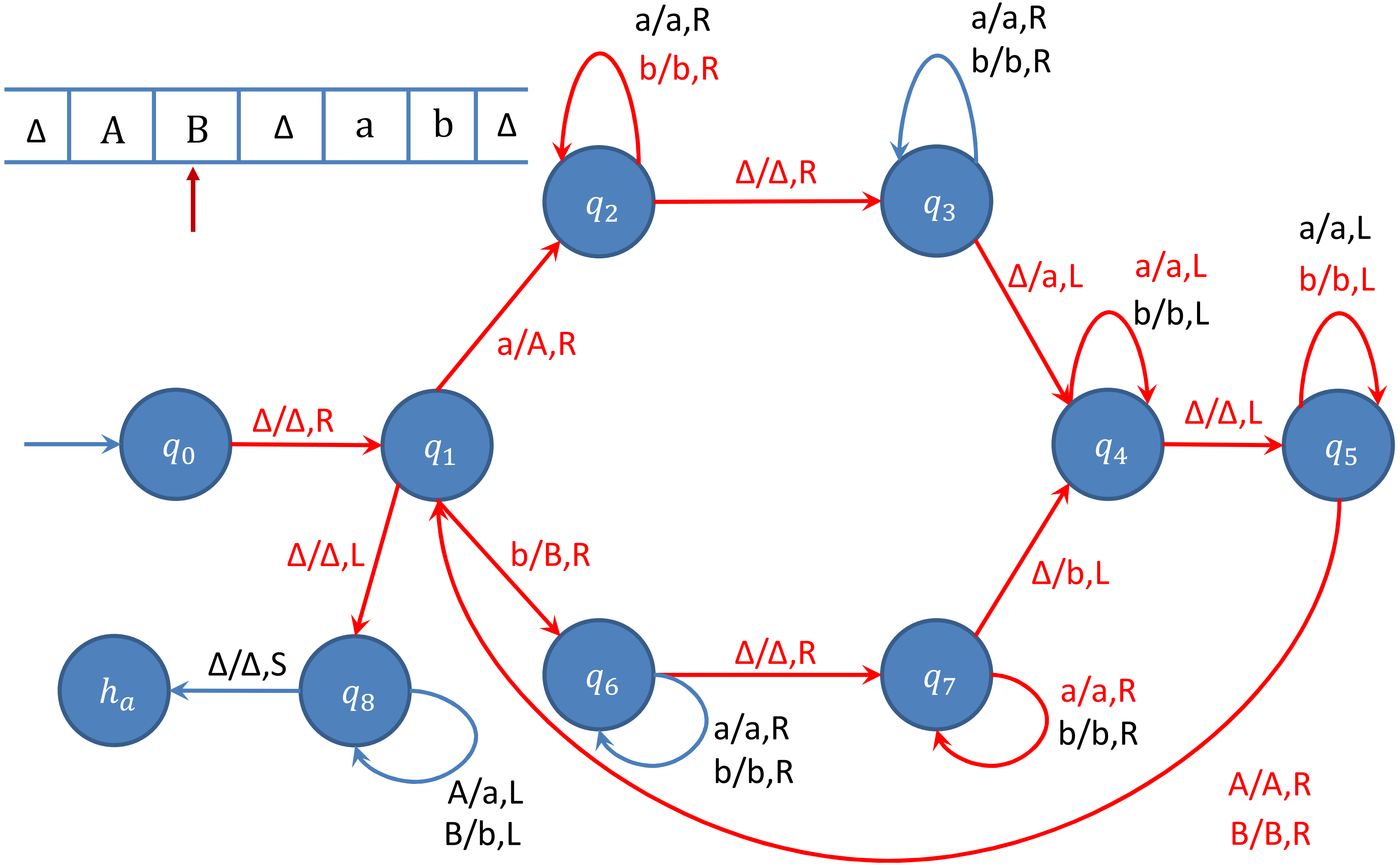
Design a Turing machine to copy a string



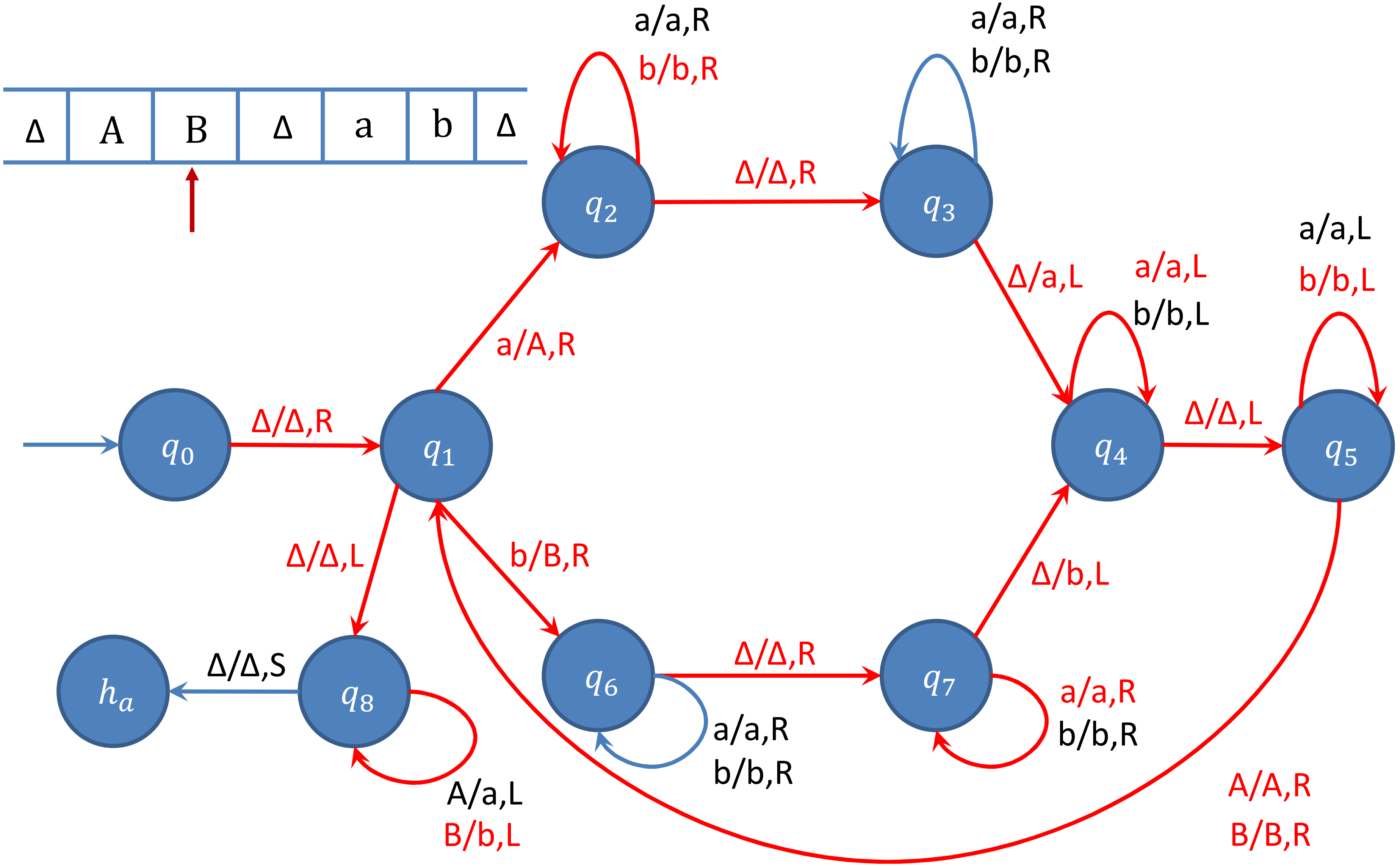
Design a Turing machine to copy a string



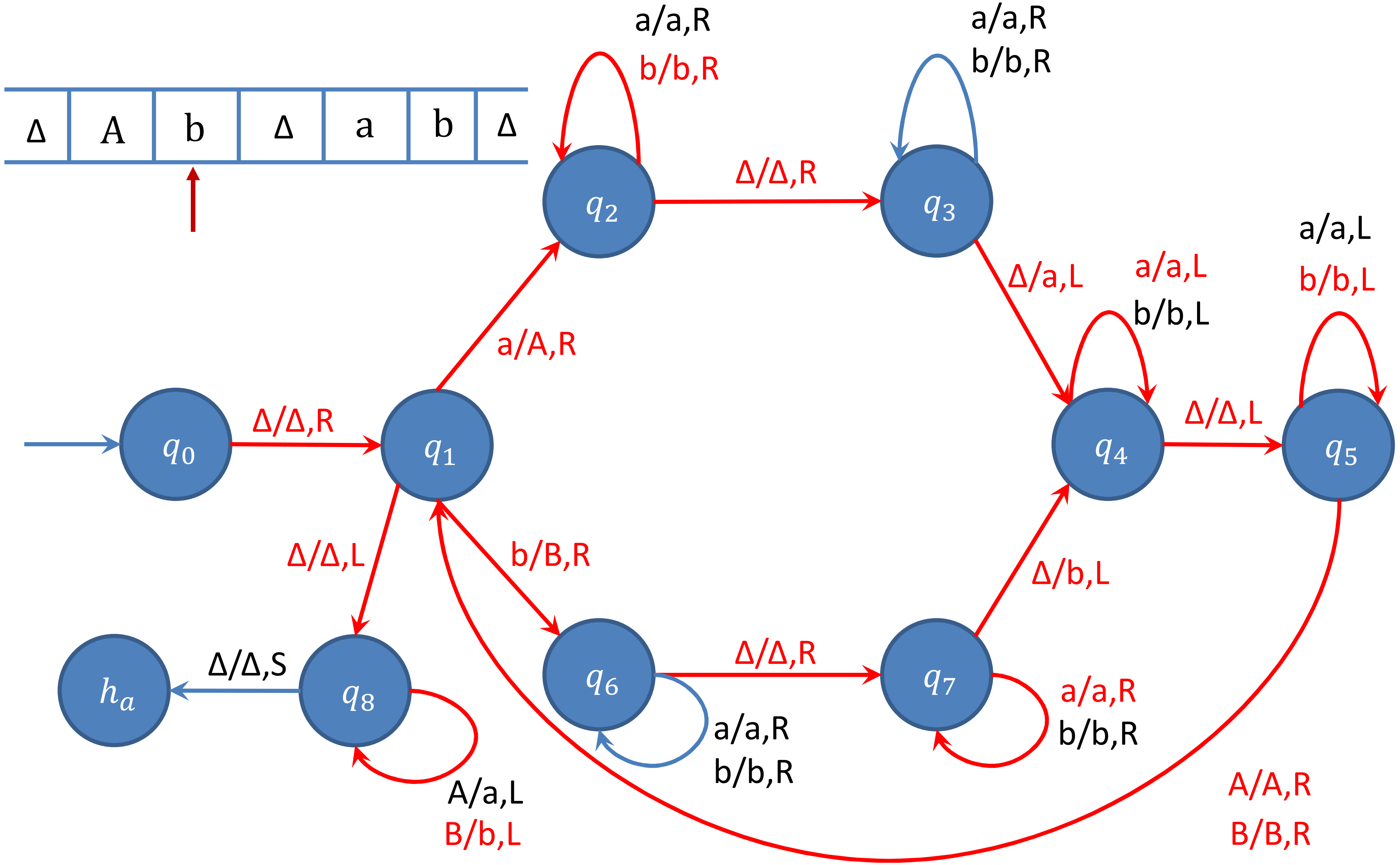
Design a Turing machine to copy a string



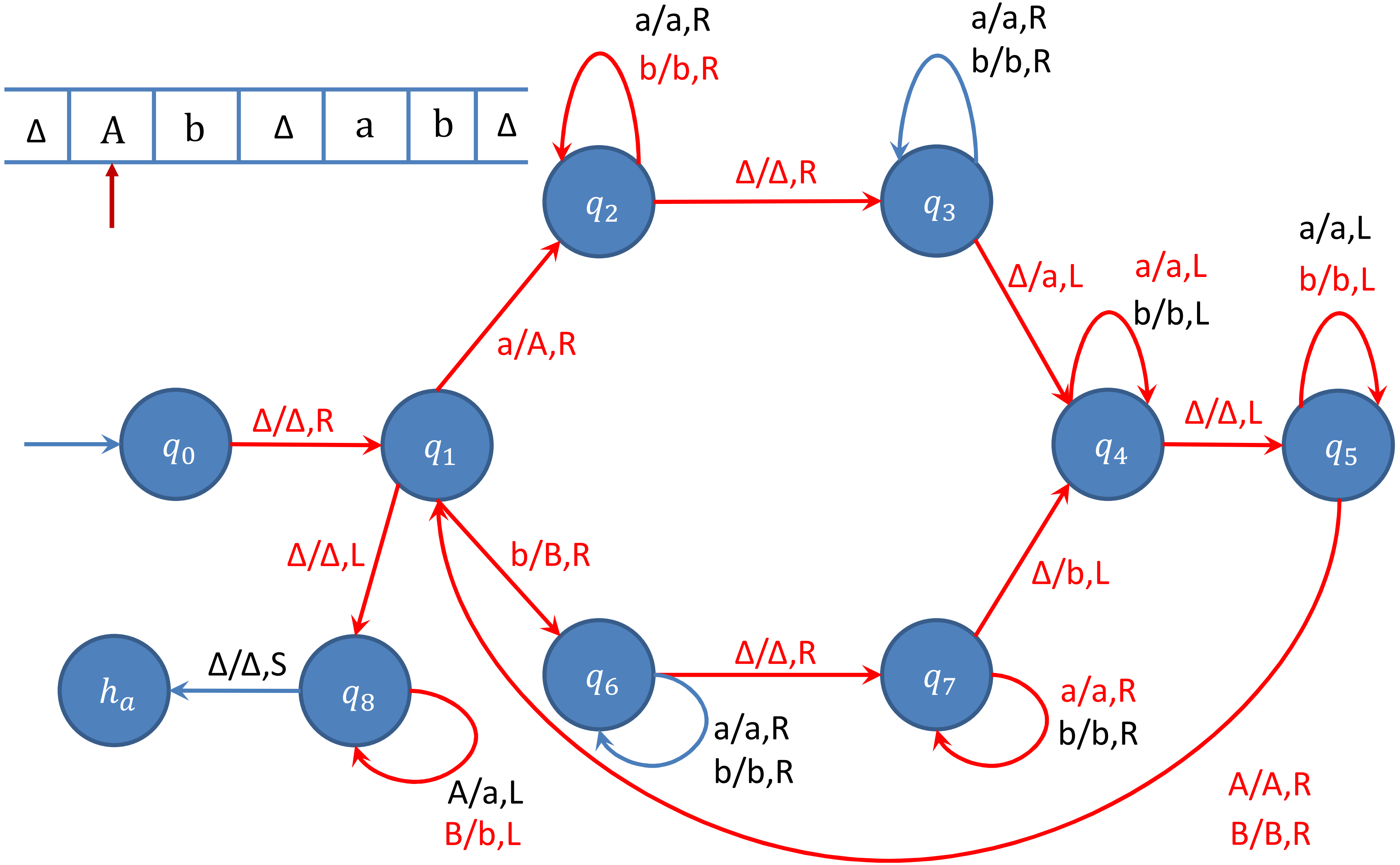
Design a Turing machine to copy a string



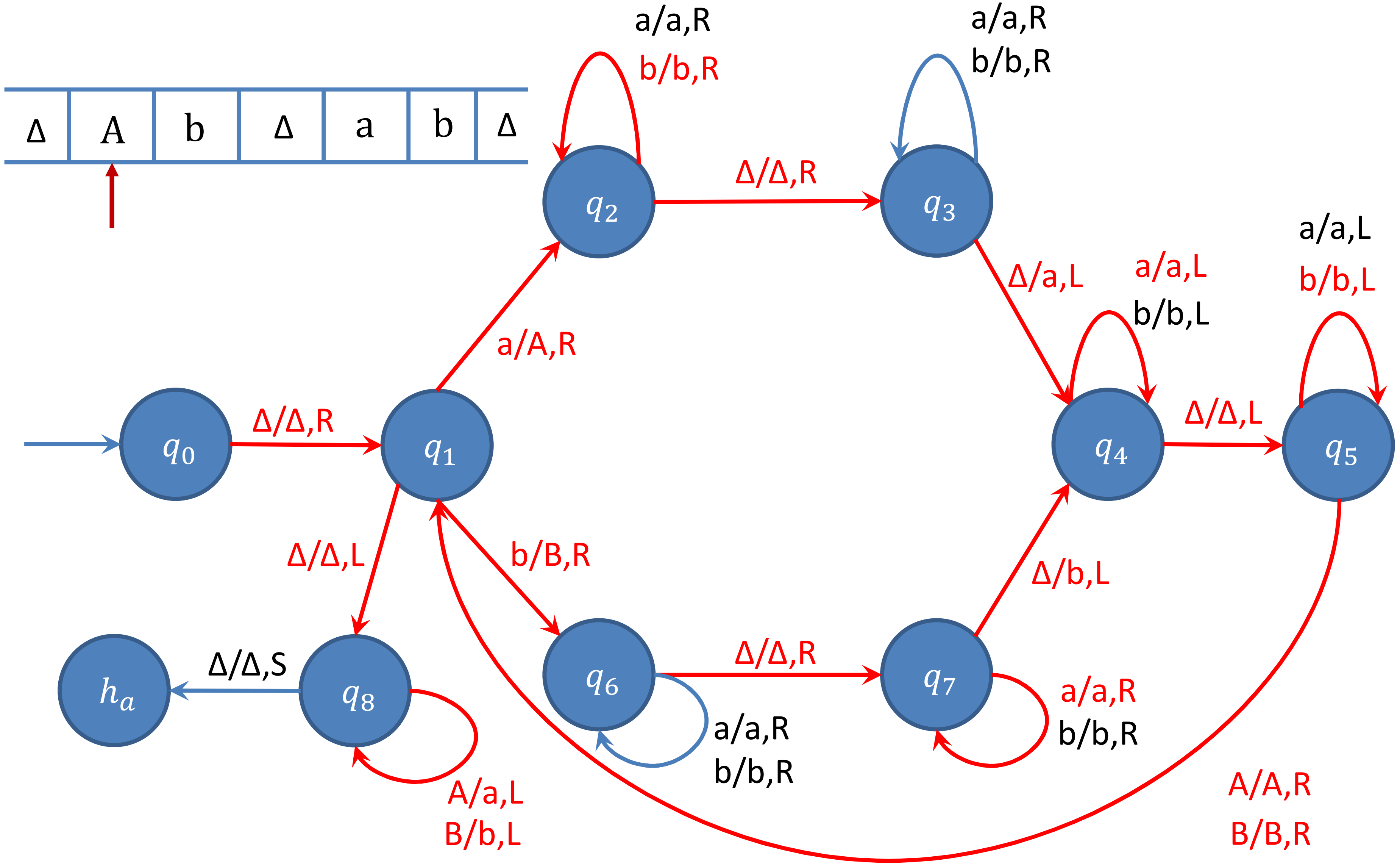
Design a Turing machine to copy a string



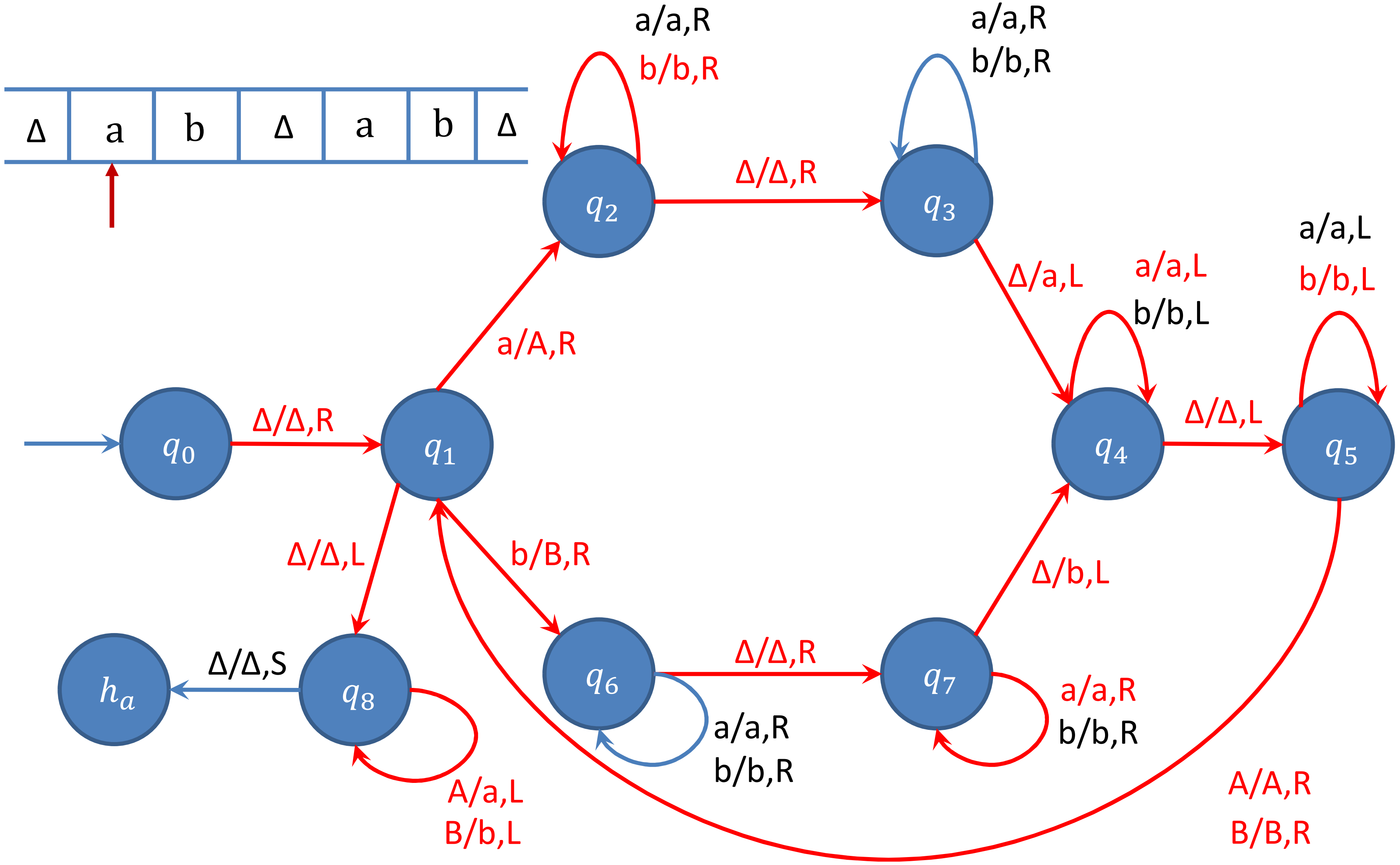
Design a Turing machine to copy a string



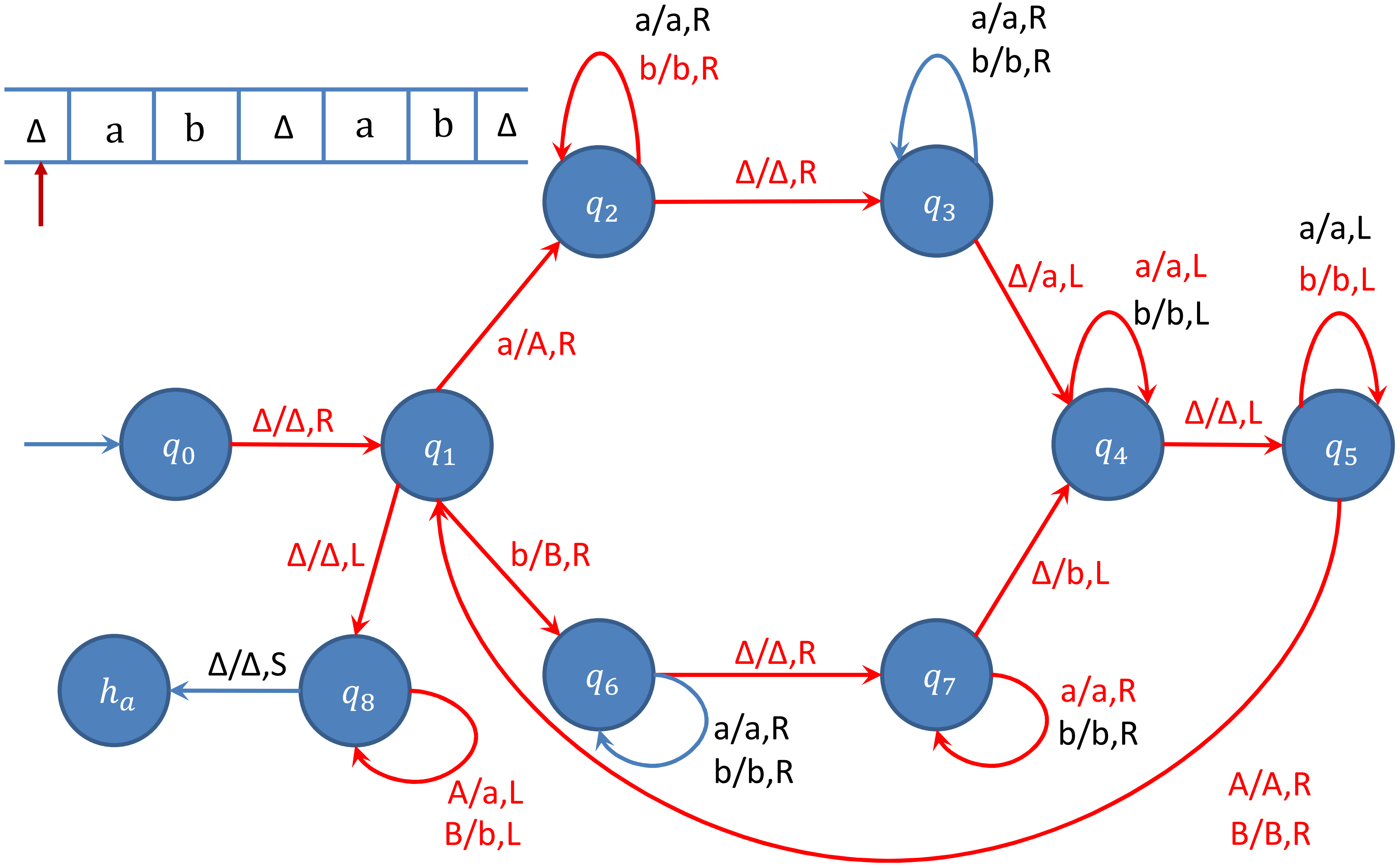
Design a Turing machine to copy a string



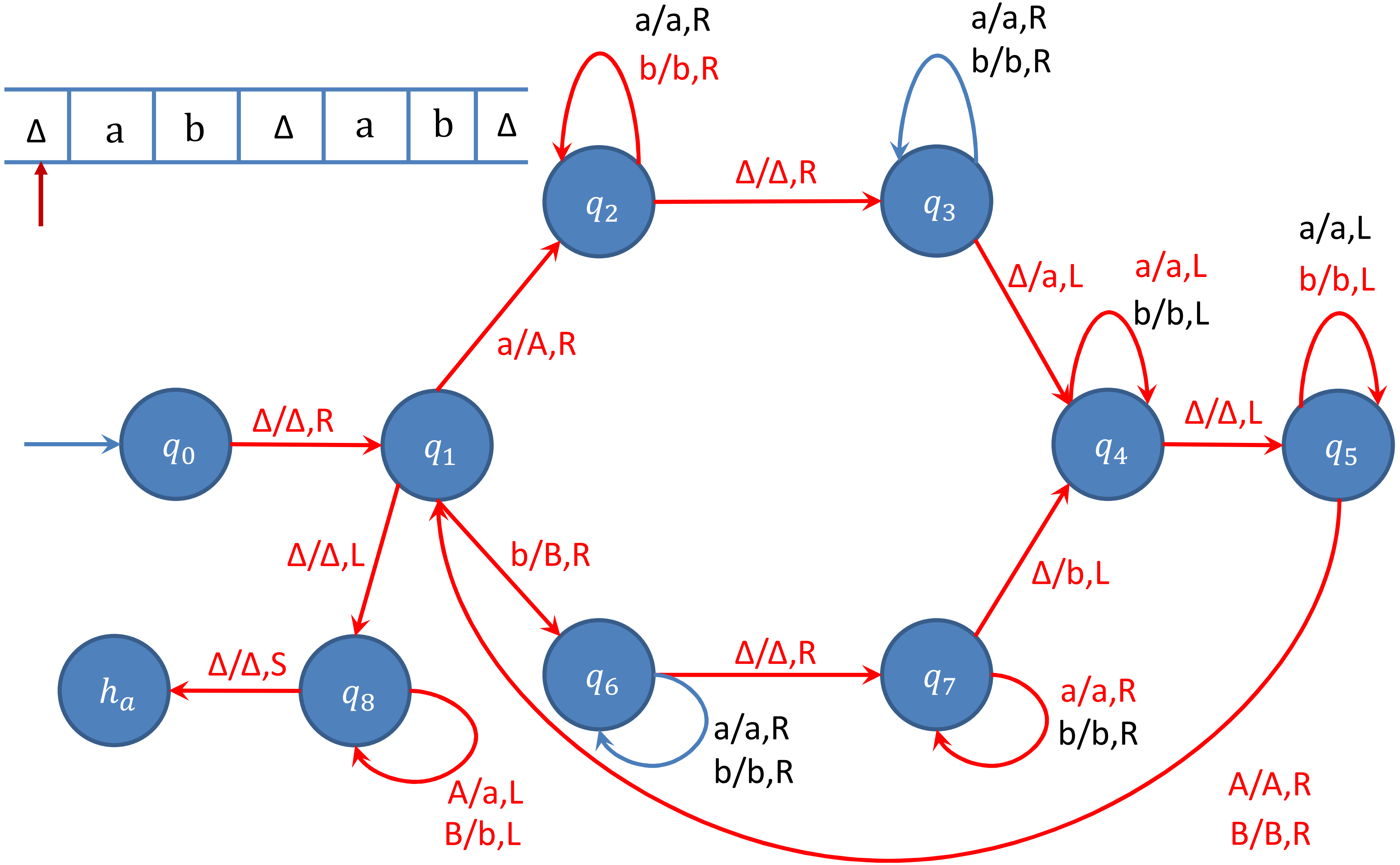
Design a Turing machine to copy a string



Design a Turing machine to copy a string



Design a Turing machine to copy a string



Universal Turing Machine

Universal Turing machine

The language

$$A_{TM} = \{< M, w > \mid M\ is\ a\ Turing\ machine\ \&\ M\ accepts\ w\}$$

is Turing Recognizable

- Question : Given the description of a Turing Machine and some input string, can we decide the machine accepts it?
- Answer: Just Run the TM on the input.
- Possible results:
  1. M accepts w: will halt and accept
  2. M rejects w: will halt and reject
  3. M loops on w: will not halt

Input & Action on Universal Turing machine

Input:

1. M=description of TM
2. w=input string for M

Action:

1. simulate M
2. Behave just like M (May accept, reject or loop)

The UTM is recognizer for

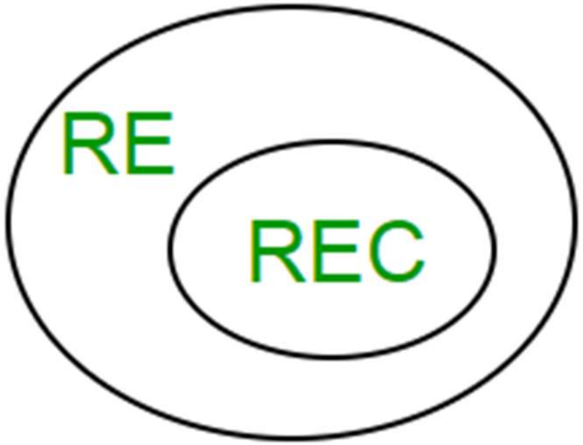
$$A_{TM} = \{< M, w > | M \ is \ a \ Turing \ machine \ and \ M \ accepts \ w\}$$

Recursive and Recursive Enumerable Languages

- **Recursive Enumerable (RE) or Type -0 Language**
- RE languages or type-0 languages are generated by type-0 grammars.
- An RE language can be accepted or recognized by Turing machine which means it will enter into final state for the strings of language and **may or may not enter into rejecting state for the strings which are not part of the language.**
- It means TM can loop forever for the strings which are not a part of the language. RE languages are also called as **Turing recognizable languages**.

Recursive and Recursive Enumerable Languages

- **Recursive Language (REC)**
- A recursive language (subset of RE) can be decided by Turing machine which means it will enter into final state for the strings of language and **rejecting state for the strings which are not part of the language.**
- e.g.; L= {$a^n b^n c^n$|n>=1} is recursive because we can construct a turing machine which will move to final state if the string is of the form $a^n b^n c^n$ else move to non-final state.
- So **the TM will always halt in this case**. REC languages are also called as **Turing decidable languages**.


RE
REC

Closure Properties of Recursive Languages

- **Union**: If L1 and If L2 are two recursive languages, their union L1∪L2 will also be recursive because if TM halts for L1 and halts for L2, it will also halt for L1∪L2.
- **Concatenation:** If L1 and If L2 are two recursive languages, their concatenation L1.L2 will also be recursive
- **Kleene Closure:** If L1is recursive, its kleene closure L1* will also be recursive.
- **Intersection and complement**: If L1 and If L2 are two recursive languages, their intersection L1 ∩ L2 will also be recursive. Complement of recursive language L1 which is ∑*-L1, will also be recursive.

Closure Properties of Recursive Languages

- As opposed to REC languages, RE languages are not closed under complement on which means complement of RE language need not be RE.

Church Turing Thesis

Church Turing Thesis

- What does computable mean?
- Meaning of term Computable was given by:
  1. Alonzo church (Lambda calculus)
  2. Allen Turing (Turing Machine)

- Any *algorithmic procedure* that can be carried out at all, by a human computer or a team of humans or an electronic computer, can be carried out by a TM. This statement usually referred to as Church's thesis, or the Church-Turing thesis.

Church Turing Thesis

- Here is an informal summary of some of the evidence.
  1. Humans normally work with a two-dimensional sheet of paper. A TM tape could be organized so as to simulate two dimensions; one likely consequence would be that the TM would require more moves to do what a human could do in one.
  2. Various enhancements of the TM model have been suggested to make the operation more like that of a human computer, or more convenient & efficient. The multitape TM is an example.
  3. Other models includes abstract machines with two stacks or with a queue, multihead TM, Non-erasing TM, Always writing TM, Multidimensional TM .
  4. Since the introduction of the Turing machine, no one has suggested any type of computation that ought to be included in the category of "*algorithmic procedure*" and cannot be implemented on a TM.