Part II Computational Projects 2020

Project number: C3459E

Attach to front of report

13.1 Minimisation of Discrete Finite-State Automata

Question 1

$n^{nk}2^n$

2	16	$2.2\text{x}10^2$	$4.1\text{x}10^3$	10^5	$3\text{x}10^6$
2	64	$5.8\text{x}10^3$	10^6	$3.1\text{x}10^8$	$1.4\text{x}10^{11}$
2	$2.6\text{x}10^2$	$1.6\text{x}10^5$	$2.7\text{x}10^8$	$9.8\text{x}10^{11}$	$6.5\text{x}10^{15}$
2	10^3	$4.3\text{x}10^6$	$6.9\text{x}10^{10}$	$3.1\text{x}10^{15}$	$3\text{x}10^{20}$

Question 2

```
Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
1
Input path: F:\\Documents\\CATAM\\II\\Table1.txt
Input alphabet size: 2
Accessible states are: 1 3 4 5 7 8 9 10


Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
1
Input path: F:\\Documents\\CATAM\\II\\Table2.txt
Input alphabet size: 3
```

```
Accessible states are: 1 2 3 4 5 6 8 9


Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
1
Input path: F:\\Documents\\CATAM\\II\\Table3.txt
Input alphabet size: 3
Accessible states are: 1 2 3 4 5 6


Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
1
Input path: F:\\Documents\\CATAM\\II\\Table4.txt
Input alphabet size: 3
Accessible states are: 1 2 3 4 5 6 7 8 9 10 11 12 13 14
15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 31 32 33
34 35 36 37 38 39 40 41 42 43 44 45 47 48 50 51 52 53
54 55 56 57 58 59 60 61 62 64 65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
91 93 94 95 96 97 99
```

This program is $O(kn^2)$.

```
Menu:
1. Determine accessible states from DFA specified by
table
```

```
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
2
Input n: 1
Input k: 2
2


Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
2
Input n: 2
Input k: 2
48


Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
2
Input n: 3
Input k: 2
3456


Menu:
```

```
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
2
Input n: 4
Input k: 2
503808
```

Question 3

```
Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
3
Input path: F:\\Documents\\CATAM\\II\\Table1.txt
Input alphabet: ab
1 4 0
5 5 0
6 5 0
3 2 0
7 6 1
6 7 1
7 7 0


Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
```

```
7. Generate minimal (n,2) transition tables for
languages of size s
3
Input path: F:\\Documents\\CATAM\\II\\Table2.txt
Input alphabet: abc
3 3 7 0
4 4 4 1
5 6 5 0
4 4 4 0
2 2 2 1
5 5 5 0
4 2 2 0


Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
3
Input path: F:\\Documents\\CATAM\\II\\Table3.txt
Input alphabet: abc
1 2 3 1
2 3 2 0
1 2 3 0


Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
```

This algorithm has complexity O(kn^2logn)

Question 4

```
Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
4
Input n: 1
2


Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
4
Input n: 2
24


Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
4
Input n: 3
1092


Menu:
```

```
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
4
Input n: 4
119867
```

Question 5

If L(D) is infinite, it cannot have an upper bound on word length, as there would only be a finite number of words of shorter length than this upper bound.

If L(D) contains a word of length |w| >= 2n, then by pumping lemma contains a word of length n<=|w|<=2n-1. Therefore, no words of length n<=|w|<=2n-1 ==> all words are of length <n. Upper bound on length ==> finite language.

Conversely, any word w of length n<=|w|<=2n-1 can be pumped an infinite number of times, so if L(D) contains a word of length n<=|w|<=2n-1, L(D) is an infinite language.

Question 6

```
Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
5
Input path: F:\\Documents\\CATAM\\II\\Table1.txt
Input alphabet: ab
infinite


Menu:
1. Determine accessible states from DFA specified by
table
```

```
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
5
Input path: F:\\Documents\\CATAM\\II\\Table2.txt
Input alphabet: abc
42


Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
5
Input path: F:\\Documents\\CATAM\\II\\Table3.txt
Input alphabet: abc
Infinite

Question 7
Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
6
Input n: 1
f(1,2,0) = 1
f(1,2,1) = 0



Menu:
```

```
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
6
Input n: 2
f(2,2,0) = 0
f(2,2,1) = 1
f(2,2,2) = 0
f(2,2,3) = 0



Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
6
Input n: 3
f(3,2,0) = 0
f(3,2,1) = 2
f(3,2,2) = 3
f(3,2,3) = 1
f(3,2,4) = 0
f(3,2,5) = 0
f(3,2,6) = 0
f(3,2,7) = 0



Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
```

```
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
6
Input n: 4
f(4,2,0) = 0
f(4,2,1) = 4
f(4,2,2) = 17
f(4,2,3) = 21
f(4,2,4) = 15
f(4,2,5) = 7
f(4,2,6) = 2
f(4,2,7) = 2
f(4,2,8) = 0
f(4,2,9) = 0
f(4,2,10) = 0
f(4,2,11) = 0
f(4,2,12) = 0
f(4,2,13) = 0
f(4,2,14) = 0
f(4,2,15) = 0
```

If a language of alphabet size 2 and size ≥ 2^n must by pigeonhole principle contain a word of length ≥n, which must have passed through ≥n+1 states. If this can be represented by an n-state DFA, there must be a loop leading to an accepting state, which can be pumped up an arbitrary amount, and thus the language is infinite.

Question 8

```
Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s

7
Input n: 1
Input s: 0

Transition Table:
1 1 0
Language:
```

```
Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
7
Input n: 2
Input s: 1

Transition Table:
2 2 1
2 2 0
Language:
empty word



Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
7
Input n: 3
Input s: 1

Transition Table:
2 3 0
2 2 0
2 2 1
Language:
b

Transition Table:
2 3 0
3 3 1
3 3 0
Language:
a
```

```
Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
7
Input n: 3
Input s: 3

Transition Table:
2 2 1
3 3 1
3 3 0
Language:
empty word
a
b
```

```
Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
7
Input n: 4
Input s: 1

Transition Table:
2 3 0
2 2 0
2 4 0
2 2 1
Language:
bb
```

```
Transition Table:
2 3 0
2 2 0
4 2 0
2 2 1
Language:
ba

Transition Table:
2 3 0
3 4 0
3 3 0
3 3 1
Language:
ab

Transition Table:
2 3 0
4 3 0
3 3 0
3 3 1
Language:
aa



Menu:
1. Determine accessible states from DFA specified by
table
2. Compute the number of (n,k)-DFAs with no
inaccessible states
3. Apply Hopcroft's table-filling algorithm to
transition table
4. Output number of (n,2)-DFA languages
5. Compute language size from transition table
6. Compute f(n,2,s)
7. Generate minimal (n,2) transition tables for
languages of size s
7
Input n: 4
Input s: 7

Transition Table:
2 2 1
4 4 1
3 3 0
3 3 1
Language:
empty word
a
b
aa
```

```
ab
ba
bb

Transition Table:
2 2 1
3 3 1
4 4 1
4 4 0
Language:
empty word
a
b
aa
ab
ba
bb
```

Question 9

The number of minimal (n,2)-DFAs corresponding to a language of size 1 is 0 for $n < 2$, and 2^{n-2} thereafter. This holds as once a word has been established to not be the word of the language, it enters the same state from which no accept state is inaccessible (and would thus be equivalent to any other such state, contradicting minimality). Thus, there are n-1 states with which to ascertain whether or not a word is the word of the language. These must form a chain as any branching would require another accept state, thus allowing multiple words in the language. Therefore, the word of the language must have length n-2, and there are 2^{n-2} such words. In the case n=2, the only word in the language is the empty word.

The only language with $s=2^{n-1}-1$ is the language that contains all the words of length < n-1. A minimal DFA corresponding to a finite language must have a unique state from which no accept state is accessible, as beyond a certain length, a word cannot be in the language. This state must also be accessible from any other state as otherwise an arbitrarily large word segment could be processed from that state and still possibly be accepted. To maximise the size of the language, every other state would be an accept state, but without loops to preserve finiteness, so these states form a chain. Any branching would decrease the maximum length, and thus size of the language. The maximum finite language from a minimal DFA therefore comes from chaining together n-1 accept states, with a sole reject state and the end that loops to itself. This is unique up to relabelling of states. As this is maximal for finite languages, there are no finite languages of size $\geq 2^{n-1}$ generated by a minimal (n,2) DFA.

Source code

```
// ConsoleApplication1.cpp : Defines the entry point
for the console application.
//

#include "stdafx.h"
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>

#define intMaxLanguageSize 63
#define intMaxStates 255

using namespace std;

struct state{
     bool accept;
     int transition[intMaxLanguageSize];
};


int intEmpty = -858993460;
int size(int arrInput[],int intNull = intEmpty)
{
     int intCount = 0;
     while (true)
     {
          if (arrInput[intCount] == intNull)
          {
               return intCount;
          }
          intCount = intCount + 1;
     }

}

/*int size(state arrDFA[])
{
     int intCount = 0;
     while (true)
     {
          if (arrDFA[intCount].accept != false)
          {
               if (arrDFA[intCount].accept != true)
               {
                    return intCount;
               }
          }
          intCount = intCount + 1;
     }
}*/
```

```
int size(state arrDFA[])
{
     int intCount = 0;
     while (true)
     {
          if (arrDFA[intCount].transition[0] < 0)
          {
               return intCount;
          }
          intCount = intCount + 1;
     }
}

bool checkElement(int intElement, int arrTest[])
{
     bool boolOutput = false;
     int intSize = size(arrTest);
     for (int intIter = 0; intIter < intSize; intIter =
intIter + 1)
     {
          if (arrTest[intIter] == intElement)
          {
               boolOutput = true;
          }
     }
     return boolOutput;
}

void bubbleSort(int arrList[])
{
     int intSize = size(arrList);
     bool boolCont = true;
     int intStore;
     while (boolCont)
     {
          boolCont = false;
          for (int intIter = 1; intIter < intSize;
intIter = intIter + 1)
          {
               if (arrList[intIter - 1] >
arrList[intIter])
               {
                    intStore = arrList[intIter - 1];
                    arrList[intIter - 1] =
arrList[intIter];
                    arrList[intIter] = intStore;
                    boolCont = true;
               }
          }
     }
}
```

```
void importCSV(string* pointFirstElement, string
strFilename, char charDelim = ',') // Function to
import a .csv file to an array of strings
{
     // pointFirstElement = Pointer for the first
element of the array
     // strFilename       = Filename of the .csv file
to be imported
     // charDelim         = CSV Delimiter

     int intIncrement = 0; // Parameter for increment
     string strLine; // Paramater for reading each line

     ifstream csvFile(strFilename.c_str()); // Generate
a stream from the file
     while (getline(csvFile, strLine)) // Iterate
through the file, reading each line to strLine
     {
          istringstream issLine(strLine); // Generate a
stream from the line
          string strEntry; // Parameter for reading
each entry
          while (getline(issLine, strEntry, charDelim))
// Iterate through the line, reading each entry to
strEntry
          {
               *(pointFirstElement + intIncrement) =
strEntry; // Write the value of strEntry to the
appropriate element
               intIncrement = intIncrement + 1; //
Increment the second index
          }
     }
}

void buildDFAFromFile(state arrDFA[], string
strFilename, int intAlphabetSize, char charDelim)
{
     string arrImport[1023];
     importCSV(&arrImport[0], strFilename, charDelim);
     int intIter = 0;
     while
(arrImport[intIter*(intAlphabetSize+1)].length() > 0)
     {
          for (int intCount = 0; intCount <
intAlphabetSize; intCount = intCount + 1)
          {
               arrDFA[intIter].transition[intCount] =
stoi(arrImport[intIter*(intAlphabetSize + 1) +
intCount])-1;
          }
          arrDFA[intIter].accept = 
stoi(arrImport[intIter*(intAlphabetSize + 1) + 
intAlphabetSize]);
          intIter = intIter + 1;
     }
}

int pow(int intBase, int intExp)
{
     int intReturn = 1;
     for (int intIter = 0; intIter < intExp; intIter = 
intIter + 1)
     {
          intReturn = intReturn * intBase;
     }
     return intReturn;
}

/*void buildDFAFromInt(state arrDFA[], int intInput, 
int intAlphabetSize, int intStates)
{
     //structure of intInput: each state has k digits, 
corresponding to the destinations of each character, 
followed by a binary digit determining accept
     int intRead = 0;
     int intDenom;
     intInput = intInput % 
(pow((intAlphabetSize*intStates), 
intStates)*pow(intStates, 2));
     for (int intIter = 0; intIter < intStates; intIter 
= intIter + 1)
     {
          for (int intSubIter = 0; intSubIter < 
intAlphabetSize; intSubIter = intSubIter + 1)
          {
               intDenom = 
pow((intAlphabetSize*(intStates-intIter)-intSubIter-1), 
intStates)*pow(intStates-intIter, 2);
               intRead = intInput / intDenom;
               arrDFA[intIter].transition[intSubIter] = 
intRead;
               intInput = intInput - intRead * 
intDenom;
          }
          intDenom = pow((intAlphabetSize*(intStates - 
intIter - 1) - 1), intStates)*pow(intStates - intIter, 
2);
          intRead = intInput / intDenom;
          arrDFA[intIter].accept = intRead;
          intInput = intInput - intRead * intDenom;
     }
}*/
```

```
void readDFA(state arrDFA[], string strAlphabet)
{
     int intAlphabetSize = strAlphabet.length();
     int intDFASize = size(arrDFA);
     for (int intIter = 0; intIter < intDFASize;
intIter = intIter + 1)
     {
          cout << "State " << intIter + 1 << '\n';
          cout << "Accept: " << arrDFA[intIter].accept
<< '\n';
          for (int intSubIter = 0; intSubIter <
intAlphabetSize; intSubIter = intSubIter + 1)
          {
               cout << strAlphabet[intSubIter] << " -->
" << arrDFA[intIter].transition[intSubIter] + 1 <<
'\n';
          }
     }
}


void buildDFAFromInt(state arrDFA[], int intInput, int
intAlphabetSize, int intStates)
{
     //structure of intInput: first n digits correspond
to accept properties of the n states, followed by nk
digits corresponding to the k destinations from each of
the n states

     intInput = intInput % (pow(intStates,
intStates*intAlphabetSize)*pow(2, intStates));
     int intAccept = intInput / pow(intStates,
intStates*intAlphabetSize);
     intInput = intInput % pow(intStates,
intStates*intAlphabetSize);

     for (int intIter = 0; intIter < intStates; intIter
= intIter + 1)
     {
          arrDFA[intIter].accept = intAccept /
pow(2,intStates-intIter-1);
          intAccept = intAccept % pow(2, intStates -
intIter - 1);
          for (int intSubIter = 0; intSubIter <
intAlphabetSize; intSubIter = intSubIter + 1)
          {
               arrDFA[intIter].transition[intSubIter] =
intInput / pow(intStates, (intStates-
intIter)*intAlphabetSize-intSubIter-1);
               intInput = intInput % pow(intStates,
(intStates - intIter)*intAlphabetSize - intSubIter -
1);
          }
     }

}


int getIndex(int intElement, int arrList[])
{
     int intLength = size(arrList);
     for (int intIter = 0; intIter < intLength; intIter
= intIter + 1)
     {
          if (intElement == arrList[intIter])
          {
               return intIter;
          }
     }
}

int dfaToInt(state arrDFA[], int intAlphabetSize, int
intStates, bool boolAscend = false)
{
     int intAccept = 0;
     int intOutput = 0;
     int arrOrder[intMaxStates];
     int intIndex = 1;
     arrOrder[0] = 0;
     if (boolAscend)
     {
          for (int intState = 0; intState < intStates;
intState = intState + 1)
          {
               for (int intDest = 0; intDest <
intAlphabetSize; intDest = intDest + 1)
               {
                    if
(!checkElement(arrDFA[intState].transition[intDest],
arrOrder))
                    {
                         arrOrder[intIndex] =
arrDFA[intState].transition[intDest];
                         intIndex = intIndex + 1;
                         if (intIndex == intStates)
                         {
                              goto reorder;
                         }
                    }
               }
          }
     reorder:
          state arrCopy[intMaxStates];
          int arrInverse[intMaxStates];
          for (int intState = 0; intState < intStates;
intState = intState + 1)
```

```
          {
               arrCopy[intState] = arrDFA[intState];
               arrInverse[intState] =
getIndex(intState, arrOrder);
          }
          for (int intState = 0; intState < intStates;
intState = intState + 1)
          {
               arrDFA[intState] =
arrCopy[arrOrder[intState]];
               for (int intTrans = 0; intTrans <
intAlphabetSize; intTrans = intTrans + 1)
               {

     arrDFA[intState].transition[intTrans] =
arrInverse[arrDFA[intState].transition[intTrans]];
               }
          }
     }

     for (int intState = 0; intState < intStates;
intState = intState + 1)
     {
          intAccept = intAccept +
(arrDFA[intState].accept*pow(2, intStates - intState -
1));
          for (int intIter = 0; intIter <
intAlphabetSize; intIter = intIter + 1)
          {
               intOutput = intOutput +
(arrDFA[intState].transition[intIter] * pow(intStates,
(intStates - intState)*intAlphabetSize - intIter - 1));
          }
     }
     intOutput = intOutput + intAccept * pow(intStates,
intStates*intAlphabetSize);
     return intOutput;
}

int lookup(char charInput, string strReference)
{
     for (int intLocation = 0; intLocation <
strReference.length(); intLocation = intLocation + 1)
     {
          if (strReference[intLocation] == charInput)
          {
               return intLocation;
          }
     }
     return 0;
}
```

```
bool emulateDFA(state arrDFA[], string strInput, string
strAlphabet)
{
     int intState = 0;
     for (int intIter = 0; intIter < strInput.length();
intIter = intIter + 1)
     {
          intState =
arrDFA[intState].transition[lookup(strInput[intIter],st
rAlphabet)];
     }
     return arrDFA[intState].accept;
}

// QUESTION 2

void findAccessibles(state arrDFA[], int arrOutput[])
{
     int intSize = size(arrDFA);
     int intIndex = 1;
     arrOutput[0] = 0;
     for (int intIter = 1; intIter < intSize; intIter =
intIter + 1)
     {
          arrOutput[intIter] = intEmpty;
     }
     int intIter = 0;
     while (intIter < intIndex)
     {
          for (int intDest = 0; intDest <
size(arrDFA[arrOutput[intIter]].transition); intDest =
intDest + 1)
          {
               if
(!checkElement(arrDFA[arrOutput[intIter]].transition[in
tDest], arrOutput))
               {
                    arrOutput[intIndex] =
arrDFA[arrOutput[intIter]].transition[intDest];
                    intIndex = intIndex + 1;
               }
          }
          intIter = intIter + 1;
     }
     bubbleSort(arrOutput);
}

// QUESTION 3
void removeInaccessibles(state arrDFA[], int
intAlphabetSize)
{
     int intStates = size(arrDFA);
     int arrAccessibles[intMaxStates];
```

```
	findAccessibles(arrDFA, arrAccessibles);
	int intLength;
	// SORT arrAccessibles URGENTLY OTHERWISE THIS ALL
BREAKS
	intLength = size(arrAccessibles);

	for (int intIter = 0; intIter < intLength; intIter
= intIter + 1)
	{
		arrDFA[intIter] =
arrDFA[arrAccessibles[intIter]];
		for (int intSubIter = 0; intSubIter <
intAlphabetSize; intSubIter = intSubIter + 1)
		{
			arrDFA[intIter].transition[intSubIter] =
getIndex(arrDFA[intIter].transition[intSubIter],arrAcce
ssibles);
		}
		for (int intSubIter = intAlphabetSize;
intSubIter < intStates; intSubIter = intSubIter + 1)
		{
			arrDFA[intIter].transition[intSubIter] =
intEmpty;
		}

	}
	for (int intIter = intLength; intIter < intStates;
intIter = intIter + 1)
	{
		arrDFA[intIter].transition[0] = -1;
	}
}

void hopcroft(state arrDFA[], string strAlphabet) //
Might be worth rewriting with deques instead. May be
worth using checkElement too.
{
	int arrP[intMaxStates][intMaxStates];
	int arrW[intMaxStates][intMaxStates];
	int arrA[intMaxStates];
	int arrX[intMaxStates];
	int arrY[intMaxStates];
	int arrIntersect[intMaxStates];
	int arrDistinct[intMaxStates];

	int intStates = size(arrDFA);

	for (int intIter = 0; intIter < intStates; intIter
= intIter + 1)
	{
		for (int intSubIter = 0; intSubIter <
intMaxStates; intSubIter = intSubIter + 1)
		{
```

```
                arrP[intIter][intSubIter] = -1;
                arrW[intIter][intSubIter] = -1;
            }
        }
        bool boolEmpty = true;
        for (int intIter = 0; intIter < intStates; intIter
    = intIter + 1)
        {
            if (arrDFA[intIter].accept)
            {
                boolEmpty = false;
            }
        }

        for (int intIter = 0; intIter < intStates; intIter
    = intIter + 1)
        {
            if (arrDFA[intIter].accept||boolEmpty)
            {
                for (int intCount = 0; intCount <
    intMaxStates; intCount = intCount + 1)
                {
                    if (arrP[0][intCount] < 0)
                    {
                        arrP[0][intCount] = intIter;
                        arrW[0][intCount] = intIter;
                        break;
                    }
                }
            }
            else
            {
                for (int intCount = 0; intCount <
    intMaxStates; intCount = intCount + 1)
                {
                    if (arrP[1][intCount] < 0)
                    {
                        arrP[1][intCount] = intIter;
                        arrW[1][intCount] = intIter;
                        break;
                    }
                }
            }
        }

        while (arrW[0][0] != -1) // While W is not empty
    do
        {

            for (int intIter = 0; intIter < intMaxStates;
    intIter = intIter + 1)
            {
                arrA[intIter] = -1;
                arrX[intIter] = -1;
                arrY[intIter] = -1;
                arrIntersect[intIter] = -1;
                arrDistinct[intIter] = -1;
            }
            for (int intIter = intStates - 1; intIter >=
0; intIter = intIter - 1) // choose and remove a set A
from W
            {
                if (arrW[intIter][0] != -1)
                {
                    for (int intSubIter = 0; intSubIter
< intMaxStates; intSubIter = intSubIter + 1)
                    {
                        arrA[intSubIter] =
arrW[intIter][intSubIter];
                        arrW[intIter][intSubIter] = -
1;
                    }
                    break;
                }
            }

            //for (char charIter : strAlphabet) // For
each c in sigma do
            for (int intChar = 0; intChar <
strAlphabet.length(); intChar = intChar + 1)
            {
                // let X be the set of states for which
a transition on c leads to a state in A
                int intXIndex = 0;
                for (int intIter = 0; intIter <
intStates; intIter = intIter + 1)
                {
                    bool boolFind = false;
                    for (int intSubIter = 0; intSubIter
< intStates; intSubIter = intSubIter + 1)
                    {
                        if
(arrDFA[intIter].transition[intChar] ==
arrA[intSubIter])
                        {
                            boolFind = true;
                            break;
                        }
                    }
                    if (boolFind)
                    {
                        arrX[intXIndex] = intIter;
                        intXIndex = intXIndex + 1;
                    }
                }
```

```
               for (int intIter = 0; intIter <
intStates; intIter = intIter + 1) // for each set Y in
P...
               {
                    int intIIndex = 0;
                    int intDIndex = 0;
                    for (int intSubIter = 0; intSubIter
< intStates; intSubIter = intSubIter + 1)
                    {
                         arrIntersect[intSubIter] = -1;
                         arrDistinct[intSubIter] = -1;
                    }
                    bool boolFound = false;
                    for (int intSubIter = 0; intSubIter
< intStates; intSubIter = intSubIter + 1)
                    {
                         arrY[intSubIter] =
arrP[intIter][intSubIter];
                         boolFound = false;
                         for (int intTriIter = 0;
intTriIter < intStates; intTriIter = intTriIter + 1)
                         {

                              if (arrY[intSubIter] != -
1)
                              {
                                   if (arrY[intSubIter]
== arrX[intTriIter])
                                   {

     arrIntersect[intIIndex] = arrY[intSubIter];
                                        intIIndex =
intIIndex + 1;
                                        boolFound =
true;
                                        break;
                                   }
                              }
                         }
                         if (!boolFound &&
arrY[intSubIter] != -1)
                         {
                              arrDistinct[intDIndex] =
arrY[intSubIter];
                              intDIndex = intDIndex +
1;

                         }
                    }
                    if (intIIndex != 0 && intDIndex !=
0) //... for which XnY is nonempty and Y \ X is
nonempty do
```

```
                    {
                         int intCount = 0;
                         for (int intSubIter = 0;
intSubIter < intStates; intSubIter = intSubIter + 1)
                         {
                              if (arrP[intSubIter][0]
== -1)
                              {
                                   intCount =
intSubIter;
                                   break;
                              }
                         }

                         for (int intSubIter = 0;
intSubIter < intStates; intSubIter = intSubIter + 1) //
Replace Y in P by the two sets XnY and Y \ X
                         {
                              arrP[intIter][intSubIter]
= arrDistinct[intSubIter];

     arrP[intCount][intSubIter] =
arrIntersect[intSubIter];
                         }
                         for (int intSubIter = 0;
intSubIter < intStates; intSubIter = intSubIter + 1)
                         {
                              if (arrW[intSubIter][0]
== -1)
                              {
                                   intCount =
intSubIter;
                                   break;
                              }
                         }
                         int intW = -1;
                         for (int intSubIter = 0;
intSubIter < intStates; intSubIter = intSubIter + 1)
                         {
                              bool boolEqual = true;
                              for (int intTriIter = 0;
intTriIter < intStates; intTriIter = intTriIter + 1)
                              {
                                   if
(arrW[intSubIter][intTriIter] != arrY[intTriIter])
                                   {
                                        boolEqual =
false;
                                        break;
                                   }
                              }
                              if (boolEqual)
                              {
```

```
                            intW = intSubIter;
                            break;
                        }
                    }
                    if (intW >= 0) // if Y is in W
                    {
                        for (int intSubIter = 0;
intSubIter < intStates; intSubIter = intSubIter + 1) //
Replace Y in W by the same two sets
                        {

    arrW[intW][intSubIter] = arrDistinct[intSubIter];

    arrW[intCount][intSubIter] =
arrIntersect[intSubIter];
                        }
                    }
                    else // else
                    {
                        if (intDIndex >=
intIIndex) // If |XnY| <= |Y\X|
                        {
                            for (int intSubIter
= 0; intSubIter < intStates; intSubIter = intSubIter +
1) // Add XnY to W
                            {

    arrW[intCount][intSubIter] =
arrIntersect[intSubIter];
                            }
                        }
                        else // else
                        {
                            for (int intSubIter
= 0; intSubIter < intStates; intSubIter = intSubIter +
1) // Add Y\X to W
                            {

    arrW[intCount][intSubIter] =
arrDistinct[intSubIter];
                            }
                        }
                    }
                }
                for (int intSubIter = 0; intSubIter
< intStates; intSubIter = intSubIter + 1)
                {
                    arrY[intSubIter] = -1;
                }
            }
        }
    }
```

```
     int arrIndex[intMaxStates];
     for (int intIter = 0; intIter < intStates; intIter
= intIter + 1)
     {
          for (int intSubIter = 0; intSubIter <
intStates; intSubIter = intSubIter + 1)
          {
               if (arrP[intIter][intSubIter] == -1)
               {
                    break;
               }
               else
               {
                    arrIndex[arrP[intIter][intSubIter]]
= arrP[intIter][0];
               }
          }
     }
     //Iterate through arrDFA[].transition, replacing
all elements with arrIndex values.
     for (int intIter = 0; intIter < intStates; intIter
= intIter + 1)
     {
          for (int intSubIter = 0; intSubIter <
strAlphabet.length(); intSubIter = intSubIter + 1)
          {
               arrDFA[intIter].transition[intSubIter] =
arrIndex[arrDFA[intIter].transition[intSubIter]];
          }
     }
     removeInaccessibles(arrDFA,strAlphabet.length());
}

// QUESTION 4
string generateWord(int intSeed, string strAlphabet,
int intLength)
{
     string strOutput = "";
     int intRead;
     for (int intDigit = 0; intDigit < intLength;
intDigit = intDigit + 1)
     {
          intRead = intSeed%strAlphabet.length();
          strOutput = strAlphabet[intRead] + strOutput;
          intSeed = intSeed / strAlphabet.length();
     }
     return strOutput;
}

bool testFinite(state arrDFA[], string strAlphabet)
{
     int intSize = size(arrDFA);
     int intAlphabetSize = strAlphabet.length();
     for (int intLength = intSize; intLength < 2 *
intSize; intLength = intLength + 1)
     {
          for (int intWord = 0; intWord <
pow(intAlphabetSize, intLength); intWord = intWord + 1)
          {
               if (emulateDFA(arrDFA,
generateWord(intWord, strAlphabet, intLength),
strAlphabet))
               {
                    return false;
               }
          }
     }
     return true;
}
int countLanguages(int intStates, string strAlphabet,
bool boolSizes = false, int* ptrSizes = nullptr)
{


     int intLanguages = 0;
     int intAlphabetSize = strAlphabet.length();
     int intMax = pow(2, intStates)*pow(intStates,
intStates*intAlphabetSize);
     int *arrLanguages = new int[intMax];
     for (int intIter = 0; intIter < intMax; intIter =
intIter + 1)
     {
          arrLanguages[intIter] = intEmpty;
     }
     state arrDFA[intMaxStates];
     int intTemp;
     if (boolSizes)
     {
          for (int intCount = 0; intCount <
intMaxStates; intCount = intCount + 1)
          {
               *(ptrSizes + intCount) = 0;
          }
     }
     for (int intDFA = 0; intDFA < intMax; intDFA =
intDFA + 1)
     {
          buildDFAFromInt(arrDFA, intDFA,
intAlphabetSize, intStates);
          removeInaccessibles(arrDFA,intAlphabetSize);
          hopcroft(arrDFA, strAlphabet);
          if (intStates == size(arrDFA))
          {
               intTemp = dfaToInt(arrDFA,
intAlphabetSize, size(arrDFA), true);
```

```
               if (!checkElement(intTemp,
arrLanguages))
               {
                    arrLanguages[intLanguages] =
intTemp;
                    intLanguages = intLanguages + 1;
                    if (boolSizes) //  if boolSizes =
true, ptrSizes points to a matrix that stores the sizes
of each language generated
                    {
                         int intCount =
arrDFA[0].accept;
                         if (testFinite(arrDFA,
strAlphabet))
                         {
                              for (int intSize = 1;
intSize < intStates; intSize = intSize + 1)
                              {
                                   for (int intWord =
0; intWord < pow(intAlphabetSize, intSize); intWord =
intWord + 1)
                                   {
                                        //string
strWord = "";
                                        //for (int
intChar = intSize; intChar > 0; intChar = intChar - 1)
                                        //{
                                        //   strWord =
strWord + strAlphabet[(intWord / (pow(intAlphabetSize,
intChar) - 1)) % intAlphabetSize];
                                        //}
                                        string strWord
= generateWord(intWord, strAlphabet, intSize);
                                        if
(emulateDFA(arrDFA, strWord, strAlphabet))
                                        {

     //*(ptrSizes + intLanguages - 1) = *(ptrSizes +
intLanguages - 1) + 1;
                                             intCount =
intCount + 1;
                                        }
                                   }
                              }
                              *(ptrSizes + intCount) =
*(ptrSizes + intCount) + 1;
                         }
                         else
                         {
                              *(ptrSizes +
pow(intAlphabetSize, intStates) + 1) = *(ptrSizes +
pow(intAlphabetSize, intStates) + 1) + 1;
                         }
                    }
               }
          }
     }
     return intLanguages;
}



void accessibleStates(string strPath, int
intAlphabetSize)
{
     state arrDFA[intMaxStates];
     buildDFAFromFile(arrDFA, strPath, intAlphabetSize,
' ');

     //readDFA(arrDFA, "abc");

     int arrAccessibles[intMaxStates];
     findAccessibles(arrDFA, arrAccessibles);
     int intSize = size(arrAccessibles);
     cout << "Accessible states are: ";
     for (int intIter = 0; intIter < intSize; intIter =
intIter + 1)
     {
          cout << arrAccessibles[intIter]+1 << ' ';
     }
     cout << "\n\n\n";
}

void countAllAccessibles(int intSize, int
intAlphabetSize)
{
     state arrDFA[intMaxStates];
     int arrAccessibles[intMaxStates];
     int intCount = 0;
     for (int intDFA = 0; intDFA <
pow(intSize,intSize*intAlphabetSize); intDFA = intDFA +
1)
     {
          buildDFAFromInt(arrDFA, intDFA,
intAlphabetSize, intSize);
          findAccessibles(arrDFA, arrAccessibles);
          if (intSize == size(arrAccessibles))
          {
               intCount = intCount + pow(2, intSize);
          }
     }
     cout << intCount << "\n\n\n";
}

void outputTransitionTable(state arrDFA[], string
strAlphabet)
```

```
{
     int intAlphabetSize = strAlphabet.length();
     int intDFASize = size(arrDFA);
     for (int intIter = 0; intIter < intDFASize;
intIter = intIter + 1)
     {
          for (int intSubIter = 0; intSubIter <
intAlphabetSize; intSubIter = intSubIter + 1)
          {
               cout <<
arrDFA[intIter].transition[intSubIter] + 1 << ' ';
          }
          cout << arrDFA[intIter].accept << '\n';
     }
}


void applyHopcroft(string strPath, string strAlphabet)
{
     state arrDFA[intMaxStates];
     buildDFAFromFile(arrDFA, strPath,
strAlphabet.length(), ' ');
     removeInaccessibles(arrDFA,strAlphabet.length());
     hopcroft(arrDFA, strAlphabet);
     outputTransitionTable(arrDFA, strAlphabet);
     cout << "\n\n";
}

void languageSize(string strPath, string strAlphabet)
{
     state arrDFA[intMaxStates];
     buildDFAFromFile(arrDFA, strPath,
strAlphabet.length(), ' ');
     removeInaccessibles(arrDFA, strAlphabet.length());
     hopcroft(arrDFA, strAlphabet);
     if (testFinite(arrDFA, strAlphabet))
     {
          int intSize = arrDFA[0].accept;
          //int intMax = pow(strAlphabet.length(),
size(arrDFA) - 1);
               //for (int intWord = 0; intWord <
intMax; intWord = intWord + 1)
               //{
               //   intSize = intSize +
emulateDFA(arrDFA, generateWord(intWord, strAlphabet),
strAlphabet);
          //   }
          for (int intLength = 0; intLength <
size(arrDFA); intLength = intLength + 1)
          {
               for (int intWord = 0; intWord <
pow(strAlphabet.length(), intLength); intWord = intWord
+ 1) {
```

```
                    intSize = intSize +
emulateDFA(arrDFA, generateWord(intWord, strAlphabet,
intLength), strAlphabet);
               }
          }

               cout << intSize << "\n\n\n";
     }
     else
     {
          cout << "infinite\n\n\n";
     }
}


void fns(int intN, string strAlphabet)
{
     int arrOutput[intMaxStates];
     countLanguages(intN, strAlphabet, true,
&arrOutput[0]);
     for (int intS = 0; intS < pow(2, intN); intS =
intS + 1)
     {
          cout << "f(" << intN << ','<<2 <<','<< intS
<< ") = " << arrOutput[intS] << '\n';
     }
     cout << "\n\n\n";
}

void genTransitions(int intStates, string strAlphabet,
int intSize)
{


     int intLanguages = 0;
     int intAlphabetSize = strAlphabet.length();
     int intMax = pow(2, intStates)*pow(intStates,
intStates*intAlphabetSize);
     int *arrLanguages = new int[intMax];
     string arrLanguage[16];
     for (int intIter = 0; intIter < intMax; intIter =
intIter + 1)
     {
          arrLanguages[intIter] = intEmpty;
     }
     state arrDFA[intMaxStates];
     int intTemp;
     for (int intDFA = 0; intDFA < intMax; intDFA =
intDFA + 1)
     {
          buildDFAFromInt(arrDFA, intDFA,
intAlphabetSize, intStates);
          removeInaccessibles(arrDFA, intAlphabetSize);
          hopcroft(arrDFA, strAlphabet);
          if (intStates == size(arrDFA))
          {
          intTemp = dfaToInt(arrDFA, intAlphabetSize, 
size(arrDFA), true);
          if (!checkElement(intTemp, arrLanguages))
          {
               arrLanguages[intLanguages] = intTemp;
               intLanguages = intLanguages + 1;
               int intCount = arrDFA[0].accept;
               if (testFinite(arrDFA, strAlphabet))

               {

                    for (int intSize = 1; intSize < 
intStates; intSize = intSize + 1)
                    {
                         for (int intWord = 0; intWord 
< pow(intAlphabetSize, intSize); intWord = intWord + 1)
                         {
                              //string strWord = "";
                              //for (int intChar = 
intSize; intChar > 0; intChar = intChar - 1)
                              //{
                              //   strWord = strWord + 
strAlphabet[(intWord / (pow(intAlphabetSize, intChar) - 
1)) % intAlphabetSize];
                              //}
                              string strWord = 
generateWord(intWord, strAlphabet, intSize);
                              if (emulateDFA(arrDFA, 
strWord, strAlphabet))
                              {

     arrLanguage[intCount] = strWord;
                                   intCount = intCount 
+ 1;
                              }
                         }

                    }

                    if (intCount == intSize)
                    {
                         cout << "\nTransition 
Table:\n";
                         outputTransitionTable(arrDFA, 
strAlphabet);
                         cout << "Language:\n";
                         if (arrDFA[0].accept)
                         {
                              arrLanguage[0] = "empty 
word";
```

```
                         }
                         for (int intWord = 0; intWord
< intCount; intWord = intWord + 1)
                         {
                              cout <<
arrLanguage[intWord] << '\n';
                              arrLanguage[intWord] =
"";
                         }
                    }
               }
          }
          }
     }
     cout << "\n\n\n";
}



int main()
{
     string strAlphabet = "abcdefghijklmnopqrstuvwxyz";

     // To run code for a specific question, uncomment
the specific line below
     //accessibleStates("Table1.txt","Table2.txt","Tabl
e3.txt","Table4.txt"); // . Note that arguments give
paths to a transition table

     int intInput;
     string strPath;
     int intArg;
     int intArgTwo;
     string strArg;

     while (true)
     {
          cout << "Menu:\n";
          cout << "1. Determine accessible states from
DFA specified by table\n";
          cout << "2. Compute the number of (n,k)-DFAs
with no inaccessible states\n";
          cout << "3. Apply Hopcroft's table-filling
algorithm to transition table\n";
          cout << "4. Output number of (n,2)-DFA
languages\n";
          cout << "5. Compute language size from
transition table\n";
          cout << "6. Compute f(n,2,s)\n";
          cout << "7. Generate minimal (n,2) transition
tables for languages of size s\n";
          cin >> intInput;
          switch (intInput)
```

```
            {
            case 1:
                 cout << "Input path: ";
                 cin >> strPath;
                 cout << "Input alphabet size: ";
                 cin >> intArg;
                 accessibleStates(strPath, intArg);
                 break;
            case 2:
                 cout << "Input n: ";
                 cin >> intArg;
                 cout << "Input k: ";
                 cin >> intArgTwo;
                 countAllAccessibles(intArg, intArgTwo);
                 break;
            case 3:
                 cout << "Input path: ";
                 cin >> strPath;
                 cout << "Input alphabet: ";
                 cin >> strArg;
                 applyHopcroft(strPath, strArg);
                 break;
            case 4:
                 cout << "Input n: ";
                 cin >> intArg;
                 cout << countLanguages(intArg, "ab") <<
"\n\n\n";
                 break;
            case 5:
                 cout << "Input path: ";
                 cin >> strPath;
                 cout << "Input alphabet: ";
                 cin >> strArg;
                 languageSize(strPath, strArg);
                 break;
            case 6:
                 cout << "Input n: ";
                 cin >> intArg;
                 fns(intArg, "ab");
                 break;
            case 7:
                 cout << "Input n: ";
                 cin >> intArg;
                 cout << "Input s: ";
                 cin >> intArgTwo;
                 genTransitions(intArg, "ab", intArgTwo);
                 break;

            default:
                 cout << "Command not recognised. Please
enter a number above.\n\n\n";
            }
       }
```

```
        return 0;
}
```