

**TOC** **EXPERIMENTS**

**COURSE CODE: CSA13**

**COURSE NAME: THEORY OF COMPUTATION**

**C PROGRAMMING:**

1.Write a C program to simulate a Deterministic Finite Automata (DFA) for the given language representing strings that start with a and end with a.

**PROGRAM:**

#include <stdio.h>

#include <string.h>

int main () {

char String [100];

printf ("Enter a string: ");

scanf ("%s", String);

if (String [0] =='a'&&String [strlen (String)-1] =='a')

{

int i;

for (i=0; i<strlen (String); i++) {

if (String[i]=='0'||String[i]=='1')

{

printf ("Invalid! \n");

return 0;

}

}

printf ("string is accepted\n");

} else {

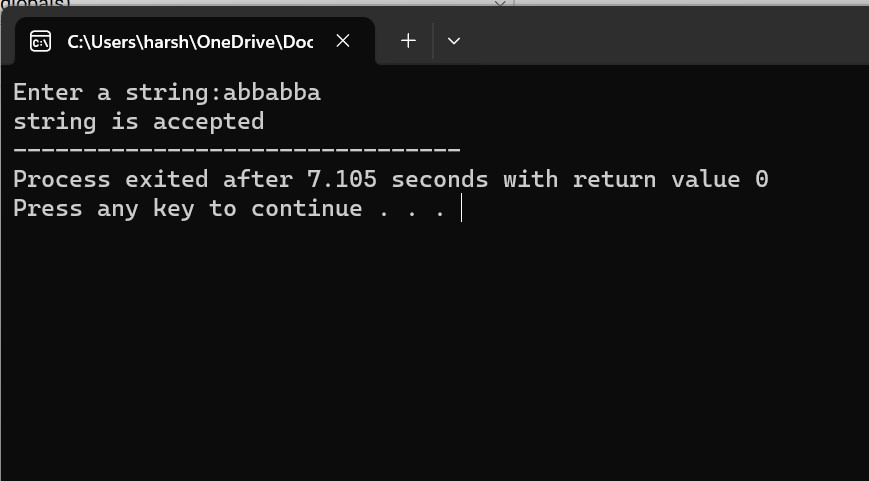
printf ("Not Accepted\n");

}

return 0;

}

**OUTPUT:**



2. Write a C program to simulate a Deterministic Finite Automata (DFA) for the given language representing strings that start with 0 and end with 1.

**PROGRAM**:

#include <stdio.h>

#include <string.h>

int main () {

char String [100];

printf ("Enter a string: ");

scanf ("%s", String);

if (String [0] =='0'&&String [strlen (String)-1] =='1')

{

int i;

for (i==; i<strlen (String); i++) {

if (String[i]<'0'||String[i]>'1')

{

printf ("Invalid! \n");

return 0;

}

}

printf ("Valid! The string starts with '0' and ends with '1'.\n");

} else {

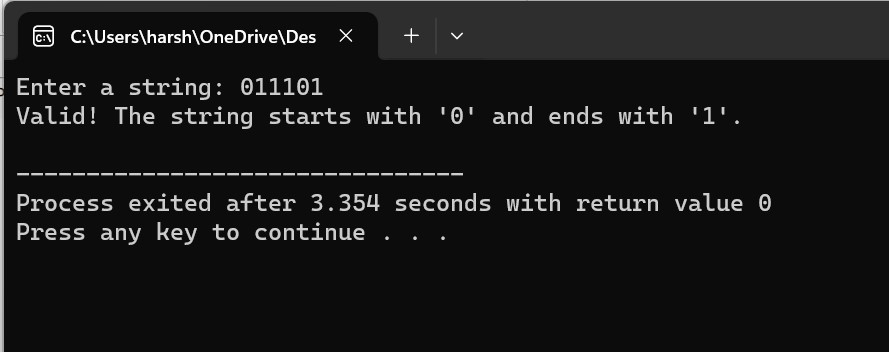
printf ("Invalid! The string does not start with '0' and end with '1'.\n");

}

return 0;

}

**OUTPUT:**



3. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG) S → 0A1 A → 0A | 1A | ε

**PROGRAM:**

#include <stdio.h>

#include <string.h>

int main () {

char String [100];

printf ("Enter a string: ");

scanf ("%s", String);

if (String [0] =='0'&&String [strlen (String)-1] =='1')

{

int i;

for (i==; i<strlen (String); i++) {

if (String[i]<'0'||String[i]>'1')

{

printf ("Invalid! \n");

return 0;

}

}

printf ("Valid! The string starts with '0' and ends with '1'.\n");

} else {

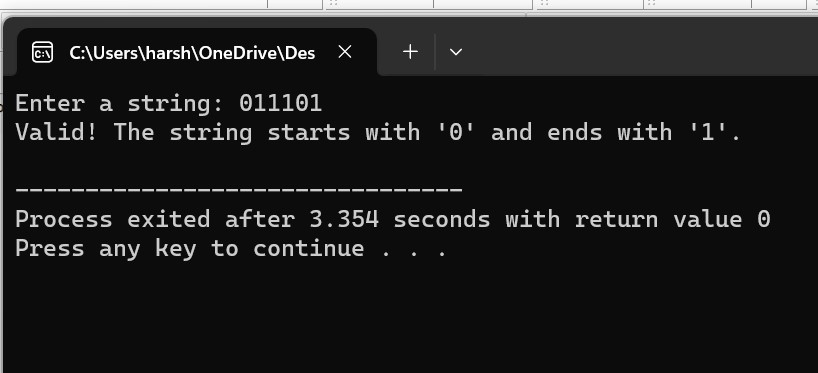
printf ("Invalid! The string does not start with '0' and end with '1'.\n");

}

return 0;

}

**OUTPUT:**



4. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG) S → 0S0 | 1S1 | 0 | 1 | ε.

**PROGRAM:**

#include <stdio.h>

#include <string.h>

int main () {

char String [100];

printf ("Enter a string: ");

scanf ("%s", String);

if (String [0] =='0'&&String [strlen (String)-1] =='1' | | String [0] =='1'&&String [strlen (String)-1] =='1'))

{

int i;

for (i==; i<strlen (String); i++) {

if (String[i]<'0'||String[i]>'1')

{

printf ("Invalid! \n");

return 0;

}

}

printf ("Valid! The string starts with '0' and ends with '0' or starts with '1' and ends with '1'.\n");

} else {

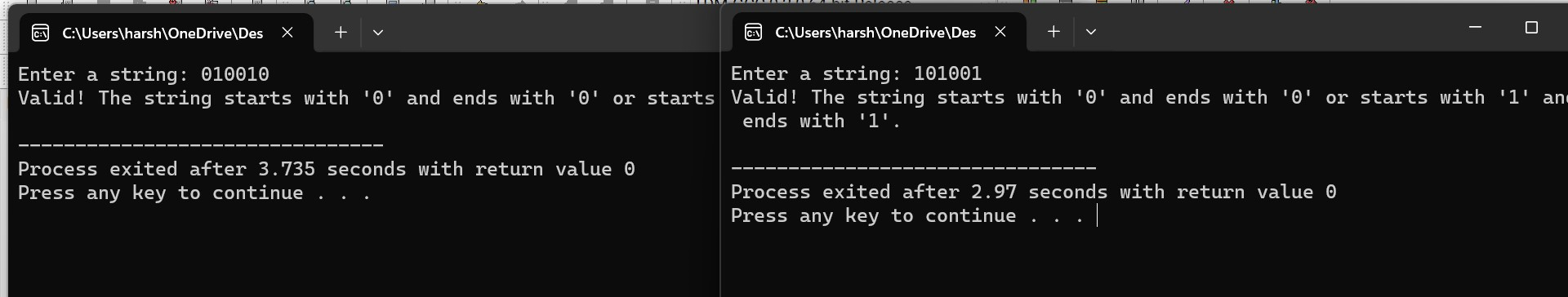
printf ("Invalid! The string starts with '0' and ends with '0' or starts with '1' and ends with '1'.\n");

}

return 0;

}

**OUTPUT:**



5. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG) S → 0S0 | A

A → 1A | ε

**PROGRAM:**

#include <stdio.h>

#include <string.h>

int main () {

char String [100];

printf ("Enter a string: ");

scanf ("%s", String);

if (String [0] =='0'&&String [strlen (String)-1] =='1' | | String [0] =='1'&&String [strlen (String)-1] =='1'))

{

int i;

for (i==; i<strlen (String); i++) {

if (String[i]<'0'||String[i]>'1')

{

printf ("Invalid! \n");

return 0;

}

}

printf ("Valid! The string starts with '0' and ends with '0' or starts with '1' and ends with '1'.\n");

} else {

printf ("Invalid! The string starts with '0' and ends with '0' or starts with '1' and ends with '1'.\n");

}

return 0;

}

**OUTPUT:**



6. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG) S → 0S1 | ε

**PROGRAM:**

#include <stdio.h>

#include <string.h>

int main () {

char String [100];

printf ("Enter a string: ");

scanf ("%s", String);

if (String [0] =='0'&&String [strlen (String)-1] =='1')

{

int i;

for (i==; i<strlen (String); i++) {

if (String[i]<'0'||String[i]>'1')

{

printf ("Invalid! \n");

return 0;

}

}

printf ("Valid! The string starts with '0' and ends with '1'.\n");

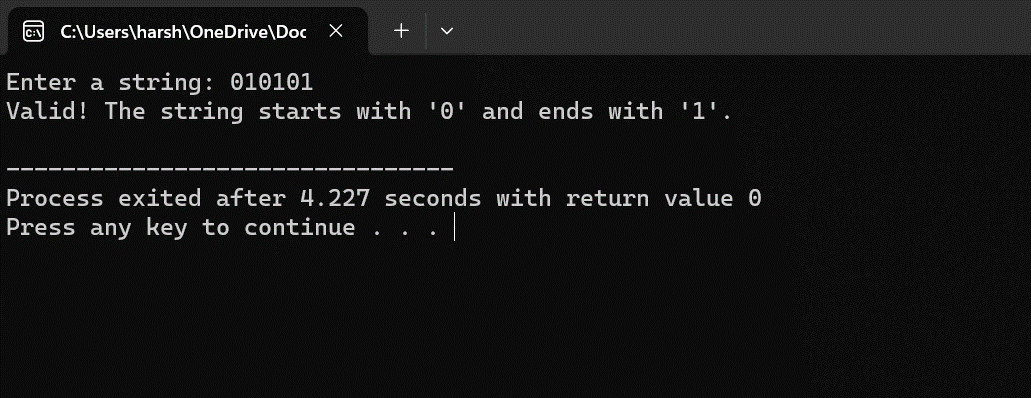
} else {

printf ("Invalid! The string does not start with '0' and end with '1'.\n");

}

return 0;

**OUTPUT:**



7. Write a C program to check whether a given string belongs to the language defined by a Context Free Grammar (CFG) S → A101A, A → 0A | 1A | ε

**PROGRAM:**#include <stdio.h>

#include <string.h>

int main () {

char String [100];

printf ("Enter a string: ");

scanf ("%s", String);

if (String [0] ==’0’ || ‘1’ &&String [strlen (String)-1] ==’0’ || ‘1’)

{

int i;

for (i==; i<strlen (String); i++) {

if (String[i]<'0'||String[i]>'1')

{

printf ("Invalid! \n");

return 0;

}

}

printf ("Valid! The string starts with '0' or '1' and ends with '0' or '1'.\n");

} else {

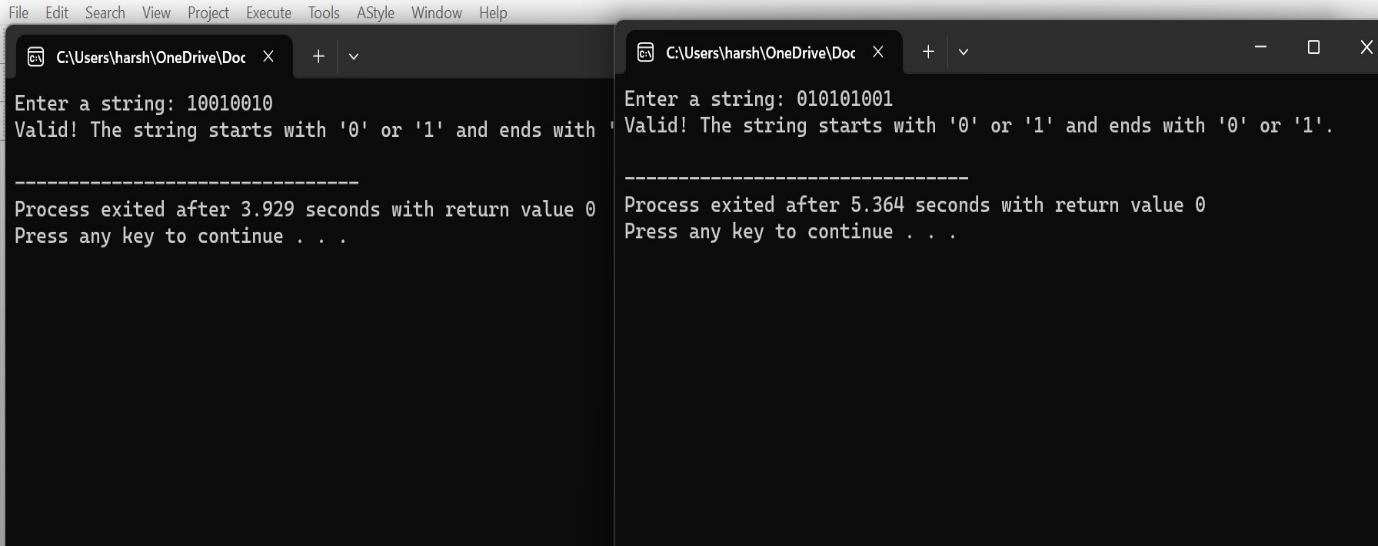
printf ("Invalid! The string does not start with '0' or '1' and end with '0' or '1'.\n");

}

return 0;

}

**OUTPUT:**



8. Write a C program to simulate a Non-Deterministic Finite Automata (NFA) for the given language representing strings that start with b and end with a.

**PROGRAM:**#include <stdio.h>

#include <string.h>

int main () {

char String [100];

printf ("Enter a string: ");

scanf ("%s", String);

if (String [0] =='a'&&String [strlen (String)-1] =='a')

{

int i;

for (i=0; i<strlen (String); i++) {

if (String[i]=='0'||String[i]=='1')

{

printf ("Invalid! \n");

return 0;

}

}

printf ("Accepted\n");

} else {

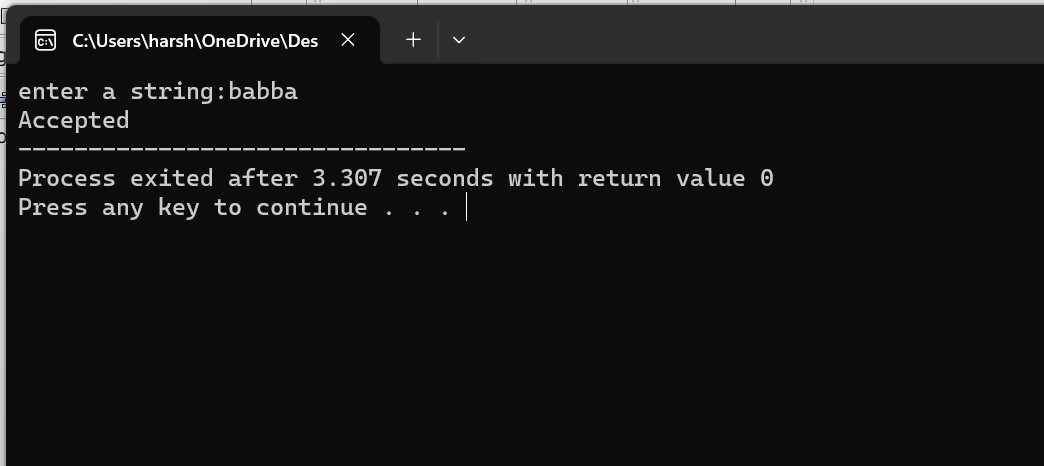
printf ("Not Accepted\n");

}

return 0;

}

**OUTPUT:**



9. Write a C program to simulate a Non-Deterministic Finite Automata (NFA) for the given language representing strings that start with o and end with 1.

**PROGRAM:**

#include <stdio.h>

#include <string.h>

int main () {

char String [100];

printf ("Enter a string: ");

scanf ("%s", String);

if (String [0] =='0'&&String [strlen (String)-1] =='1')

{

int i;

for (i==; i<strlen (String); i++) {

if (String[i]<'0'||String[i]>'1')

{

printf ("Invalid! \n");

return 0;

}

}

printf ("Valid! The string starts with '0' and ends with '1'.\n");

} else {

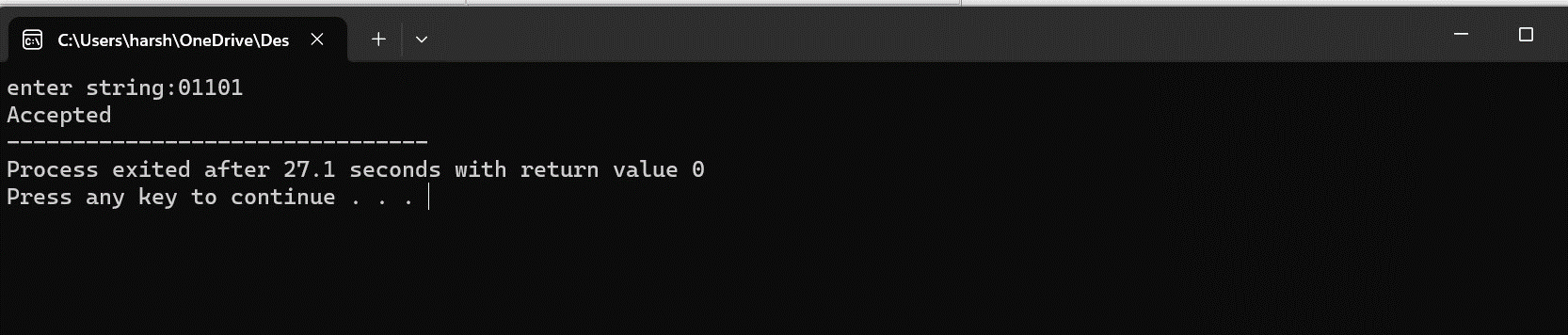
printf ("Invalid! The string does not start with '0' and end with '1'.\n");

}

return 0;

}

**OUTPUT:**



10. Write a C program to find ε -closure for all the states in a Non-Deterministic Finite Automata (NFA) with ε -moves.

**PROGRAM:**

#include <stdio.h>

int main () {

int n;

int m;

printf ("Enter the number of states: ");

scanf ("%d", &n);

printf ("Enter the number of transitions: ");

scanf ("%d", &m);

int transitions [3][3];

for (int i = 0; i < m; i++) {

printf ("Enter transition %d (fromState inputSymbol toState): ", i + 1);

scanf ("%d", &transitions[i][0]);

char inputSymbol [2];

scanf ("%1s", inputSymbol);

scanf ("%d", &transitions[i][2]);

if (inputSymbol [0] == 'e') {

transitions[i][1] = 'e';

} else {

transitions[i][1] = inputSymbol [0];

}

}

for (int i = 0; i < n; i++) {

printf("e-closure(%d): {%d ", i, i);

for (int j = 0; j < m; j++) {

if (transitions[j][0] == i && transitions[j][1] == 'e') {

printf ("%d ", transitions[j][2]);

}

}

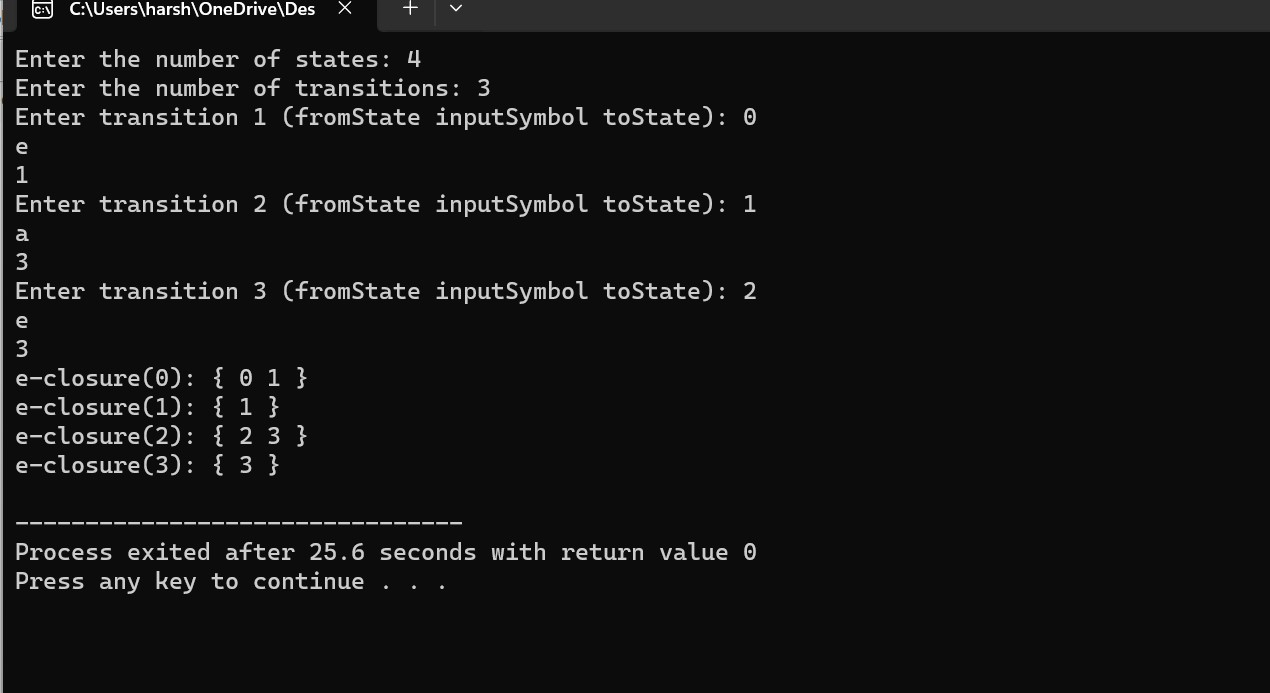
printf (“} \n”);

}

return 0;

}

**OUTPUT:**



11. Write a C program to find ε -closure for all the states in a Non-Deterministic Finite Automata (NFA) with ε -moves.

**PROGRAM:**

#include <stdio.h>

int main () {

int n;

int m;

printf ("Enter the number of states: ");

scanf ("%d", &n);

printf ("Enter the number of transitions: ");

scanf ("%d", &m);

int transitions [3][3];

for (int i = 0; i < m; i++) {

printf ("Enter transition %d (fromState inputSymbol toState): ", i + 1);

scanf ("%d", &transitions[i][0]);

char inputSymbol [2];

scanf ("%1s", inputSymbol);

scanf ("%d", &transitions[i][2]);

if (inputSymbol [0] == 'e') {

transitions[i][1] = 'e';

} else {

transitions[i][1] = inputSymbol [0];

}

}

for (int i = 0; i < n; i++) {

printf("e-closure(%d): {%d ", i, i);

for (int j = 0; j < m; j++) {

if (transitions[j][0] == i && transitions[j][1] == 'e') {

printf ("%d ", transitions[j][2]);

}

}

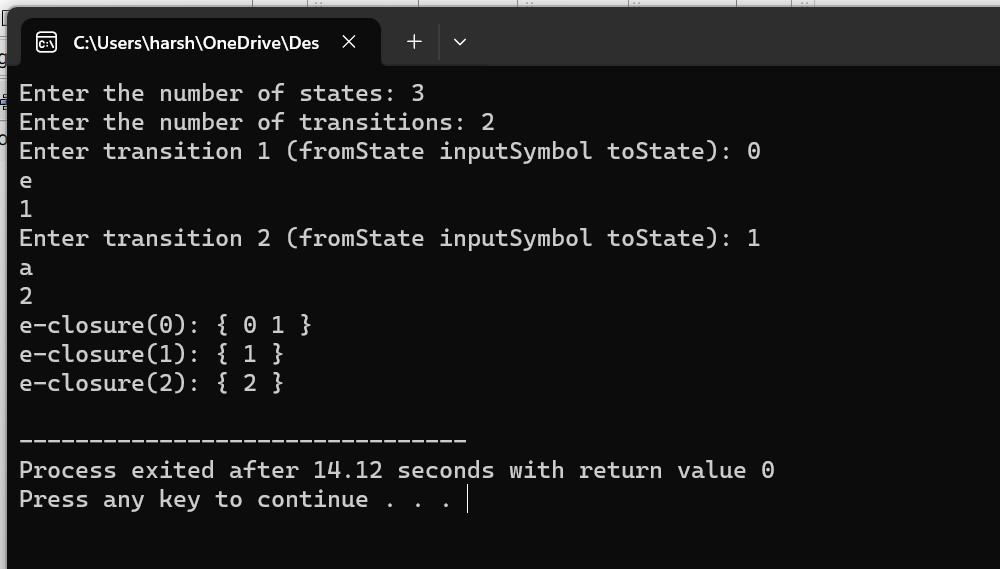
printf (“} \n”);

}

return 0;

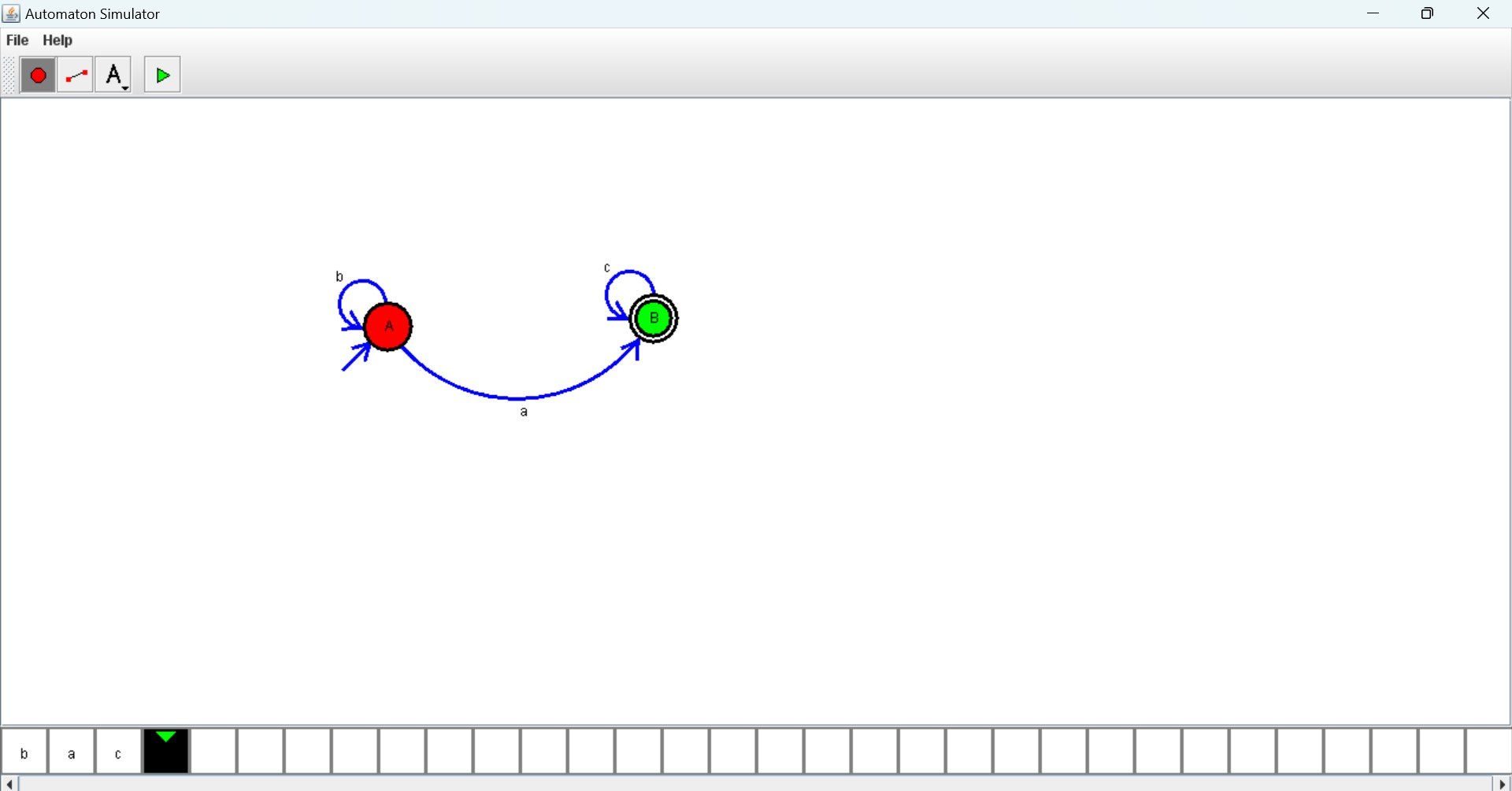
}

**OUTPUT:**

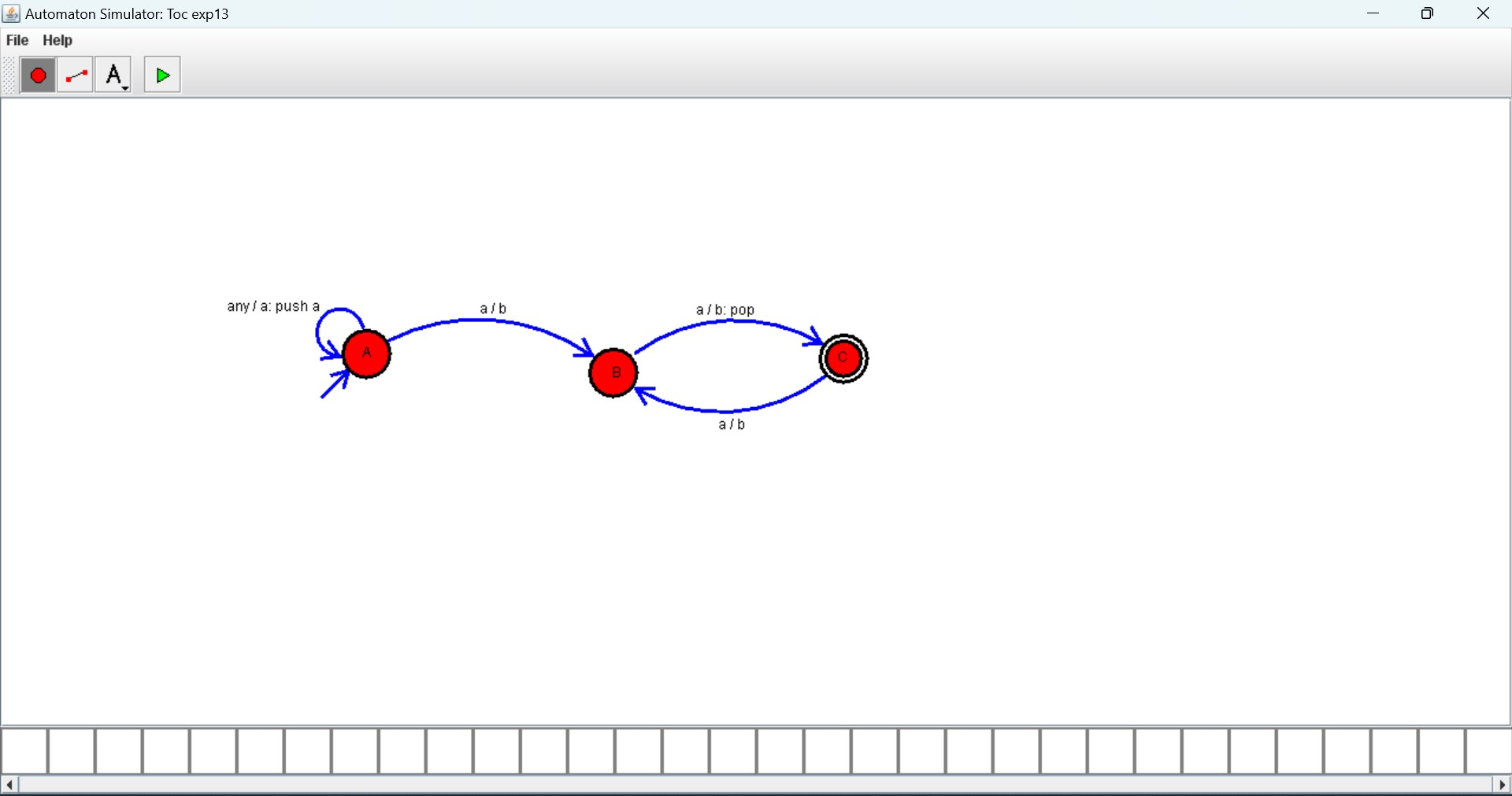


**AUTO SIM:**

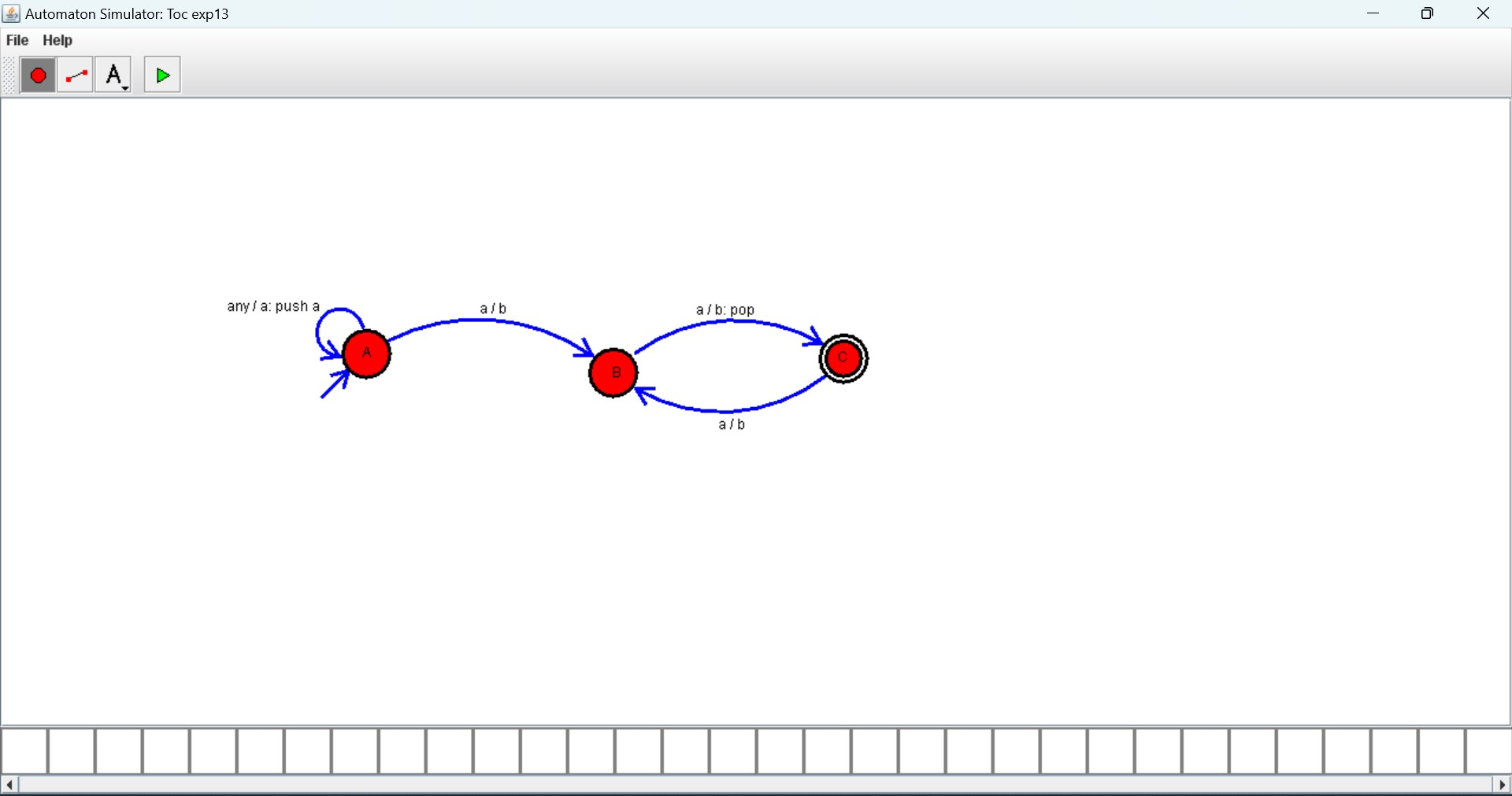
12. Design DFA using simulator to accept the input string “a”, “a” and” bac”



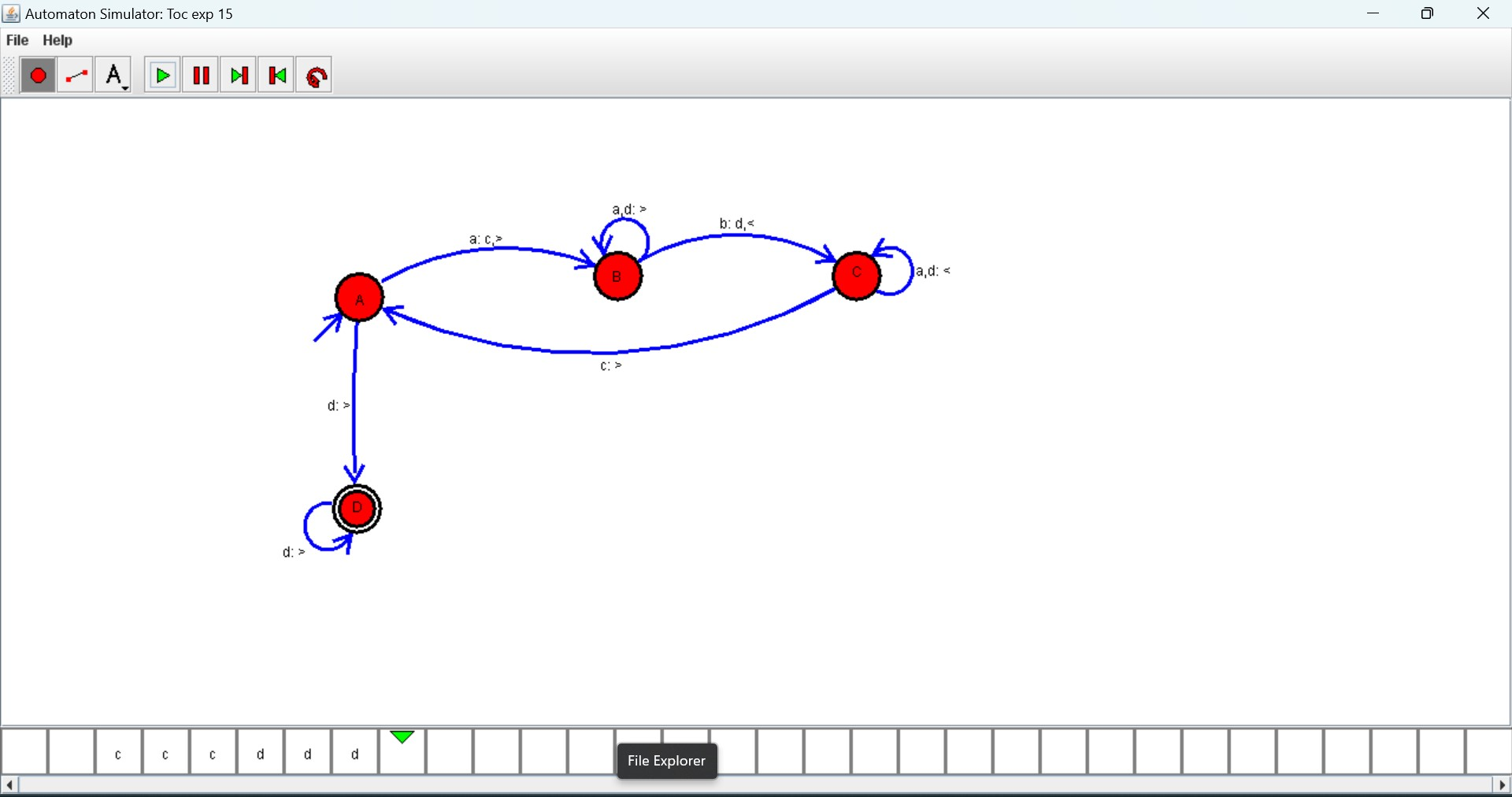
13. Design PDA using simulator to accept the input string aabb



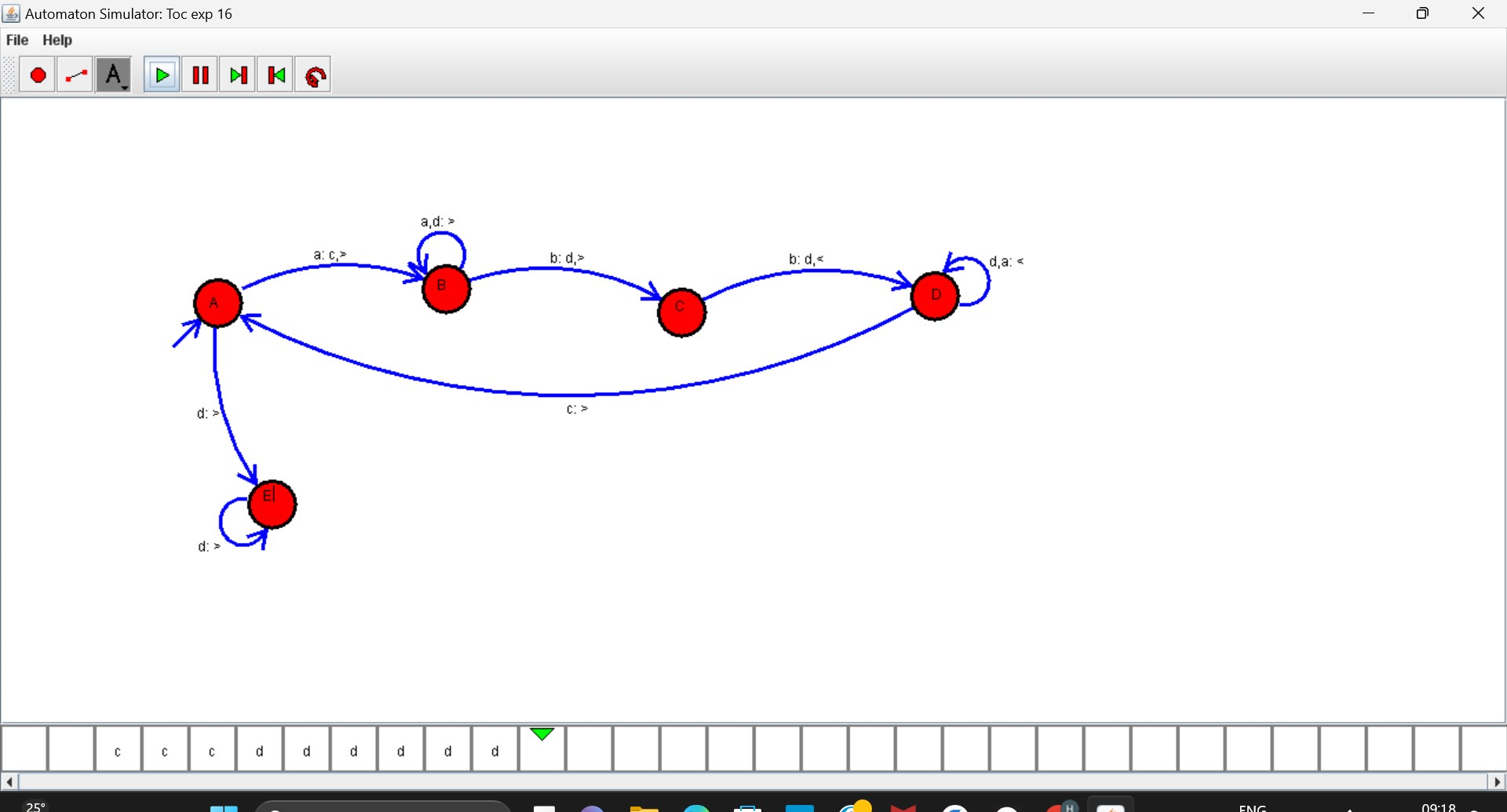
14. Design PDA using simulator to accept the input string a^nb^2n



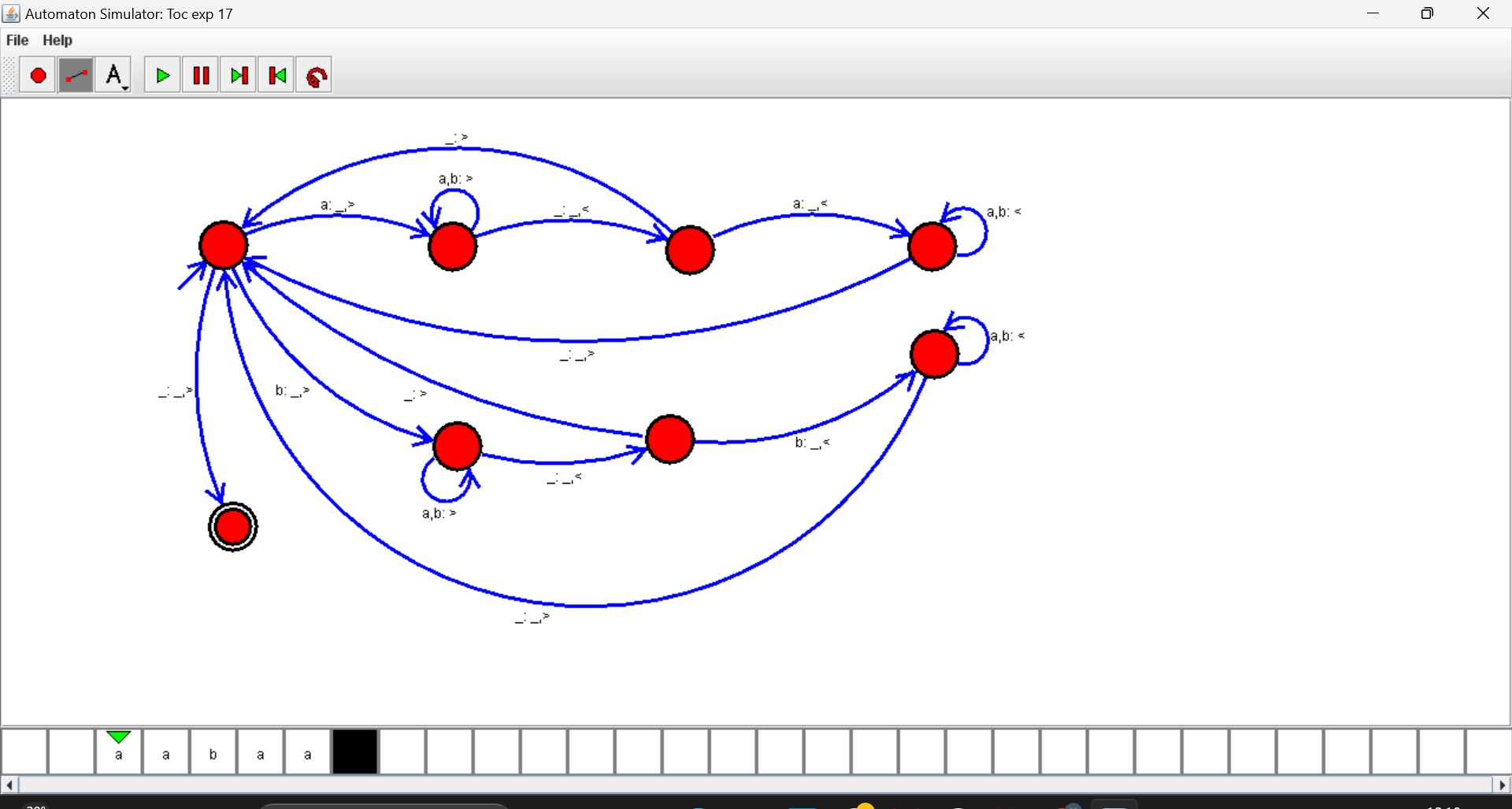
15. Design TM using simulator to accept the input string a ^nb^ n



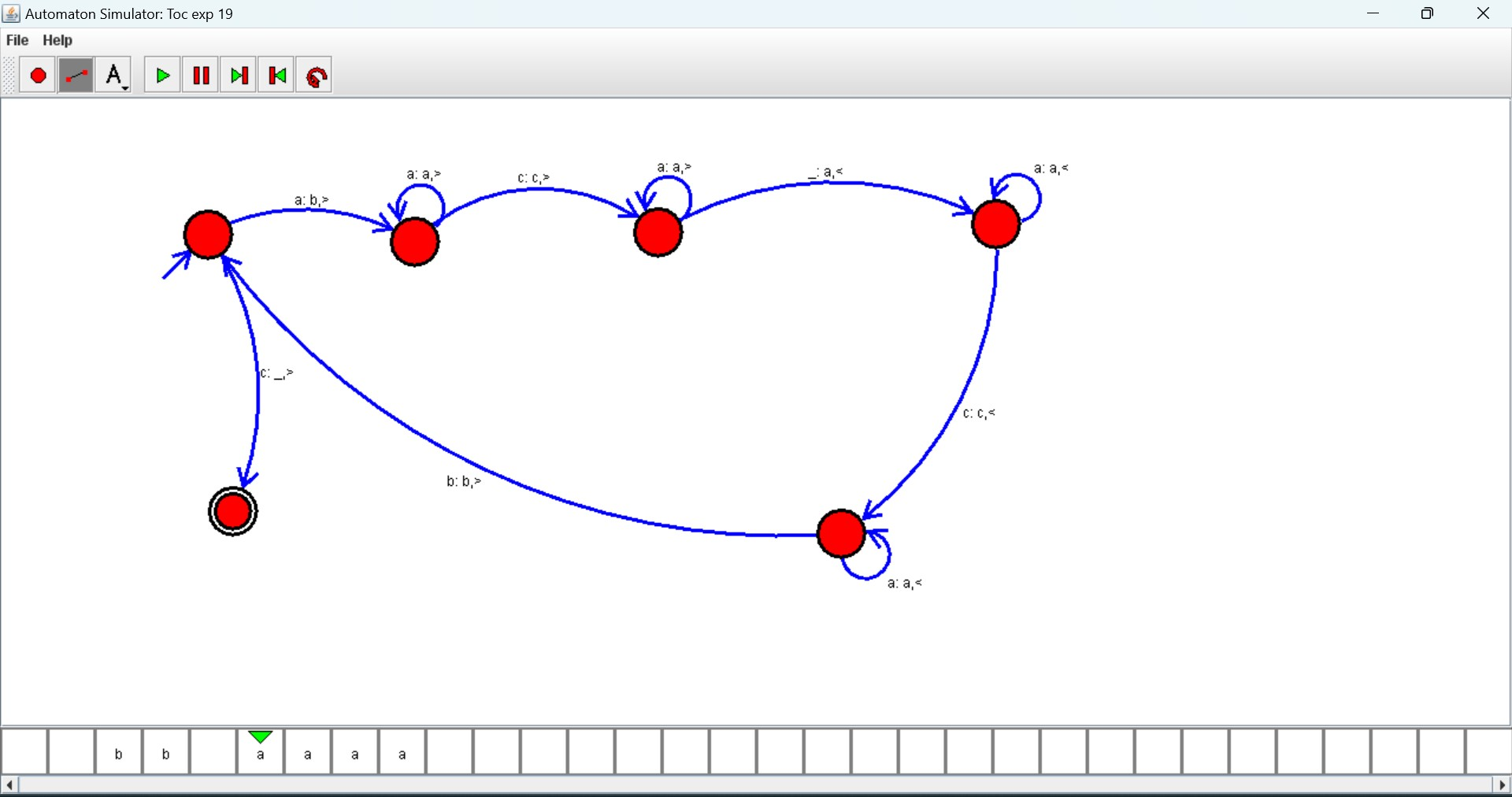
16. Design TM using simulator to accept the input string a^nb^2n



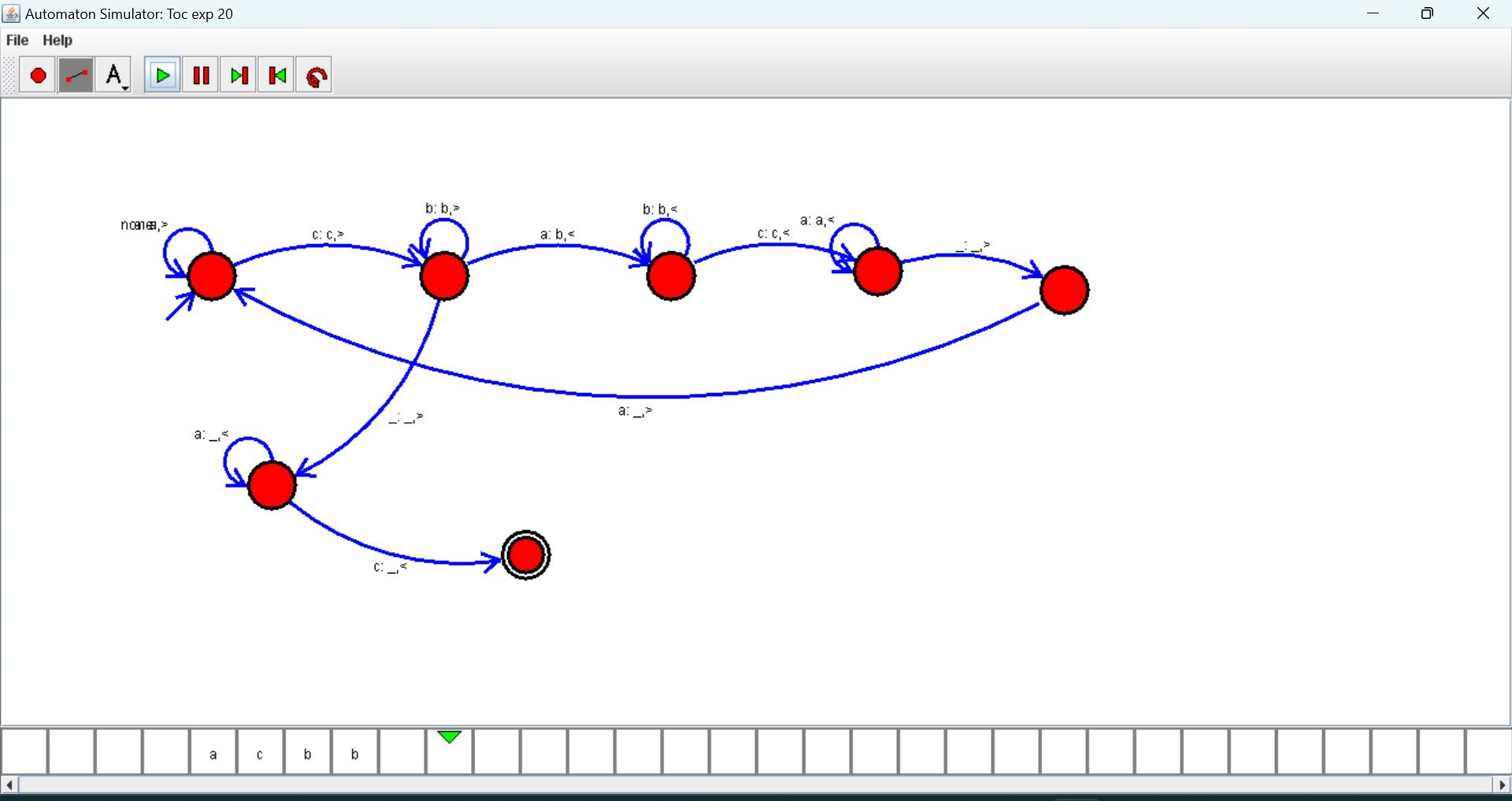
17. Design TM using simulator to accept the input string Palindrome ababa



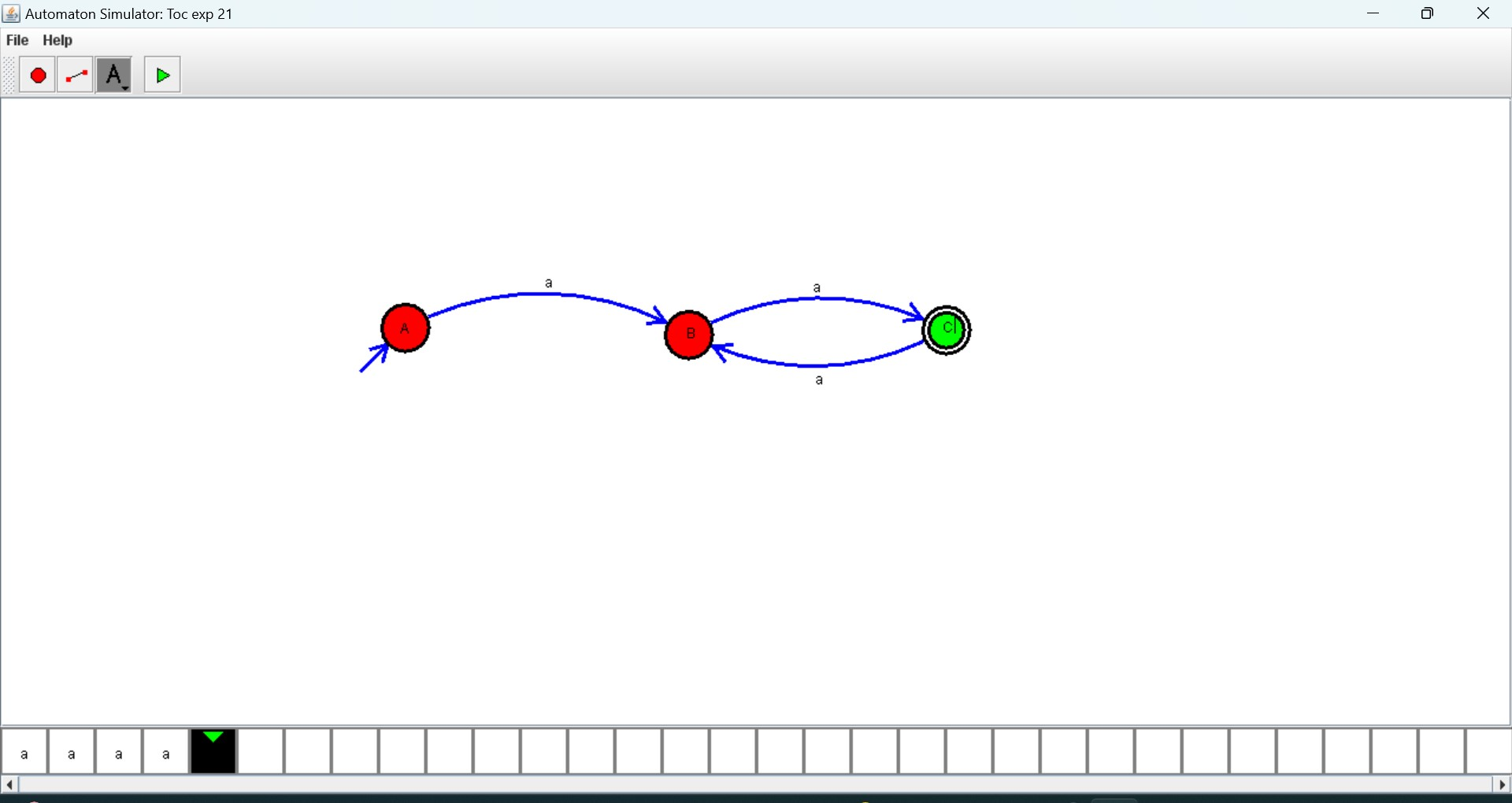
19. Design TM using simulator to perform addition of ‘aa’ and ‘aaa’



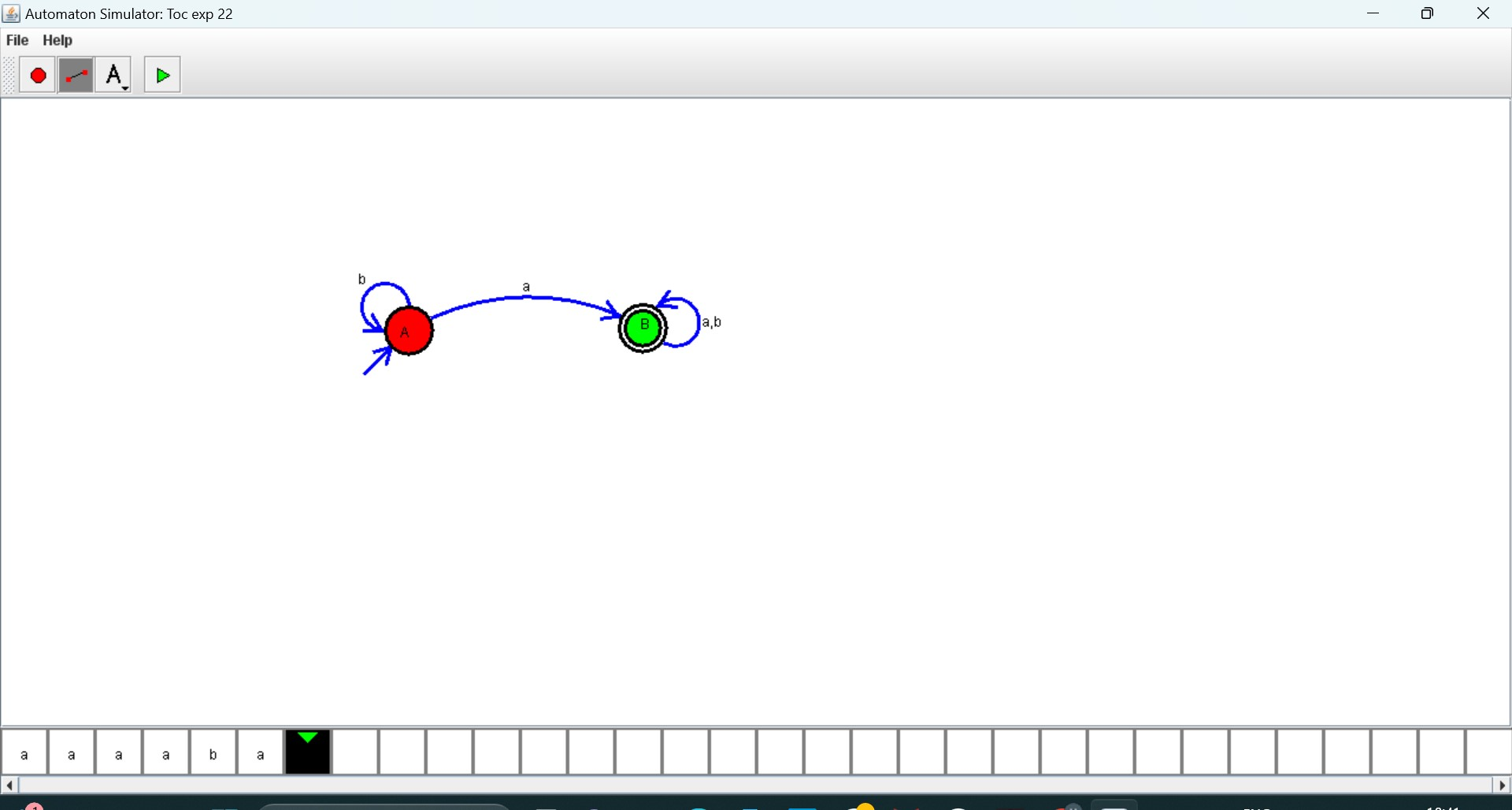
20. Design TM using simulator to perform subtraction of aaa-aa



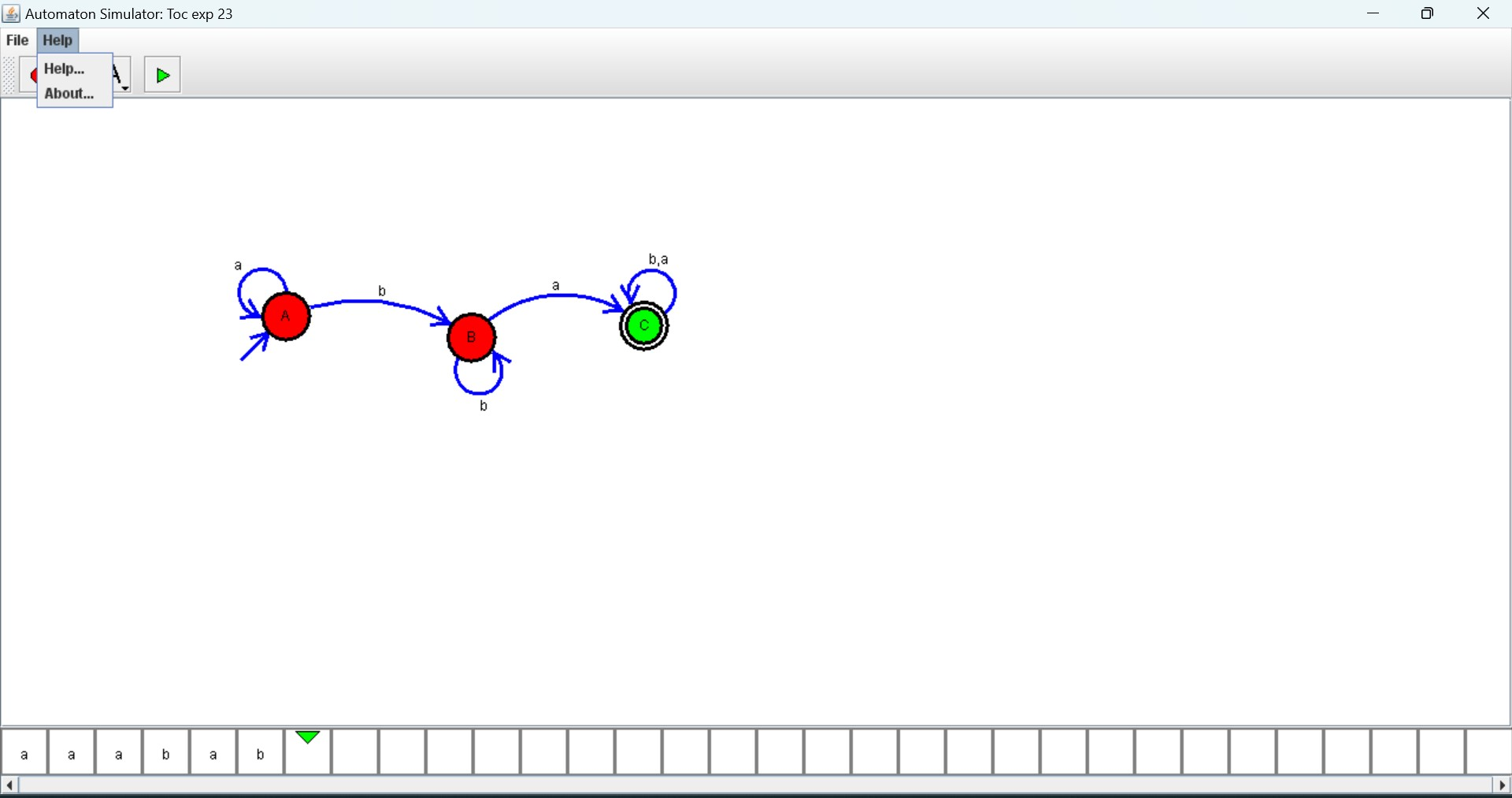
21. Design DFA using simulator to accept even number of a’s.



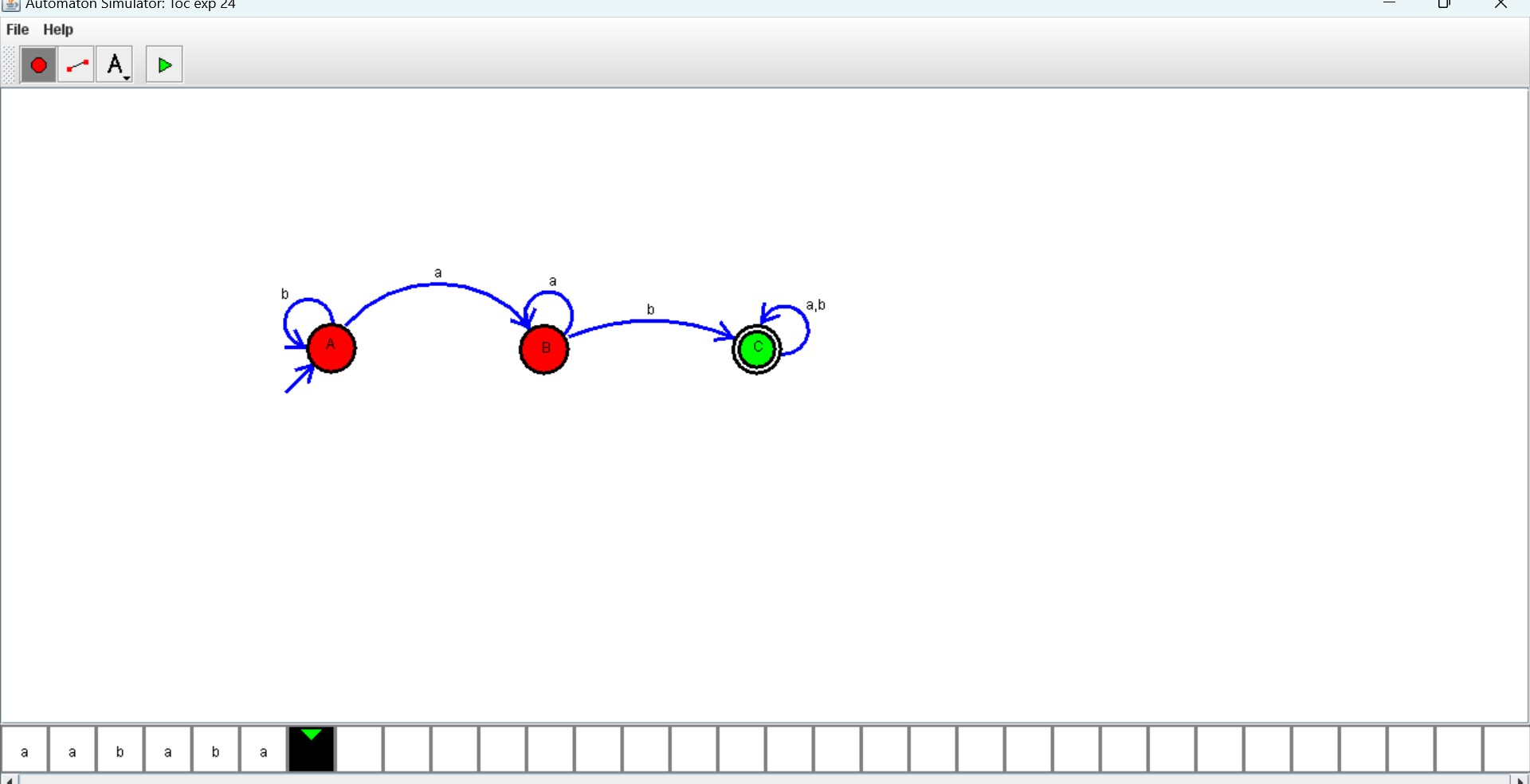
22. Design DFA using simulator to accept odd number of a’s



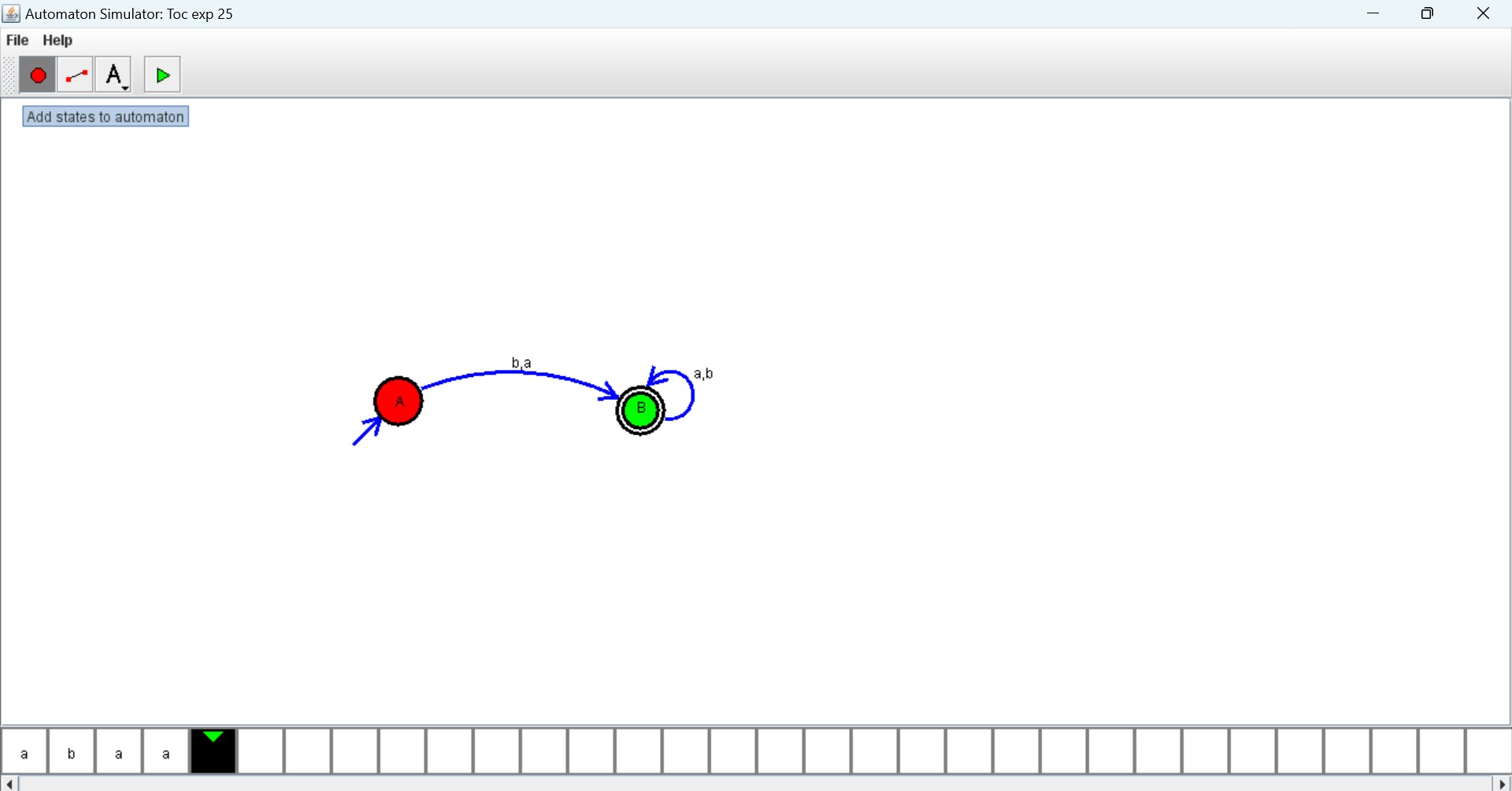
23. Design DFA using simulator to accept the string the end with ab over set {a, b)} w= aaabab



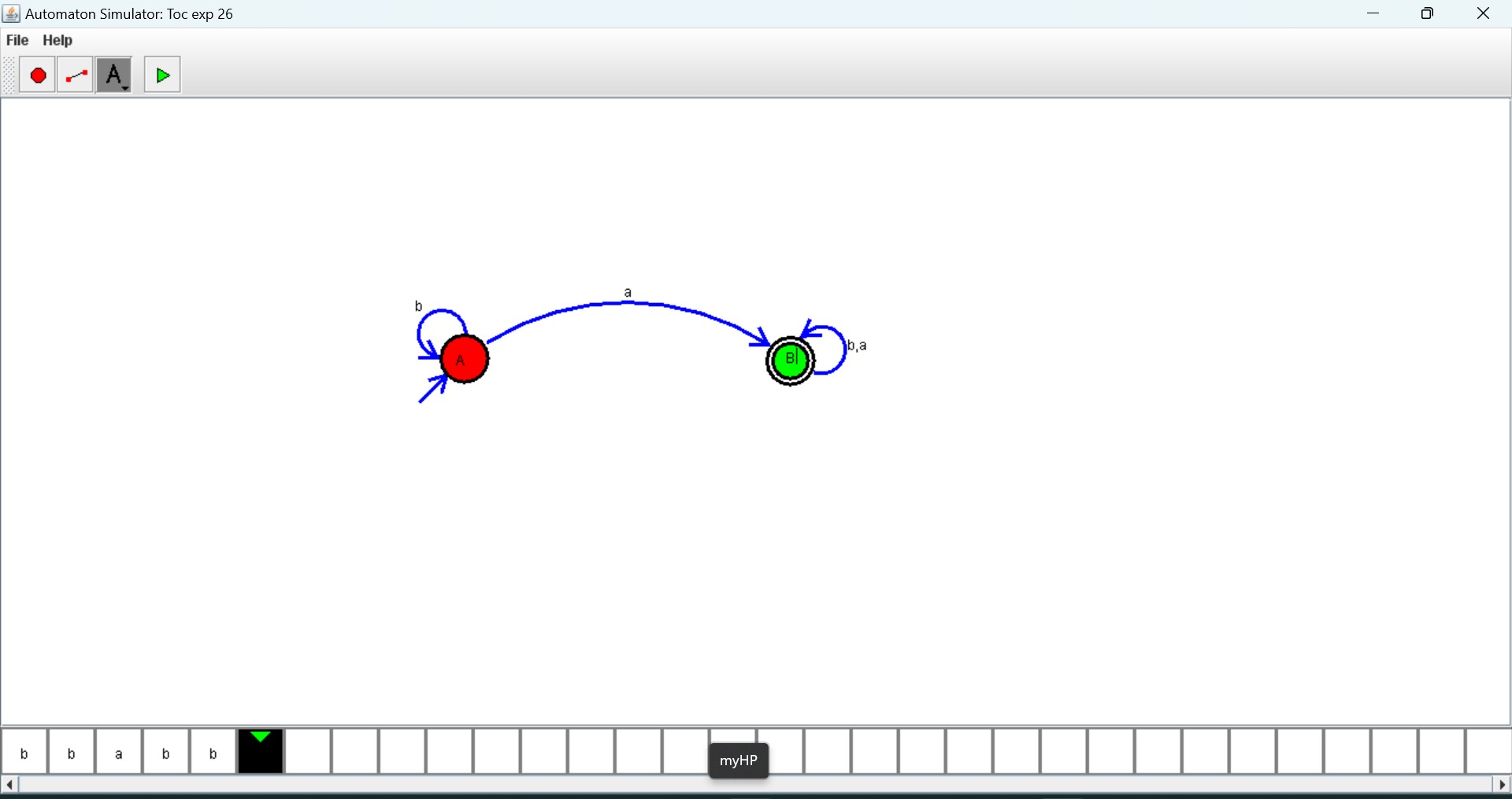
24. Design DFA using simulator to accept the string having ‘ab’ as substring over the set {a,b}



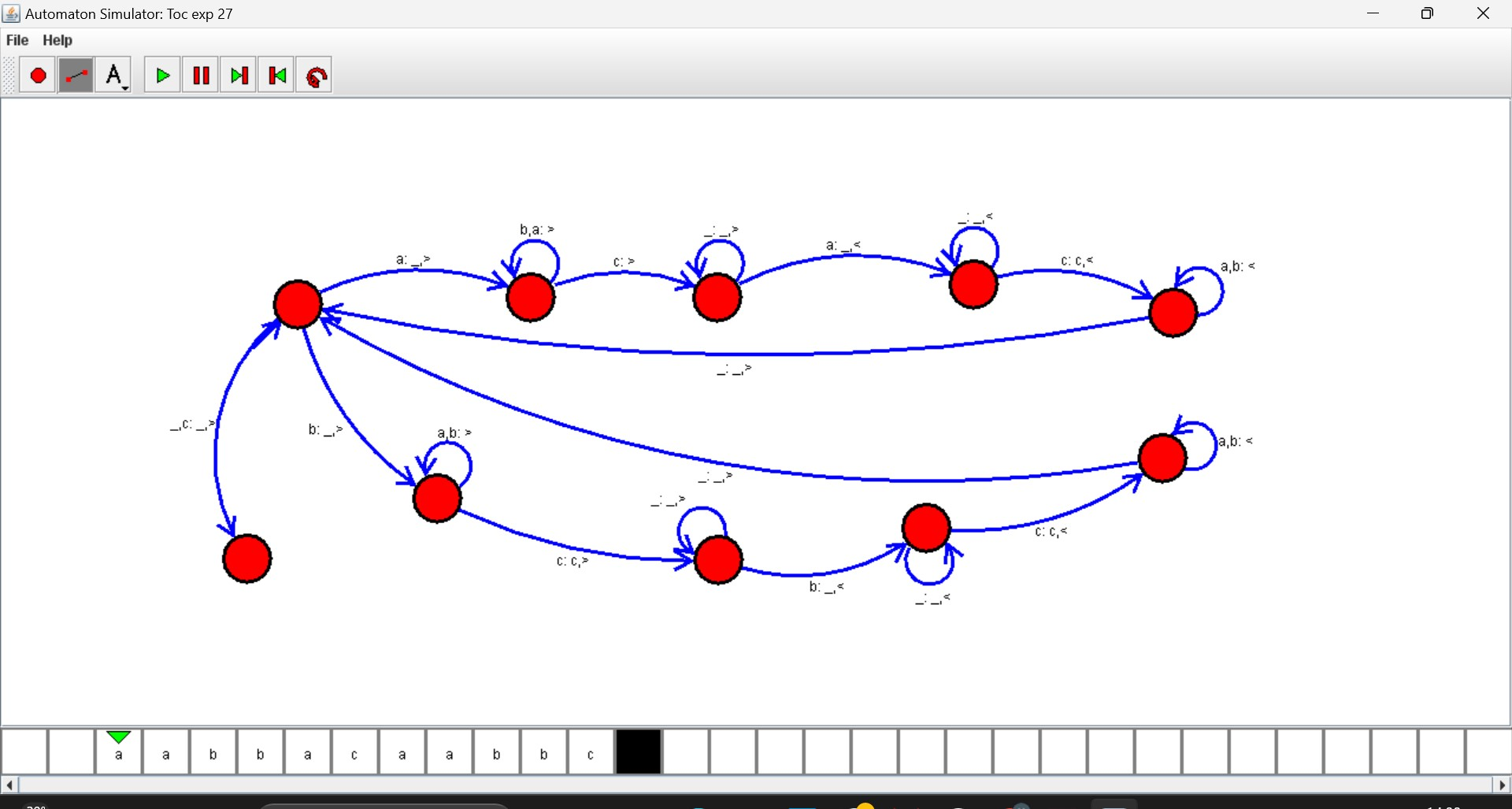
25. Design DFA using simulator to accept the string start with a or b over the set {a, b}



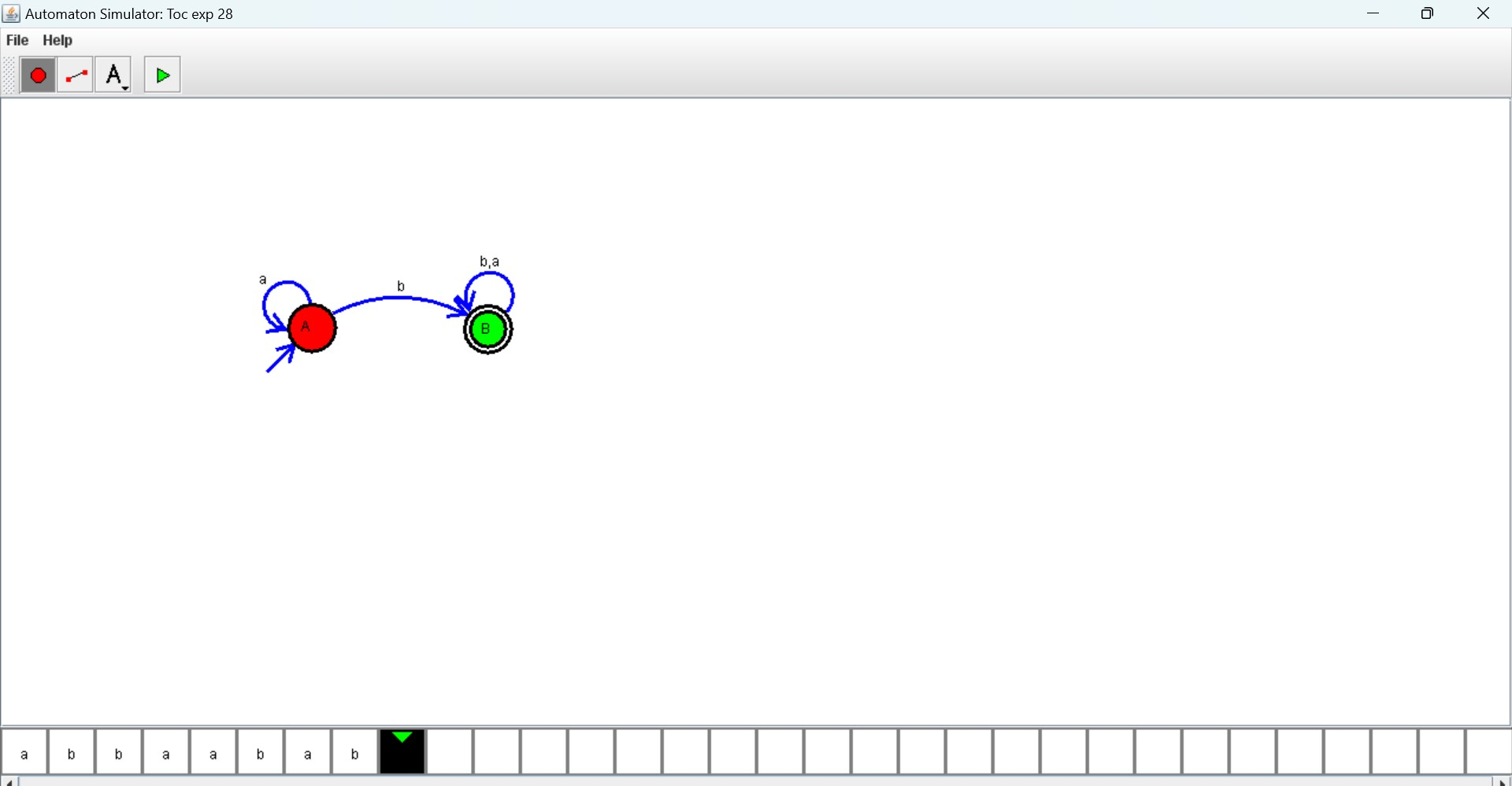
26. Design TM using simulator to accept the input string Palindrome bbabb



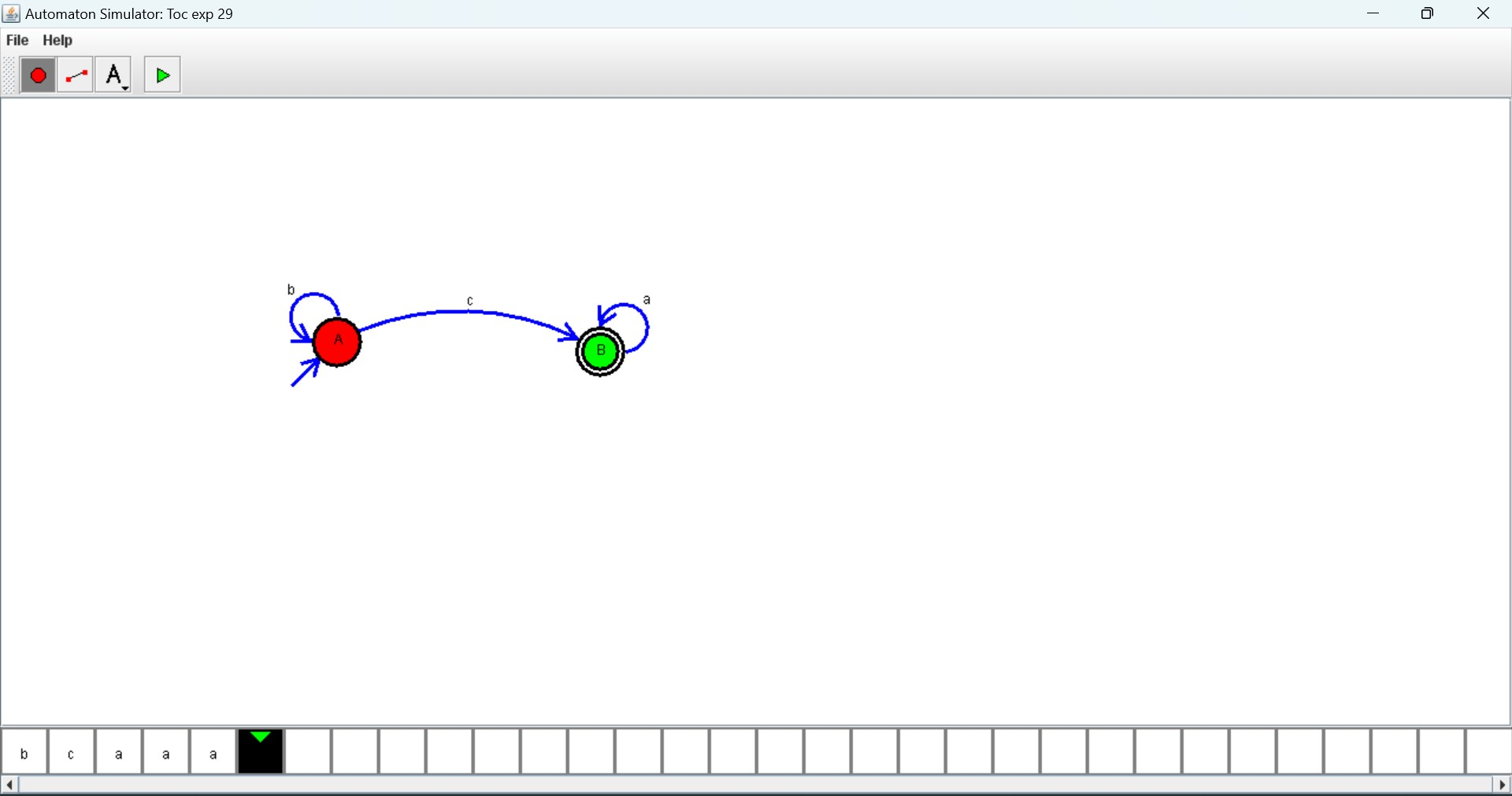
27. Design TM using simulator to accept the input string wcw



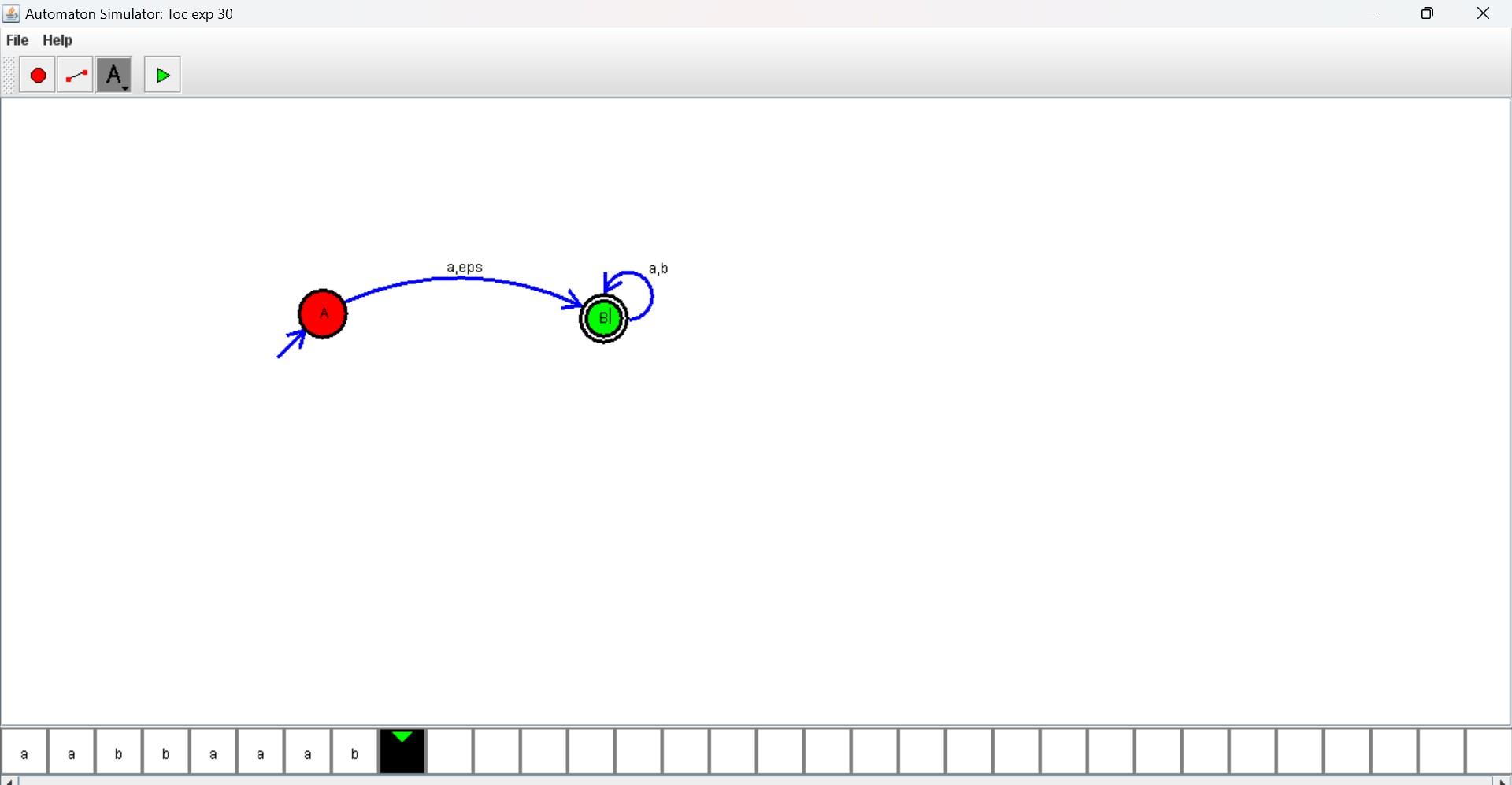
28. Design DFA using simulator to accept the string the end with ab over set {a, b) W= abbaabab



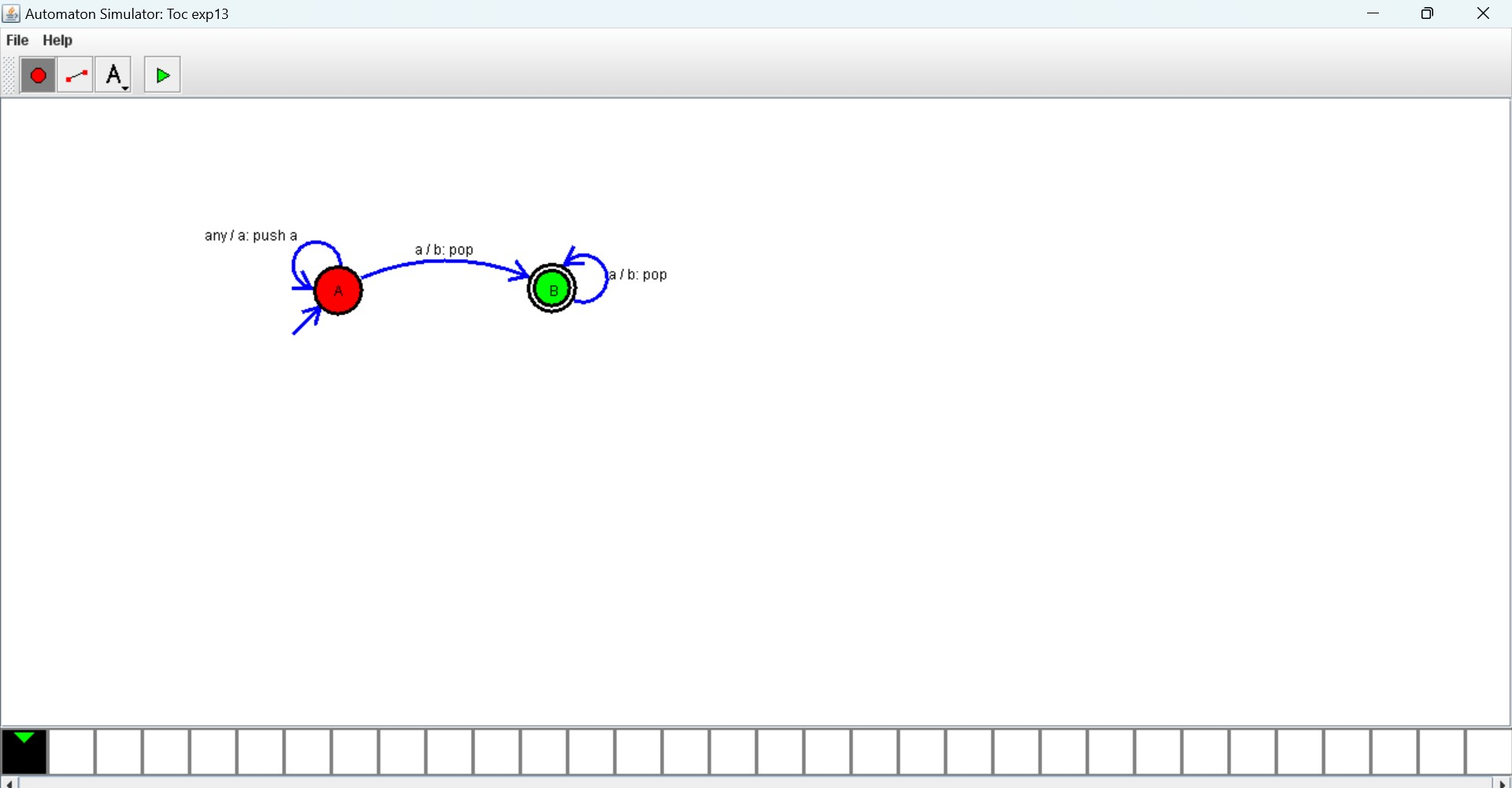
29. Design DFA using simulator to accept the input string “bc”,” c”, and” bcaaa”.



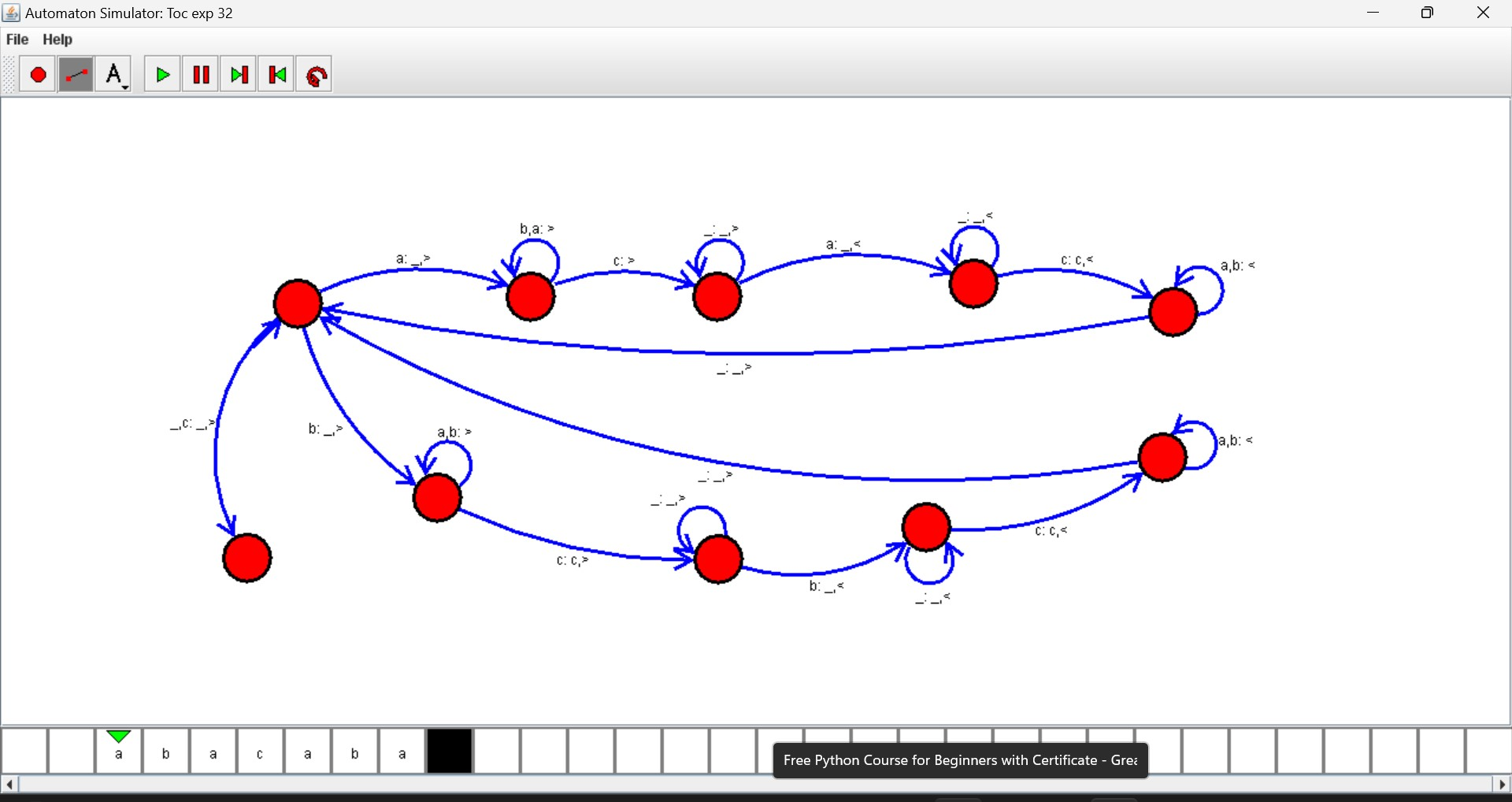
30. Design NFA to accept any number of a’s where input = {a, b}.



31. Design PDA using simulator to accept the input string a^nb^n



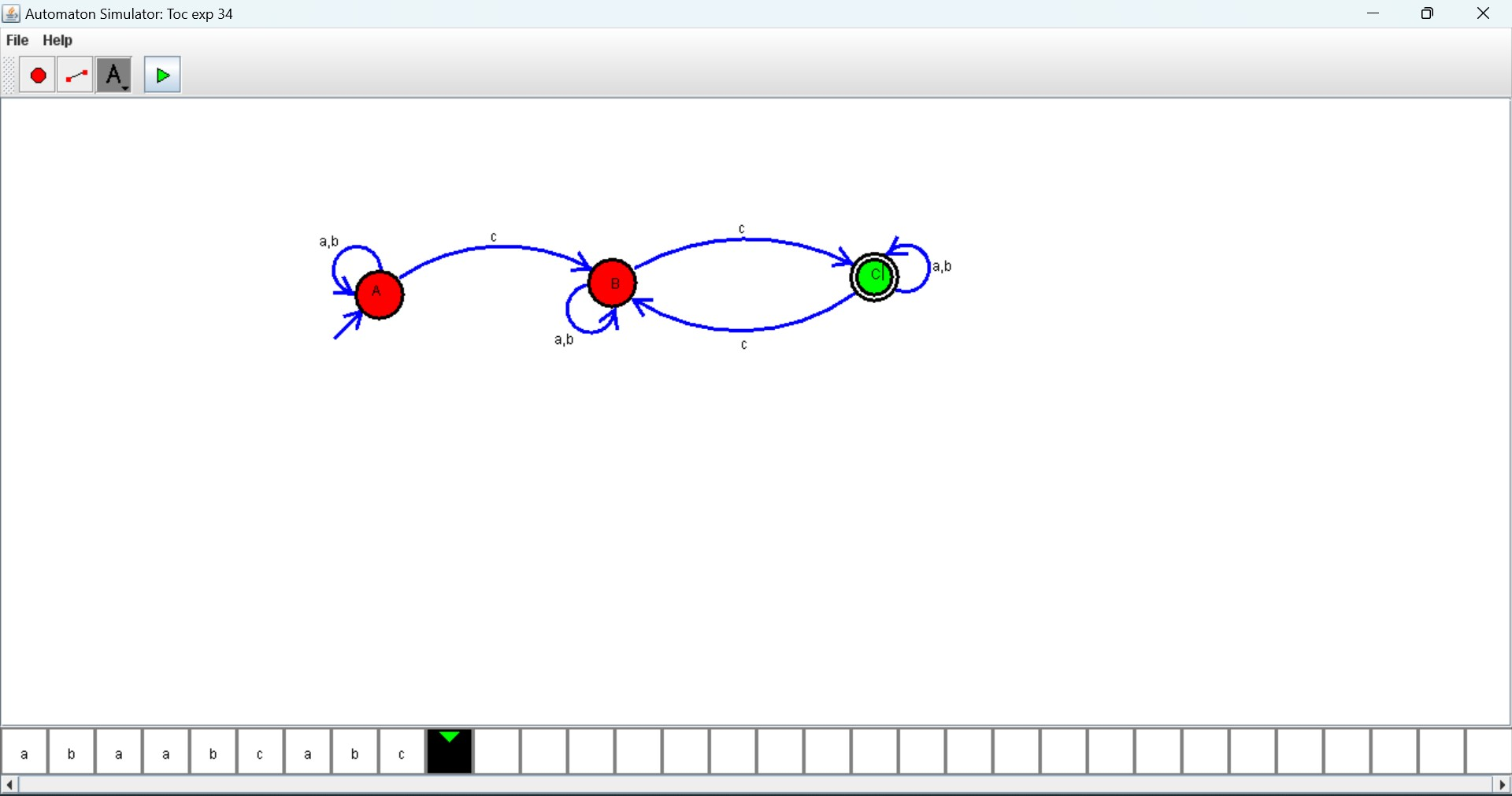
32. Design TM using simulator to perform string comparison where w = {aba aba}



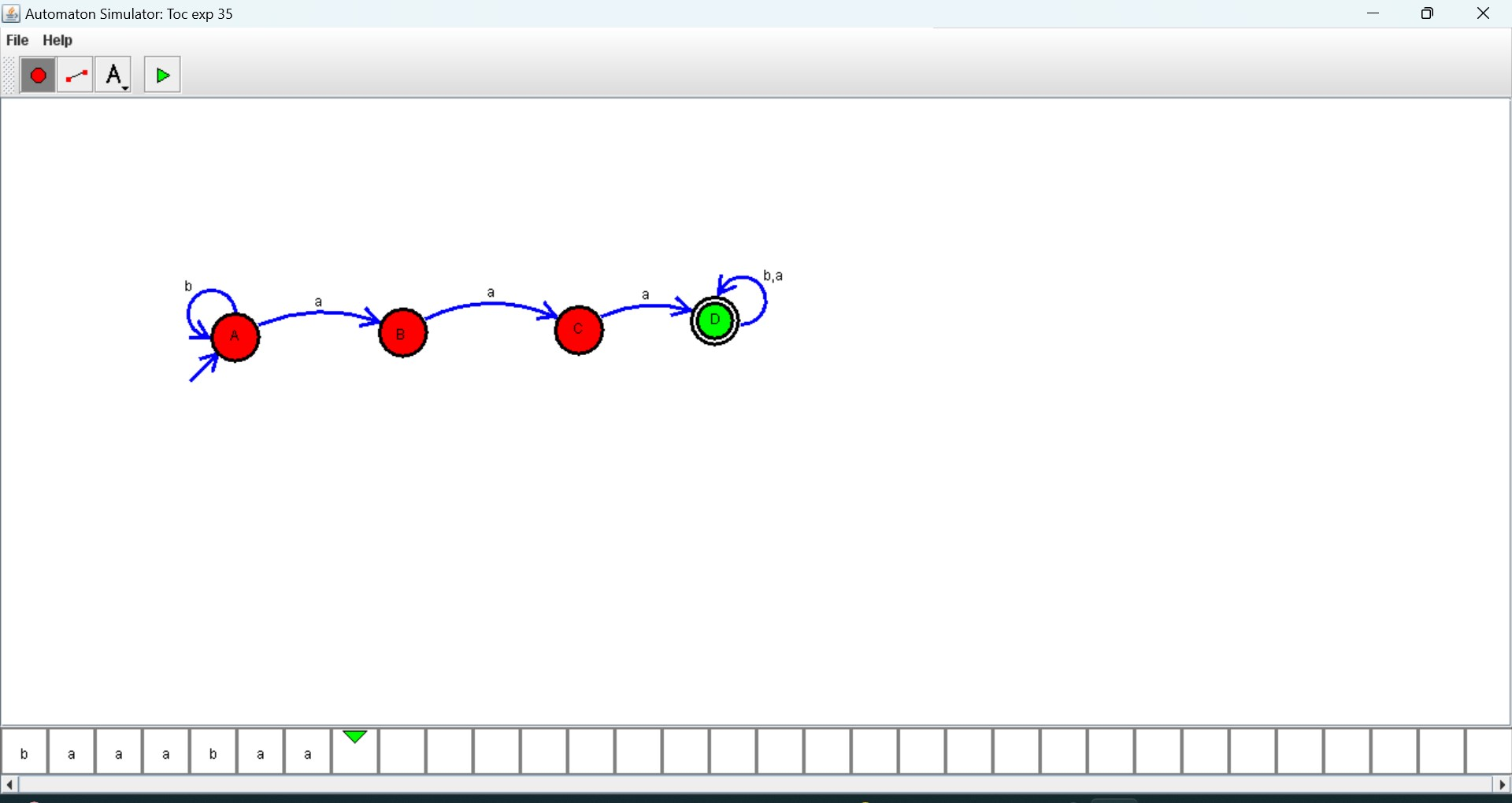
33. Design DFA using simulator to accept the string having ‘abc’ as substring over the set {a, b, c}



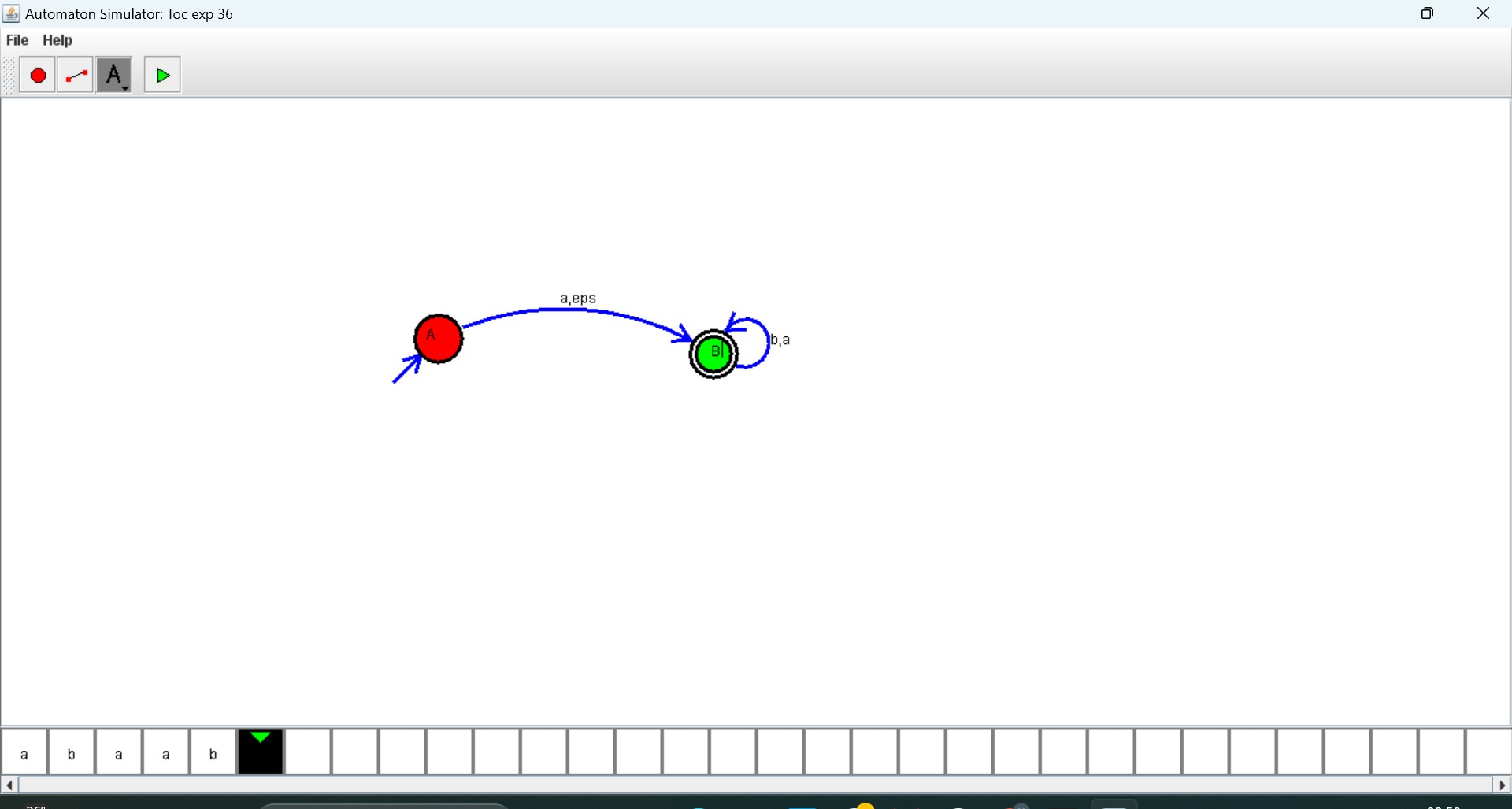
34. Design DFA using simulator to accept even number of c’s over the set {a, b, c}



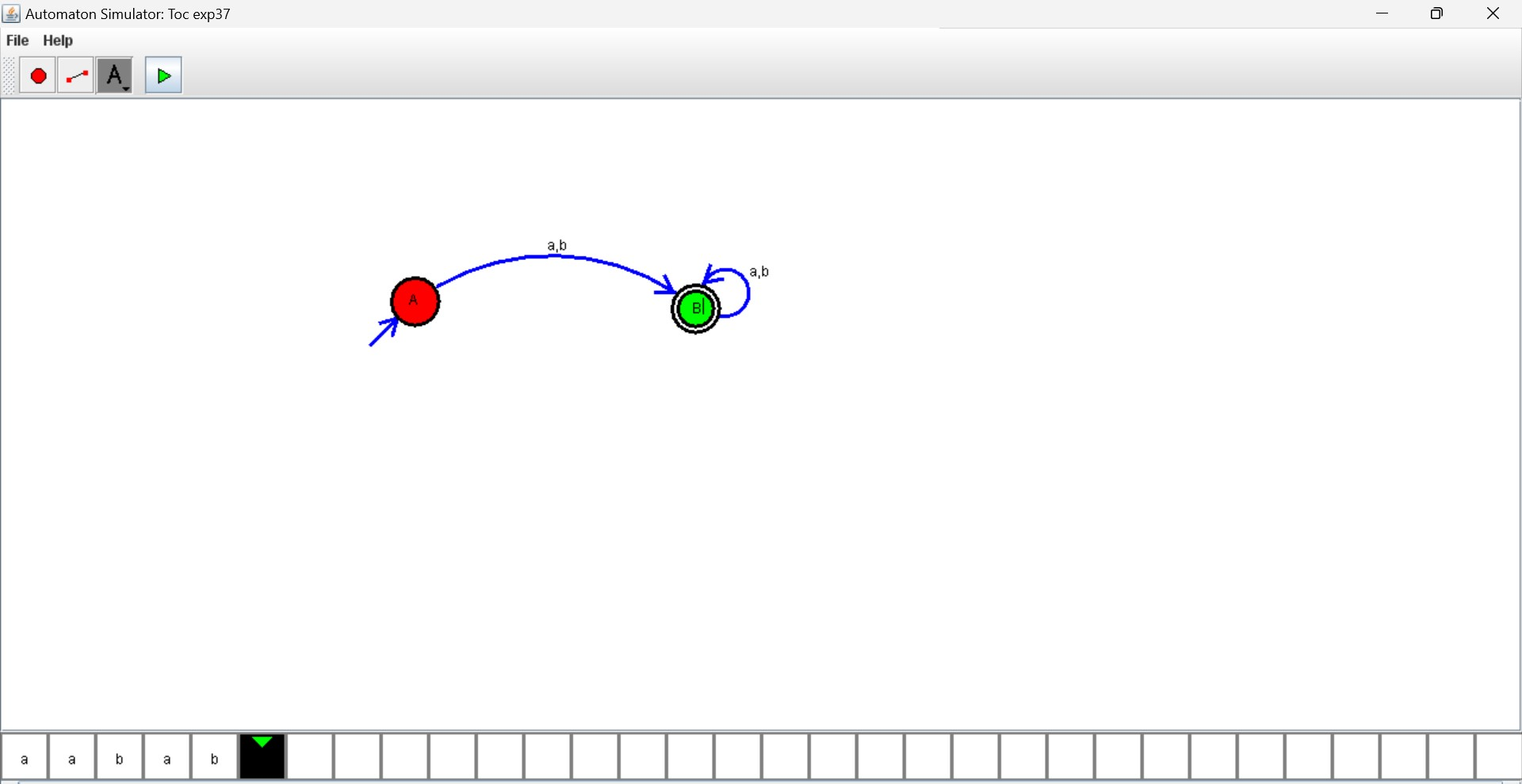
35. Design DFA using simulator to accept strings in which a’s always appear tripled over input {a, b}



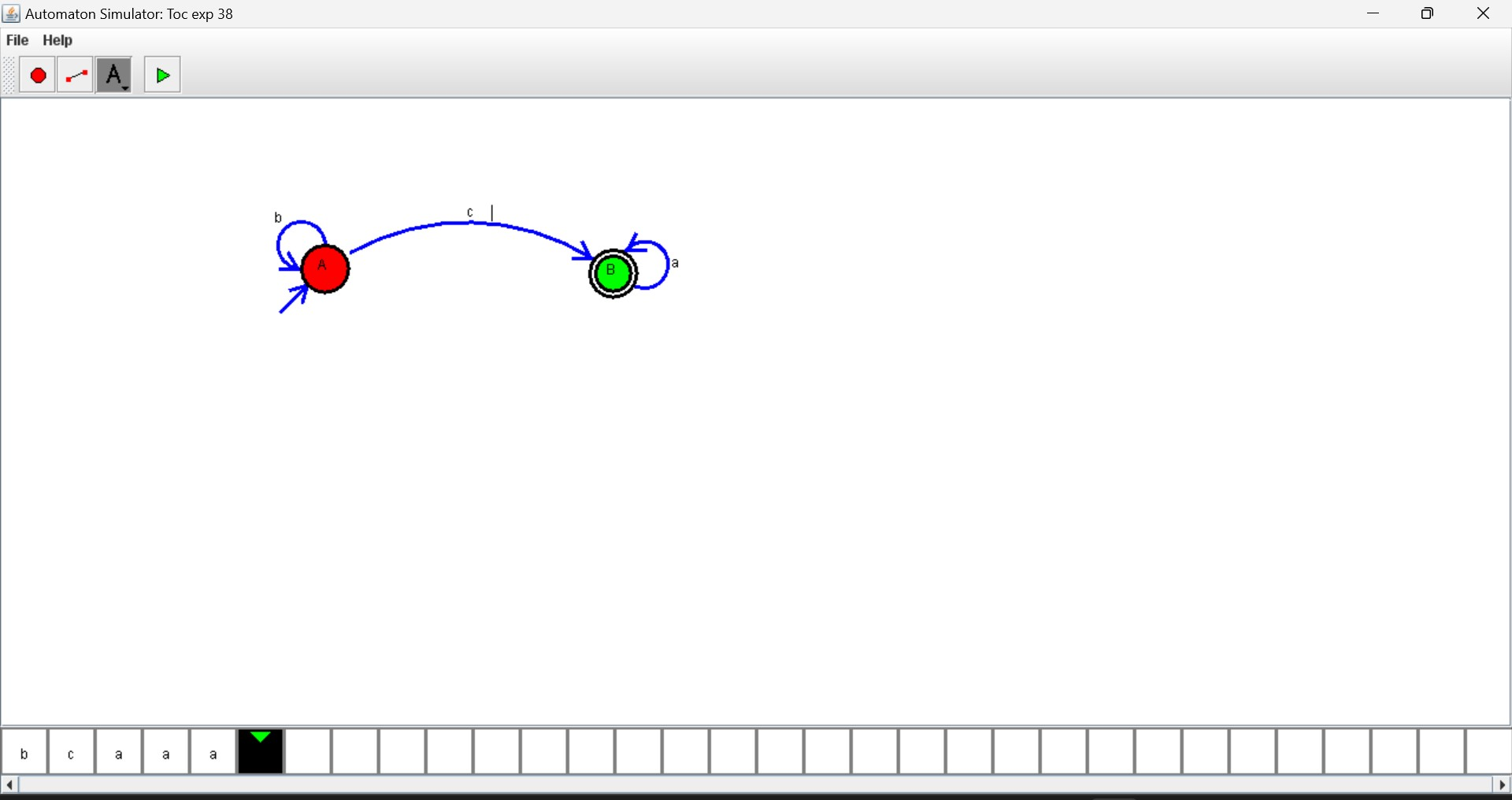
36. Design NFA using simulator to accept the string the start with a and end with b over set {a, b} and check W= abaab is accepted or not.



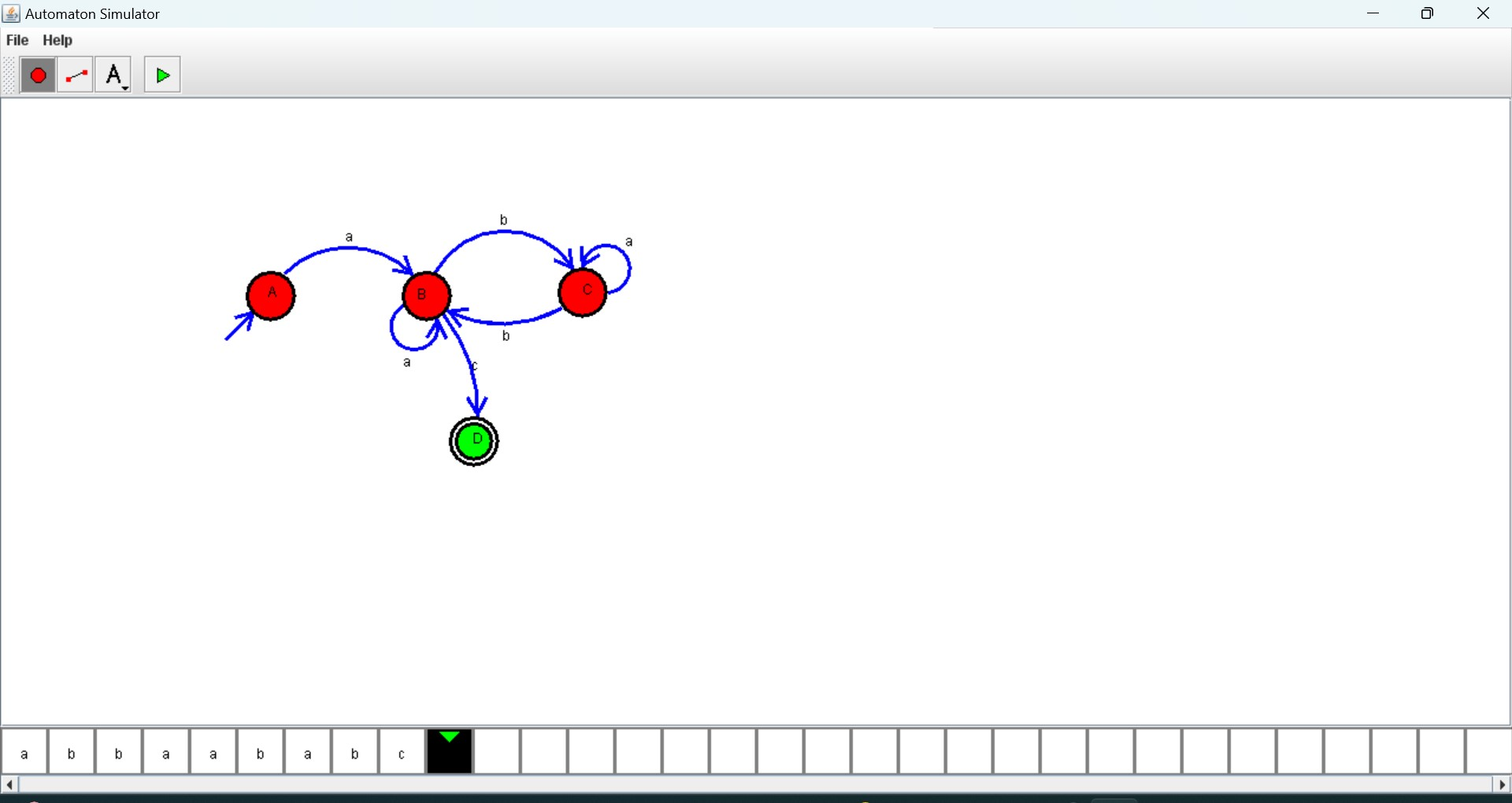
37. Design NFA using simulator to accept the string that start and end with different symbols over the input {a, b}.



38. Design NFA using simulator to accept the input string “bbc”,” c”, and” bcaaa”.



39. Design DFA using simulator to accept the string the end with abc over set {a, b, c) W= abbaababc



40. Design NFA to accept any number of b’s where input = {a, b}.

