Tema 1

1. Implementați un automat finit determinist (DFA).

Programul citește din fișierul “input.txt” următoarele informații:

- pe prima linie, numărul de stări ale automatului (numit N)
- pe a doua linie, stările automatului (numere întregi, nu neapărat consecutive)
- pe linia a treia, numărul de tranziții (să îl notam M)
- pe următoarele M linii, descrierea unei tranziții, sub formatul x y l, unde:
  - x este starea (nodul) din care pleacă tranziția (muchia, arcul)
  - y este starea in care ajunge tranziția
  - l este litera tranziției
- pe următoarea linie este scris un număr natural S, care înseamnă starea inițială
- pe următoarea linie găsim nrF, care înseamnă numărul de stări finale
- pe următoarea linie găsim nrF numere întregi (nrF <= N), reprezentând stările finale
- pe următoarea linie găsim NrCuv, care înseamnă numărul de cuvinte ce urmează a fi verificate
- pe următoarele NrCuv linii se găsește cate un cuvânt

Programul trebuie sa afișeze în fișierul “output.txt”, pe nrCuv linii separate, unul din cuvintele DA (+ reconstituire drum) sau NU, semnificând dacă cuvântul respectiv este sau nu acceptat de automat.

Observație: alfabetul limbajului descris de automat va fi format doar din litere mici ale alfabetului englez

```
# Implementarea unui DFA (AFD)

f = open("input.txt")
g = open("output.txt", "w")


# Citim numarul de stari ale automatului.
nrstari = int(f.readline())


# Citim starile automatului.
stari = [int(x) for x in f.readline().split()]


# Salvam starea maxima pentru a cunoaste cat spatiu alocam matricei de
adiacenta.
dimensiune_mat = max(stari)


# Citim numarul de tranzitii ale automatului.
tranzitii = int(f.readline())


# Citim tranzitiile automatului, formand o lista de tupluri:
# (starea de unde plecam, starea in care ajungem, litera tranzitiei).
```

```
muchii = []
i = 0
while i < tranzitii:
    x, y, val = [x for x in f.readline().split()]
    muchii.append((int(x), int(y), val))
    i += 1


# Citim starea initiala.
stare_initiala = int(f.readline())


# Citim numarul de stari finale.
nrfinale = int(f.readline())


# Citim starile finale si le salvam intr-o lista.
stari_finale = [int(x) for x in f.readline().split()]


# Citim numarul de cuvinte pe care vrem sa le verificam.
nrcuv = int(f.readline())


# Citim cuvintele si le adaugam intr-o lista.
i = 0
cuvinte = []
while i < nrcuv:
    cuvinte.append(f.readline().strip())
    i += 1


# Initializam o matrice de liste nule. Am initializat matricea astfel
deoarece
# intr-o stare se pot reintoarce mai multe valori si trebuie sa le retinem pe
toate.
mat = [[[] for x in range(dimensiune_mat+1)] for y in
range(dimensiune_mat+1)]


# Adaugam valori in matrice astfel:
# mat[stare_de_unde_plecam][stare_unde_ajungem] = [lista cu valorile pe care
le putem avea]
for muchie in muchii:
    mat[muchie[0]][muchie[1]].append(muchie[2])


# Initializam o lista vida pentru a afisa si drumul parcurs pentru
verificarea cuvantului.
drum = []

# Daca avem cuvantul vid, iar starea initiala este si stare finala, inseamna
ca este acceptat.
# Daca starea initiala nu este si finala, cuvantul vid nu este acceptat.
# Daca cuvantul nu este vid, pentru fiecare cuvant setam nodul curent starea
initiala,
# adaugandu-l la drum si il salvam intr-o variabila auxiliara din care vom
```

```
elimina pas cu pas litera parcursa.
# Astfel putem verifica daca s-a parcurs tot cuvantul, evitand situatia in
care nodul curent ramane blocat
# la o stare finala, avand ulterior litere ce nu corespund unei tranzitii
bune.
# Pentru fiecare litera in cuvant, pentru fiecare stare viitoare, verificam
daca gasim litera
# corespunzatoare intr-o tranzitie.
# Daca o gasim, actualizam nodul curent cu cel gasit si il adaugam la drum,
apoi oprim cautarea
# si trecem la urmatoarea litera.
# Dupa ce am verificat toate literele, daca nodul curent se afla in starile
finale,
# cuvantul este acceptat si putem afisa drumul, altfel nu.
for cuvant in cuvinte:
    if len(cuvant) == 0:
        if stare_initiala in stari_finale:
            g.write("Da -> " + str(stare_initiala) + "\n")
        else:
            g.write("Nu\n")
    else:
        nod_curent = stare_initiala
        index = 0
        salv = cuvant
        drum.append(nod_curent)
        while index < len(cuvant):
            for j in range(dimensiune_mat+1):
                if cuvant[index] in mat[nod_curent][j]:
                    nod_curent = j
                    salv = salv[1:]
                    drum.append(nod_curent)
                    break
            index += 1
        if nod_curent in stari_finale and len(salv) == 0:
            g.write("Da -> ")
            for x in drum:
                g.write(str(x) + " ")
            g.write("\n")
            drum = []
        else:
            g.write("Nu\n")
            drum = []


f.close()
g.close()
```

Exemplu de rulare:

Fișier de input:

```
4
1 2 3 4
8
1 2 a
```

```
2 2 b
2 2 a
1 3 b
3 3 a
3 4 b
4 4 a
4 4 b
1
1
4
7
abbbba
baaab

aaaaab
baba
babaaa
ba
```

Fișier de output:

```
Nu
Da -> 1 3 3 3 3 4
Nu
Nu
Da -> 1 3 3 4 4
Da -> 1 3 3 4 4 4 4
Nu
```

2. Implementați un automat finit nedeterminist (NFA).

Programul citește din fișierul “input2.txt” următoarele informații:

- pe prima linie, numărul de stări ale automatului (numit N)
- pe a doua linie, stările automatului (numere întregi, nu neapărat consecutive)
- pe linia a treia, numărul de tranziții (să îl notam M)
- pe următoarele M linii, descrierea unei tranziții, sub formatul x y l, unde:
    - x este starea (nodul) din care pleacă tranziția (muchia, arcul)
    - y este starea in care ajunge tranziția
    - l este litera tranziției
- pe următoarea linie este scris un număr natural S, care înseamnă starea inițială
- pe următoarea linie găsim nrF, care înseamnă numărul de stări finale
- pe următoarea linie găsim nrF numere întregi (nrF <= N), reprezentând stările finale
- pe următoarea linie găsim NrCuv, care înseamnă numărul de cuvinte ce urmează a fi verificate
- pe următoarele NrCuv linii se găsește cate un cuvânt

Programul trebuie sa afișeze în fișierul “output2.txt”, pe nrCuv linii separate, unul din cuvintele DA sau NU, semnificând dacă cuvântul respectiv este sau nu acceptat de automat.

Observație: alfabetul limbajului descris de automat va fi format doar din litere mici ale alfabetului englez, iar automatul citit nu este neapărat complet.

```
# Implementarea unui NFA (AFN)

f = open("input2.txt")
g = open("output2.txt", "w")


# Citim numarul de stari ale automatului.
nrstari = int(f.readline())


# Citim starile automatului.
stari = [int(x) for x in f.readline().split()]


# Salvam starea maxima pentru a cunoaste cat spatiu alocam matricei de
adiacenta.
dimensiune_mat = max(stari)


# Citim numarul de tranzitii ale automatului.
tranzitii = int(f.readline())


# Citim tranzitiile automatului, formand o lista de tupluri:
# (starea de unde plecam, starea in care ajungem, litera tranzitiei).
muchii = []
i = 0
while i < tranzitii:
    x, y, val = [x for x in f.readline().split()]
    muchii.append((int(x), int(y), val))
    i += 1


# Citim starea initiala.
stare_initiala = int(f.readline())


# Citim numarul de stari finale.
nrfinale = int(f.readline())


# Citim starile finale si le salvam intr-o lista.
stari_finale = [int(x) for x in f.readline().split()]


# Citim numarul de cuvinte pe care vrem sa le verificam.
nrcuv = int(f.readline())


# Citim cuvintele si le adaugam intr-o lista.
i = 0
cuvinte = []
```

```
while i < nrcuv:
    cuvinte.append(f.readline().strip())
    i += 1

# Initializam o matrice de liste nule. Am initializat matricea astfel
deoarece
# intr-o stare se pot reintoarce mai multe valori si trebuie sa le retinem pe
toate.
mat = [[[] for x in range(dimensiune_mat+1)] for y in
range(dimensiune_mat+1)]


# Adaugam valori in matrice astfel:
# mat[stare_de_unde_plecam][stare_unde_ajungem] = [lista cu valorile pe care
le putem avea]
for muchie in muchii:
    mat[muchie[0]][muchie[1]].append(muchie[2])

# Salvam cuvantul initial intr-o variabila auxiliara.
# Pornind de la starea initiala, pentru fiecare stare a automatului
# verificam daca exista o tranzitie corespunzatoare
# pentru prima litera a cuvantului.
# Daca nu exista, functia ia sfarsit repede dupa parcurgere,
# ok-ul nu se modifica, deci cuvantul nu e acceptat.
# Daca da, bifam prima litera si o stergem, actualizand astfel
# cuvantul pe care o sa il parcurgem cu aceeasi functie.
# Daca am parcurs tot cuvantul si ajungem intr-o stare finala, cuvantul este
# acceptat, deci modificam ok-ul si iesim din functie, verificarea luand
sfarsit.
# Daca nu am parcurs tot cuvantul, reapelam pentru starea in care am gasit
legatura si cuvantul curent.
# Daca am luat-o pe o tranzitie gresita si intr-un final functia nu s-a oprit
(nu a acceptat cuvantul),
# ne intoarcem la cuvantul anterior folosind salvarea de la inceputul
functiei
# si cautam stari in continuare, reluand algoritmul.


def dfs(nod_curent, cuvant_curent):
    global ok
    aux = cuvant_curent
    for j in stari:
        if aux[0] in mat[nod_curent][j]:
            aux = aux[1:]
            if len(aux) == 0:
                if j in stari_finale:
                    ok = True
                break
            dfs(j, aux)
            aux = cuvant_curent


# Daca avem cuvantul vid, iar starea initiala este si stare finala, inseamna
ca este acceptat.
# Daca starea initiala nu este si finala, cuvantul vid nu este acceptat.
# Daca cuvantul nu este vid, pentru fiecare cuvant apelam functia de
verificare.
```

```
for cuvant in cuvinte:
    if len(cuvant) == 0:
        if stare_initiala in stari_finale:
            g.write("Da -> " + str(stare_initiala) + "\n")
        else:
            g.write("Nu\n")
    else:
        # Presupunem cuvantul ca fiind neacceptat.
        ok = False
        dfs(stare_initiala, cuvant)
        # Daca ok nu a fost modificat in functie (daca nu exista nici un drum
pana la o stare
        # finala pentru a verifica cuvantul), cuvantul nu este acceptat,
altfel este acceptat.
        if ok is False:
            g.write("Nu\n")
        else:
            g.write("Da\n")
```

Exemplu de rulare:

Fișier de input:

```
4
1 2 3 4
5
1 1 a
1 1 b
1 2 a
2 3 a
3 4 a
1
1
4
4
abab
abaa
aaaa
abbb
```

Fișier de output:

```
Nu
Nu
Da
Nu
```

Student: Enescu Irina Stefania

Grupa 133