#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int \*\*transitionMap; // 2D array which is used to store state transitions. transitionMap[i][j] is the state reached when state i is given symbol j

int \*\*partitionTransitionMap; // same as transitionMap, except row indices represent partition numbers, not state numbers

int startState; // The starting state. This is used as the root for DFS to eliminate unreachable states

long int reachable; // A bitset to represent states that are reachable

long int allStates; // A bitset to represent all states in the FSM

long int finalStates; // A bitset to represent final states in the FSM

long int nonFinalStates; // A bitset to represent non-final states in the FSM

long int \*P; // array of partitions. Each partition is a bitset of states

void dfs(int v)

{

reachable |= (1 << v);

int i;

// Try exploring all paths..

for(i=0; i<26; i++)

if((transitionMap[v][i] != -1) && ((reachable & (1 << transitionMap[v][i])) == 0))

{

dfs(transitionMap[v][i]);

}

}

int main(){

// We start off with no states

finalStates = 0;

allStates = 0;

int i, j, k;

// Initialize our transition maps. We set transition[i][j] to be -1 in order to indicate that state/partition i does not transition when given symbol j

transitionMap = (int\*\*)malloc(64\*sizeof(int\*));

for (i = 0; i < 64; i++){

transitionMap[i] = (int\*) malloc(26\*sizeof(int));

for (j = 0; j < 26; j++){

transitionMap[i][j] = -1;

}

}

partitionTransitionMap = (int\*\*)malloc(64\*sizeof(int\*));

for (i = 0; i < 64; i++){

partitionTransitionMap[i] = (int\*) malloc(26\*sizeof(int));

for (j = 0; j < 26; j++){

partitionTransitionMap[i][j] = -1;

}

}

// read start state

char buff[125];

printf("Enter the start state\n");

fgets(buff, sizeof(buff), stdin);

char \*p = strtok(buff, " ");

startState = atoi(p);

// read final states

printf("Enter the final state(s)\n");

fgets(buff, sizeof(buff), stdin);

p = strtok(buff, " ");

while (p != NULL)

{

int state = atoi(p);

finalStates |= 1 << (state);

p = strtok(NULL, " ");

}

// read transitions

int from;

char symbol;

int to;

printf("Enter the transitions one by one in the form state symbol state.\n Press Ctrl+D when finished\n");

while (fscanf(stdin, "%d %c %d", &from, &symbol, &to) != EOF) {

transitionMap[from][symbol-'a'] = to; // add transition

// add from and to states to the allStates bitset

allStates |= (1 << from);

allStates |= (1 << to);

}

// initialize reachable bitset to 0 and run dfs to determine reachable states

reachable = 0;

dfs(startState);

// filter unreachable states

allStates &= reachable;

finalStates &= reachable;

// initialize array of partitions to include empty bitsets

P = (long int\*) malloc(64\*sizeof(long int));

for (i = 0; i < 64 ; i++){

P[i] = 0; // no partition exists

}

// P should include two partitions to start: final states and non-final states

nonFinalStates = allStates & ~finalStates;

P[0] = finalStates;

P[1] = nonFinalStates;

int nextPartitionIndex = 2; // Store how many partitions have been added already

// There will be at most 64 partitions. At each iteration, we operate on a partition and add at most 1 more partition

for (i = 0; i < 64; i++){

// A bitset for a new partition. This partition will include all states that are distinct from the state corresponding to the leftmost bit in P[i]

long int newPartition = 0;

// Done partitioning

if (P[i] == 0){

break;

}

// Try to find leftmost bit in the bitset. This loop will only run to its entirety once when that bit is found

for (j = 63; j >= 0; j--) {

// Potential leftmost bit. If found, this bit will remain in the bit set.

long int staticState = (long int) 1 << j;

// Check if this state is in the current bitset

if ((P[i] & (staticState)) != 0){

// The lestmost bit state will be associated with this partition. Therefore, we must copy over the transitions for this state to the transitions for

// the corresponding partition

partitionTransitionMap[i] = transitionMap[j];

// Check for states that should be removed from this partition. All states will be bits right of the staticState bit

int k;

for (k = j - 1; k >= 0; k -- ){

// Potential state to remove

long int otherState = (long int) 1 << k;

// Check if this state is in the current bitset

if ((P[i] & (otherState)) != 0){

// Iterate across the entire alphabet and check if staticState and otherState can transition to different partitions.

int l;

for (l = 0; l < 26; l++){

int staticNext = -1; // next partition for static

int otherNext = -1; // next partition for other

int m;

for (m = 0; m < nextPartitionIndex; m++){

if ((P[m] & (1 << transitionMap[j][l])) != 0){

staticNext = m; //found static next

}

if ((P[m] & (1 << transitionMap[k][l])) != 0){

otherNext = m; // found other next

}

}

// If partitions differ, remove the other state and add it to the new partition. Then break, since we are done with this partition

if (transitionMap[j][l] != transitionMap[k][l] && (staticNext != otherNext)){

P[i] &= ~(1 << k);

newPartition |= (1 << k);

break;

}

}

}

}

break;

}

}

// New partition exists. Add it to P and increment nextPartitionIndex

if (newPartition != 0){

P[nextPartitionIndex] = newPartition;

nextPartitionIndex++;

}

}

// find and print start partition

int startPartition = 0;

for (i = 0; i < nextPartitionIndex; i ++){

if ((P[i] & (1 << startState)) != 0 ){

startPartition = i;

break;

}

}

printf("The new start state is:\n");

printf("%d \n", startPartition);

// find and print final partitions

printf("The new final state(s) is\\are:\n");

for (i = 0; i < nextPartitionIndex; i++){

if ((P[i] & finalStates) != 0){

printf("%d ", i);

}

}

printf("\n");

// find and print all transitions

printf("The new transitions are:\n");

for (i = 0; i < nextPartitionIndex; i++){

for (j = 0; j < 26; j++) {

if (partitionTransitionMap[i][j] != -1){

for (k = 0; k < nextPartitionIndex; k++){

if ((P[k] & (1 << partitionTransitionMap[i][j])) != 0){

printf("%d %c %d\n", i, j + 'a', k);

}

}

}

}

}

return 0;

}