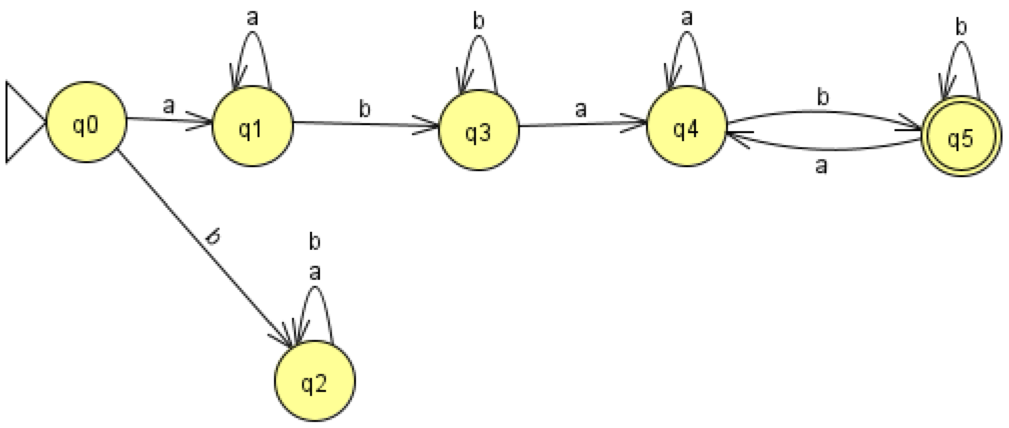
Lucas Hasting CS 421, FA25
Homework #2 December 30, 2025

A) Draw the transition graph for the DFA $L_1 = \{aw_1baw_2b : w_1, w_2 \in \{a,b\}^*\}$.



B) Minimize your DFA from part A. Show your work and the minimized transition graph.

Answer. To begin, the following notation will be used: A numerical value (0 through 5) will refer to it's corresponding state (q_0 through q_5). the pairs $(n, 5)$ where $n \in \{0,1,2,3,4\}$ are all marked as distinguishable since 5 is a final state and n is not. The table below shows the state pairs and the set of states corresponding to the input, for example the pair $(0,1)$ are the states q_0 and q_1 and the set $a = \{1\}$ is the set of states reached after the value a is consumed, which is the state q_1 for both states. The table of all pairs and inputs, not already marked as distinguishable, is shown below:

Pairs	Sets
$(0,1)$	$a=\{1\}, b=\{2,3\}$
$(0,2)$	$a=\{1,2\}, b=\{2\}$
$(0,3)$	$a=\{1,4\}, b=\{2,3\}$
$(0,4)$	$a=\{1,4\}, b=\{2,5\}$
$(1,2)$	$a=\{1,2\}, b=\{3,2\}$
$(1,3)$	$a=\{1,4\}, b=\{3\}$
$(1,4)$	$a=\{1,4\}, b=\{3,5\}$
$(2,3)$	$a=\{2,4\}, b=\{2,3\}$
$(2,4)$	$a=\{2,4\}, b=\{2,5\}$
$(3,4)$	$a=\{4,4\}, b=\{4,5\}$

Therefore, since all state pairs have at least one input with 2 states, all the states are distinguishable (and marked as distinguishable). So, the graph from part A was already minimized, and the minimized transition graph is shown in part A.

C) Implement your DFA from part *B* as a C++ program. The DFA description must be read from a text file for each program run (dfa.txt). A single input string must be read from stdin for each program run. The program will print “accept” if the string is in the language and “reject” if the string is not in the language. You may **not** use a regular expressions library. You may **not** use any string member function that operates on more than one character (substr, find, search, etc.). You may use the string length member function. Your code should implement exactly how a DFA works when it processes a string and be general enough to work with any DFA input file. Submit a print out of your source code.

main.cpp:

```
/* Name: Lucas Hasting
 * Date: 9/15/2025
 * Class: CS 421
 * Assignment: HW #2
 * Instructor: James Jerkins
 * Sources: https://cplusplus.com/reference/string/string/
 *          https://en.cppreference.com/w/cpp/container/vector.html
 *          https://www.geeksforgeeks.org/cpp/unique_ptr-in-cpp/
 *          https://www.geeksforgeeks.org/cpp/file-handling-c-classes/
 */

#include <iostream>
#include <memory>
#include "DFA.h"
using namespace std;

int main(){
    //create the DFA
    unique_ptr<DFA> automaton(new DFA("dfa.txt"));

    //get input from the user
    string input;
    cin >> input;

    //execute the automaton on the input string
    string result = automaton->execute(input);

    //display the result
    cout << result << endl;
    return 0;
}
```

DFA.h:

```
/* Name: Lucas Hasting
 * Date: 9/15/2025
 * Class: CS 421
 * Assignment: HW #2
 * Instructor: James Jerkins
 * Sources: see main.cpp
 */
```

```
#ifndef DFA_H
#define DFA_H

#include <string>
#include <vector>
using namespace std;

class DFA{
    private:
        //initial state
        int q0;

        //vector of final states
        vector<int> F;

        //vector of symbols in the alphabet
        vector<char> Sigma;

        //vector at ith index contains state transistions
        //for jth symbol
        vector<vector<int>> delta;
    public:
        //contructs DFA from filename
        DFA(string filename);

        //executes the DFA based on input string
        string execute(string input);
};

#endif
```

DFA.cpp:

```
/* Name: Lucas Hasting
 * Date: 9/15/2025
 * Class: CS 421
 * Assignment: HW #2
 * Instructor: James Jerkins
 * Sources: see main.cpp
 */

#include <fstream>
#include "DFA.h"

//constructs the DFA with a file
DFA::DFA(string filename){
    //open input file
    ifstream input(filename);
    //declare variables needed
    int states;
    int final_states;
    int symbols;

    //get the number of states
    input >> states;

    //get initial state
    input >> q0;

    //get the number of final states
    input >> final_states;

    //get all final states from file
    for(int i = 0; i < final_states; ++i){
        int f;
        input >> f;
        F.push_back(f);
    }

    //get the number of symbols
    input >> symbols;

    //get the symbols
    for(int i = 0; i < symbols; ++i){
        char s;
        input >> s;
        Sigma.push_back(s);
    }

    //get the transition function
    for(int i = 0; i < states; ++i){
        vector<int> temp_vect;
        for(int j = 0; j < symbols; ++j){
            int s;
            input >> s;
            temp_vect.push_back(s);
        }
        delta.push_back(temp_vect);
    }

    //close the input file
    input.close();
}

//method to execute the DFA on an input string
string DFA::execute(string input){
    //start at the current state
    int current_state = q0;
    //run delta (transistion) function on every input in input string
    for (int i = 0; i < input.size(); ++i){
        //transistion to next state
        for(int j = 0; j < Sigma.size(); ++j){
            if(Sigma[j] == input[i]){
                current_state = delta[current_state][j];
                break;
            }
        }
    }

    //check if the current state is a final state (accept if so,
    //reject otherwise)
    for (int i = 0; i < F.size(); ++i){
        if(current_state == F[i]){
            return "accepted";
        }
    }
    return "rejected";
}
```

dfa.txt:

```
6
0
1
5
2
a b
1 2
1 3
2 2
4 3
4 5
4 5
```