

Assignment 2

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


int **transitionMap;

int **partitionTransitionMap;

int startState;

long int reachable;

long int allStates;

long int finalStates;

long int nonFinalStates;

long int *P;


void dfs(int v)
{
    reachable |= (1 << v);
    for(int i=0; i<26; i++)
    {
        if((transitionMap[v][i] != -1) && ((reachable & (1 << transitionMap[v][i])) == 0))
            dfs(transitionMap[v][i]);
    }
}
```

```

int main()
{
    finalStates = 0;
    allStates = 0;
    transitionMap = (int**)malloc(64*sizeof(int*));
    for (int i = 0; i < 64; i++)
    {
        transitionMap[i] = (int*) malloc(26*sizeof(int));
        for (int j = 0; j < 26; j++)
            transitionMap[i][j] = -1;
    }

    partitionTransitionMap = (int**)malloc(64*sizeof(int*));
    for (int i = 0; i < 64; i++)
    {
        partitionTransitionMap[i] = (int*) malloc(26*sizeof(int));
        for (int j = 0; j < 26; j++)
            partitionTransitionMap[i][j] = -1;
    }

    char buff[125];
    fgets(buff, sizeof(buff), stdin);
    char *p = strtok(buff, " ");
    startState = atoi(p);

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

```

```

p = strtok(buff, " ");
while (p != NULL)
{
    int state = atoi(p);
    finalStates |= 1 << (state);
    p = strtok(NULL, " ");
}

int from;
char symbol;
int to;
while (fscanf(stdin, "%d %c %d", &from, &symbol, &to) != EOF)
{
    transitionMap[from][symbol-'a'] = to;
    allStates |= (1 << from);
    allStates |= (1 << to);
}

reachable = 0;
dfs(startState);
allStates &= reachable;
finalStates &= reachable;
P = (long int*) malloc(64*sizeof(long int));
for (int i = 0; i < 64 ; i++)
    P[i] = 0;

```

```

nonFinalStates = allStates & ~finalStates;

P[0] = finalStates;
P[1] = nonFinalStates;

int nextPartitionIndex = 2;


for (int i = 0; i < 64; i++)
{
    long int newPartition = 0;
    if (P[i] == 0)
        break;
    for (int j = 63; j >= 0; j--)
    {
        long int staticState = (long int) 1 << j;
        if ((P[i] & (staticState)) != 0)
        {
            partitionTransitionMap[i] = transitionMap[j];
            for (int k = j - 1; k >= 0; k -- )
            {
                long int otherState = (long int) 1 << k;
                if ((P[i] & (otherState)) != 0){
                    for (int l = 0; l < 26; l++)
                    {
                        int staticNext = -1;
                        int otherNext = -1;
                        for (int m = 0; m < nextPartitionIndex; m++)
                        {

```

```

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

        staticNext = m;

    if ((P[m] & (1 << transitionMap[k][l]))

        otherNext = m;

    }

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

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

    }

}

}

break;

}

}

if (newPartition != 0)
{

    P[nextPartitionIndex] = newPartition;
    nextPartitionIndex++;

}

}

```

```

int startPartition = 0;
for (int i = 0; i < nextPartitionIndex; i++){
    if ((P[i] & (1 << startState)) != 0 ){
        startPartition = i;
        break;
    }
}

printf("%d \n", startPartition);
for (int i = 0; i < nextPartitionIndex; i++)
{
    if ((P[i] & finalStates) != 0)
        printf("%d ", i);
}
printf("\n");

for (int i = 0; i < nextPartitionIndex; i++)
{
    for (int j = 0; j < 26; j++)
    {
        if (partitionTransitionMap[i][j] != -1)
        {
            for (int k = 0; k < nextPartitionIndex; k++)
            {
                if ((P[k] & (1 << partitionTransitionMap[i][j])) != 0)
                    printf("%d %c %d\n", i, j + 'a', k);
            }
        }
    }
}

```

```
        }  
    }  
}  
}  
return 0;  
}
```