Formal Languages and Automata Theory

Programming Assignment

Last date of submission: **20–March–2021**

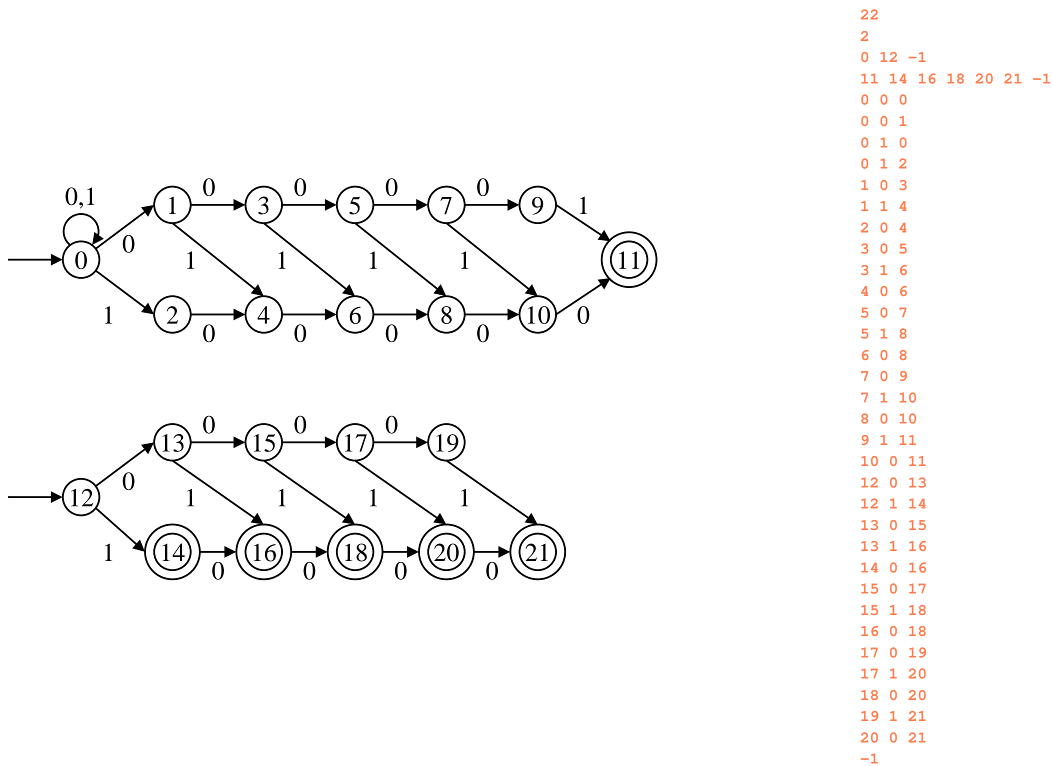
This assignment deals with the implementation of some utilities for finite automata. It restricts only to DFA and NFA, and does not deal with ε-NFA. In short, you are asked to carry out the following tasks.

- Read the data for an NFA $N = (Q_N, \Sigma, \Delta, S, F_N)$ from a file.
- Use the subset-construction procedure to convert the input NFA to an equivalent DFA $D = (Q, \Sigma, \delta, s, F)$.
- Use a graph-traversal algorithm to find out all reachable (or accessible) states in D.
- Remove all unreachable states from D to get a DFA $D' = (Q', \Sigma, \delta', s', F')$.
- Find all groups of equivalent states in D'.
- Collapse each group of equivalent states in D' to a single state, and generate the resulting minimized DFA $D'' = (Q'', \Sigma, \delta'', s'', F'')$.

Part 1: Representing DFA and NFA

In order to define N, the components $Q_N, \Sigma, \Delta, S, F_N$ need to be specified. Let us first see how this information is stored in a file. Let $n = |Q_N|$, and $m = |\Sigma|$. The file storing N starts with the two integers n and m (in that order). We take $Q_N = \{0, 1, 2, \ldots, n-1\}$, and $\Sigma = \{0, 1, 2, \ldots, m-1\}$. The start states are then listed in the file one after another. The list is terminated by -1. Subsequently, the final states are listed one after another, terminated again by -1. For the NFA of Figure 1, the start states may appear as $0 \;\; 12 \;\; -1$, whereas the final states may appear as $11 \;\; 14 \;\; 16 \;\; 18 \;\; 20 \;\; 21 \;\; -1$. It is not needed to list the start and/or the final states in the sorted order. The rest of the file lists the transitions of N. If $q \in \Delta(p, a)$, a line in the file lists $p \;\; a \;\; q$. The list of these entries terminates when $p = -1$. Again, the individual transitions may appear in any order.

Figure 1: An NFA for E_6 and its representation in a file



Write a function *readNFA*(*filename*) to read the input file, and store the information in a user-defined data structure N. Store n and m as integer fields in N. For an NFA, S, F, and all $\Delta(p, a)$ are subsets of Q_N. Assume that

$n = |Q_N| < 32$, so we can store a subset of Q_N as a bit array in a 32-bit unsigned integer variable. For example, for the NFA of Figure 1, we have the binary storage of the following subsets. The bits are numbered $0, 1, 2, 3, \ldots, 31$ from the least significant end. The i-th bit is 1 if i is in the subset; it is 0 otherwise.

$$\begin{aligned} S &= \{0, 12\} &&= 0000\ 0000\ 0000\ 0000\ 0001\ 0000\ 0000\ 0001, \\ F_N &= \{11, 14, 16, 18, 20, 21\} &&= 0000\ 0000\ 0011\ 0101\ 0100\ 1000\ 0000\ 0000, \\ \Delta(0,1) &= \{0, 2\} &&= 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0101, \\ \Delta(19,0) &= \emptyset &&= 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000. \end{aligned}$$

It follows that a single 32-bit unsigned integer variable suffices to store each of S and F_N. A two-dimensional $n \times m$ matrix of 32-bit unsigned integer variables can, on the other hand, store the entire transition table $\Delta(p, a)$.

In the remaining parts, you work with DFA, so a representation of a DFA $D = (Q, \Sigma, \delta, s, F)$ is also needed. Q and Σ can be represented as earlier by two integers $n = |Q|$ and $m = |\Sigma|$. We assume that $n < 2^{32}$, so a single 32-bit unsigned integer can store n. For D, the start state s is a unique element of Q, and can also be stored as an integer. Moreover, each $\delta(p, a)$ is again a unique member of Q, so an $n \times m$ array of integers can store the entire transition table δ. A problem comes when we want to represent a subset of Q. Unlike an NFA, a DFA may contain more than 32 states (for example, the subset-construction procedure on the NFA of Figure 1 gives a DFA with $2^{22} = 4,194,304$ states). We follow the same bit-level representation of each state in the subset, but now we need an array A of 32-bit integers. $A[0]$ stores the membership information of states $0 \ldots 31$, $A[1]$ stores the membership information of states $32 \ldots 63$, $A[2]$ stores the membership information of states $64 \ldots 95$, and so on. Define a data type for sets of states as an array of unsigned integer variables. This data type can store the set F of final states of D, and will be used later again when other state sets are used (for example, the set of reachable states of D).

Write two functions *printNFA*(N) and *printDFA*(D) to print the information stored in an NFA or a DFA data type, in the format illustrated in the sample output. If D contains too many (like > 64) states, it is preferable to skip the printing of the final states and the transition table of D.

In this assignment, you are *required* to handle your own data structures for storing sets of states. Do **not** use any set data type available in any library (like the STL). Not only the bit-level storage specified above is space-efficient but also using a general-purpose set library will likely lead to infeasible running times for Part 2 which deals with exponential numbers of states.

Part 2: NFA-to-DFA conversion by subset construction

Write a function *subsetcons(N)* that, given an NFA N as input, returns an equivalent DFA D obtained by the subset-construction procedure. Here, N and D must be in the format specified in Part 1. We have $Q = 2^{Q_N}$, that is, the states of Q are subsets of the set of states of N. With the bit-level storage of sets, the states of D have a natural one-to-one correspondence with the sets of states of N (that is, members of 2^{Q_N}). Let $P \subseteq Q_N$ be a state of D. For each $a \in \Sigma$, we have $\delta(P, a) = \bigcup_{p \in P} \Delta(p, a)$. Each $\Delta(p, a)$ for N is a set stored as a 32-bit unsigned integer. Therefore the set union in the above formula is nothing but the word-level OR of $\Delta(p, a)$ values. You nevertheless need to find the bits set in P, but that too can be carried out efficiently by bit-wise operators.

Bit-wise operators

The bit-wise OR of two 32-bit unsigned integers V and W can be computed as in the following examples.

```
U = (V | W);
W |= V;
```

Let $j \in \{0, 1, 2, \ldots, 31\}$ be a bit position. The j-th bit in W can be set as:

```
W |= (1U << j);
```

In order to check whether the j-th bit in W is set, check whether the following expression is non-zero.

```
(W & (1U << j))
```

Part 3: Finding and removing unreachable states in a DFA

Let $D = (Q, \Sigma, \delta, s, F)$ be a DFA (like the one obtained in Part 2). D can be considered as a directed graph (with possible self-loops). Run a graph-traversal algorithm (depth-first search seems to be the easiest) starting from the start state s. After the traversal completes, the *visited* array (a set of DFA states) stores the states that can be reached from s. The remaining states in Q are unreachable, and can be thrown away from Q with impunity.

Write a function *findreachable*(D) to return the set R of states of D reachable from s, and another function *rmunreachable*(D,R) to return an equivalent DFA $D' = (Q', \Sigma, \delta', s', F')$ in which all the states not in R are removed. Let $n = |Q|$, and $n' = |Q'|$ (so $n' \leqslant n$). By our convention, we represent $Q = \{0, 1, 2, \ldots, n-1\}$ and $Q' = \{0, 1, 2, \ldots, n'-1\}$. After some states are thrown out from Q, a renumbering of the remaining states is necessary for Q'. Moreover, the transition table for D should be revised to the transition table for D' in order to reflect this renumbering.

Part 4: Finding and collapsing equivalent states in a DFA

The DFA D' obtained in Part 2 may still be not minimal. In order to obtain the minimal DFA, we need to identify groups of equivalent states, and collapse each group to a single state. Implement the algorithm covered in the class in a function *findequiv*(D') to return a two-dimensional array M such that the states i and j in D' are equivalent if and only if $M[i][j]$ is not marked. Finally, write a function *collapse*(D', M) to return the state-collapsed (that is, minimal) DFA D''. Note that a renumbering of the states and a subsequent readjustment of the transition-function table are again necessary before D'' is returned.

The *main* function

- The user supplies the name of a file storing the information about the input NFA N. Call *readNFA* to read the file, and store the information in an NFA data type N. Print N.
- Call *subsetcons* on N to obtain the DFA D. Print D (skip printing very long lists).
- Call *findreachable* to obtain the list R of states reachable from s in D. Print the members of R.
- Call *rmunreachable* to get the DFA D' from D. Print D'.
- Call *findequiv* to obtain the two-dimensional array M.
- Identify the equivalent groups from M, and print the groups (this may be done in the function *collapse*).
- Call *collapse* to generate the minimal DFA D''. Print D''.

Sample output

Let us consider the following families of languages over the binary alphabet $\{0,1\}$. For each of these families, k is a positive integer-valued parameter.

$$L_k = \Big\{ x \;\Big|\; \text{The } k\text{-th last symbol of } x \text{ is } 1 \Big\},$$
$$A_k = \Big\{ x \;\Big|\; \text{The last } k \text{ symbols of } x \text{ contains at least one } 1 \Big\},$$
$$E_k = \Big\{ x \;\Big|\; \text{The last } k \text{ symbols of } x \text{ contains exactly one } 1 \Big\}.$$

All strings in L_k are of length $\geqslant k$. On the other hand, A_k and E_k are allowed to contain strings x of length $< k$ with the interpretation that x must contain at least one or exactly one 1. An NFA for E_6 is already given in Figure 1. NFA for L_6 and A_6 are supplied in Figures 2 and 3. Also, consider the following language over $\{0,1,2,3\}$. Figure 4 shows an NFA for this language, directly converted from the (unoptimized) regular expression $3^*03^*13^*23^* + 3^*03^*23^*13^* + 3^*13^*03^*23^* + 3^*13^*23^*03^* + 3^*23^*03^*13^* + 3^*23^*13^*03^*$. The sample output for E_6 is given below. Outputs for L_6, A_6, and B can be obtained from the course web-site.

$$B = \Big\{ x \in \{0,1,2,3\}^* \;\Big|\; x \text{ contains each of the symbols } 0, 1, 2 \text{ exactly once} \Big\}.$$

Figure 2: An NFA for L_6

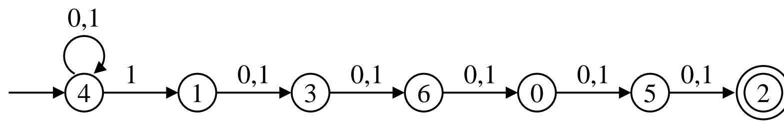


Figure 3: An NFA for A_6

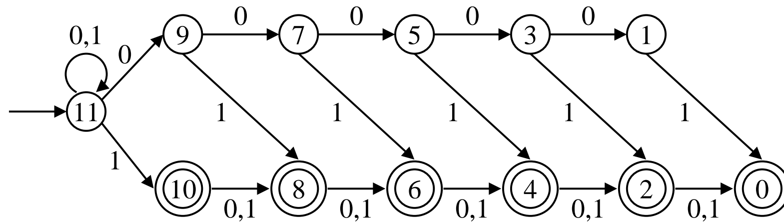


Figure 4: An NFA for B

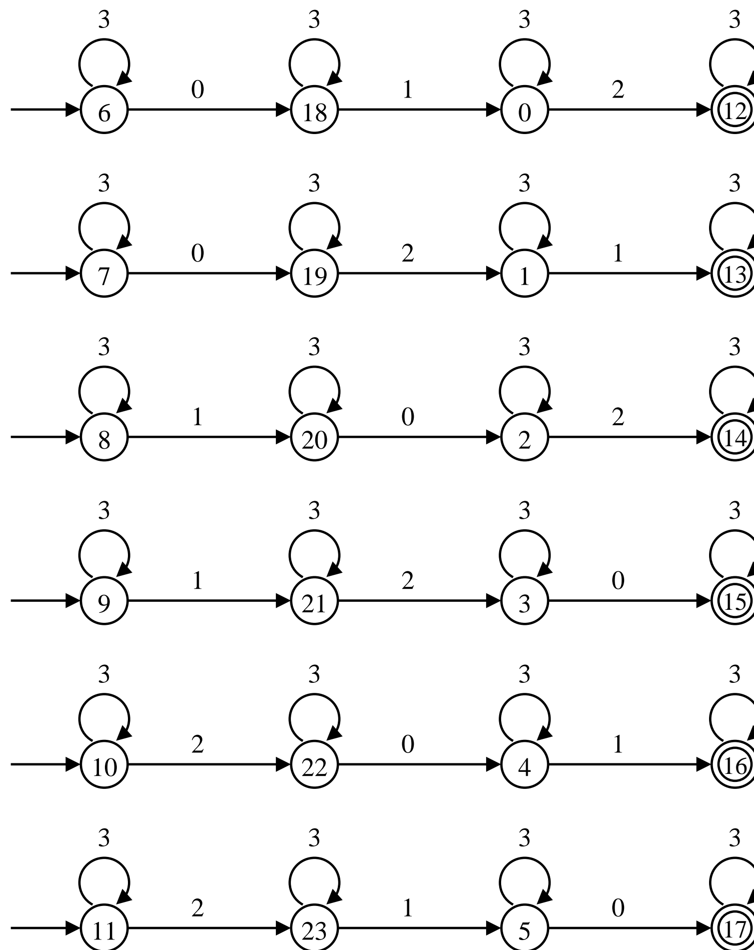
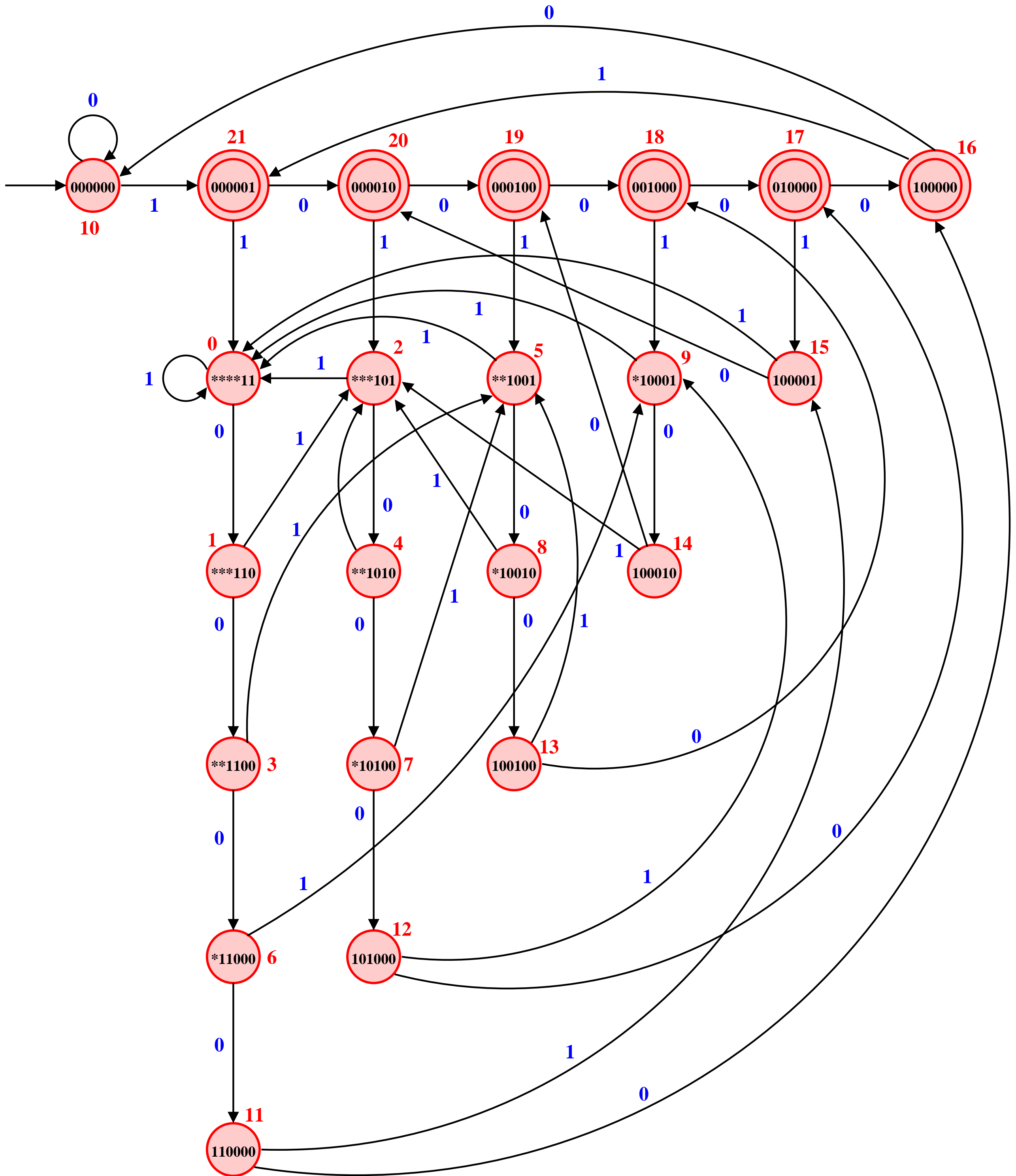


Figure 5: The minimal DFA for E_6



The minimal DFA for E_6 can be obtained from the first principles (see Figure 5). The output DFA D'' from your program should be isomorphic to this DFA. The state numbers obtained in the sample are shown beside the states in Figure 5. The logical names appear inside the circles.

```
+++ Input NFA
    Number of states: 22
    Input alphabet: {0,1}
    Start states: {0,12}
    Final states: {11,14,16,18,20,21}
    Transition function
        Delta(0,0) = {0,1}
        Delta(0,1) = {0,2}
        Delta(1,0) = {3}
        Delta(1,1) = {4}
        Delta(2,0) = {4}
        Delta(2,1) = {}
        Delta(3,0) = {5}
        Delta(3,1) = {6}
        Delta(4,0) = {6}
        Delta(4,1) = {}
        Delta(5,0) = {7}
        Delta(5,1) = {8}
        Delta(6,0) = {8}
        Delta(6,1) = {}
        Delta(7,0) = {9}
        Delta(7,1) = {10}
        Delta(8,0) = {10}
        Delta(8,1) = {}
        Delta(9,0) = {}
        Delta(9,1) = {11}
        Delta(10,0) = {11}
        Delta(10,1) = {}
        Delta(11,0) = {}
        Delta(11,1) = {}
        Delta(12,0) = {13}
        Delta(12,1) = {14}
        Delta(13,0) = {15}
        Delta(13,1) = {16}
        Delta(14,0) = {16}
        Delta(14,1) = {}
        Delta(15,0) = {17}
        Delta(15,1) = {18}
        Delta(16,0) = {18}
        Delta(16,1) = {}
        Delta(17,0) = {19}
        Delta(17,1) = {20}
        Delta(18,0) = {20}
        Delta(18,1) = {}
        Delta(19,0) = {}
        Delta(19,1) = {21}
        Delta(20,0) = {21}
        Delta(20,1) = {}
        Delta(21,0) = {}
        Delta(21,1) = {}

+++ Converted DFA
    Number of states: 4194304
    Input alphabet: {0,1}
    Start state: 4097
    4128768 final states
    Transition function: Skipped

+++ Reachable states: {5,19,21,75,83,85,299,331,339,341,683,1195,1323,1355,1363,1365,2731,3243,3371,3403,3411,3413,4097,8195,16389,
                       32779,65555,65557,131115,262219,262227,262229,524459,1048875,1048907,1048915,1048917,2098347,2098475,2098507,
                       2098515,2098517}

+++ Reduced DFA after removing unreachable states
    Number of states: 42
    Input alphabet: {0,1}
    Start state: 22
    Final states: {16,17,18,19,20,21,24,26,27,29,30,31,33,34,35,36,37,38,39,40,41}
    Transition function
    a\p|  0  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 30 31 32 33 34 35 36 37 38 39 40 41
    ---+--------------------------------------------------------------------------------------------------------------------------------
     0 |  1  3  4  6  7  8 11 12 13 14 10 16 17 18 19 20 10 16 17 18 19 20 23 25 26 28 29 30 32 33 34 35 10 37 38 39 40 16 17 18 19 20
     1 |  0  2  0  5  2  0  9  5  2  0 21 15  9  5  2  0 21 15  9  5  2  0 24 27  0 31  2  0 36  5  2  0 41  9  5  2  0 15  9  5  2  0

+++ Equivalent states
    Group  0: {0}
    Group  1: {1}
    Group  2: {2}
    Group  3: {3}
    Group  4: {4}
    Group  5: {5}
    Group  6: {6}
    Group  7: {7}
    Group  8: {8}
    Group  9: {9}
    Group 10: {10,22,23,25,28,32}
    Group 11: {11}
    Group 12: {12}
    Group 13: {13}
    Group 14: {14}
    Group 15: {15}
    Group 16: {16}
    Group 17: {17,37}
    Group 18: {18,33,38}
    Group 19: {19,29,34,39}
    Group 20: {20,26,30,35,40}
    Group 21: {21,24,27,31,36,41}
```

```
+++ Reduced DFA after collapsing equivalent states
    Number of states: 22
    Input alphabet: {0,1}
    Start state: 10
    Final states: {16,17,18,19,20,21}
    Transition function
    a\p|  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
    ---+------------------------------------------------------------------
     0 |  1  3  4  6  7  8 11 12 13 14 10 16 17 18 19 20 10 16 17 18 19 20
     1 |  0  2  0  5  2  0  9  5  2  0 21 15  9  5  2  0 21 15  9  5  2  0
```

Remark: Parts 3 and 4 renumber the states. If you want your new numbers to match the sample outputs exactly, note the renumbering schemes used here. In Part 3 (remove inaccessible states), the reachable states are numbered sequentially in the sorted order of their old numbers. In Part 4, the groups are numbered sequentially in the sorted order of the smallest states in the groups. It is, of course, assumed that you follow the subset-numbering scheme specified in Part 1. If you use different numbering schemes, you would get isomorphic DFA, and these isomorphisms can be manually verified for correctness.

Submit a single C/C++ file. Do not use global/static variables. Do not use STL data types.