Nume: Vlad Theia-Madalin

Grupa: 324CC

Tema 2

Problema 1

In esenta, problema 1 ne cere sa aflam daca graful H contine un subgraph izomorf cu graful G.

a)Algoritm determinist

```
subgraphMatching(G, H) // G = (V_G, E_G), H = (V_H, E_H), n = card(V_H), m = card(V_G)
    arr[0] = 0 //vector generarea a combinari de n luate cate m seturi de noduri
    for_each v ∈ V_G // O(m)
        deg_G[v] = 0
        for_each w ∈ V_G // O(m)
            if v != w && (v, w) ∈ E_G
                deg_G[v]++
    sort(deg_G) // O(m^2)
    if check(1, n, m) == 1 //O(n * m^2)
        return 1
    return 0
```

```
check(p, n, m) //avem nevoie de p pentru backtracking
    for k = arr[p - 1] + 1…n // O(n)
        arr[p] = k
        if p == m
            V = Ø // multime de noduri, va contine m noduri din multimea V_H
            for i = 1…p //O(m)
                V = V U { i }
            for_each v ∈ V //O(m)
                deg_V[v] = 0 // vector de grade pentru nodurile din multimea V
                for_each w ∈ V // O(m)
                    if v != w && (v, w) ∈ E_G
                        deg_V[v]++
            sort(deg_V) // O(m^2)
            if deg_G == deg_V // daca doua grafuri au vectorii de grade sortati identici, sunt izomorfe
                return 1
        check(p + 1, n, m)
    return 0

sort(deg)
    for i = 0…n-2 // n = nr de elemente ale vectorului  // O(n)
        for j = i+1…n-1 // O(n)
            if deg[ i ] < deg[ j ]
                aux = deg[ i ]
                deg[ i ] = deg[ j ]
                deg[ j ] = aux
```

b)Algoritm nedeterminist

```
subgraphMatching(G, H) // G = (V_G, E_G), H = (V_H, E_H)
    arr[0] = 0 //vector generarea a combinari de n luate cate m seturi de noduri
    for_each v ∈ V_G
        deg_G[v] = 0
        for_each w ∈ V_G
            if v != w && (v, w) ∈ E_G
                deg_G[v]++
    sort(deg_G)
    V = ∅ // multime de noduri, va contine m noduri din multimea V_H
    for i = 1…p
        v = choice(V_H) // genereaza submultimi de k noduri ale multimii V_H
        V = V U { v } // nu mai este nevoie de backtracking
    for_each v ∈ V
        deg_V[v] = 0 // vector de grade pentru nodurile din multimea V
        for_each w ∈ V
            if v != w && (v, w) ∈ E_G
                deg_V[v]++
    sort(deg_V)
    if deg_G == deg_V // daca doua grafuri au vectorii de grade sortati identici, sunt izomorfe
        success
    fail
```

c) Complexitatea temporala a algoritmilor

d) Demonstratie problema subgrafului izomorf este NP_completa

Numim Q problema subgrafului izomorf.

Q este monomorfism de la H la G => Q ∈ NP (1)

Problema K_Clica este un caz special al problemei Q, dar problema K_Clica este NPC => Q este NP_dura (2)

Din (1) si (2) => Q este NPC

Problema 2

a) Algoritm determinist

```
partition(list, n) // vectorul sortat de bancnote, numarul de elemente din vector
    A, B = 0 // sumele de bani pe care le vor primi hotii
    for i=1...n
        if A <= B
            A = A + list[ i ]
        else
            B = B + list[ i ]
    if A == B // din moment ce avem o problema de decizie, nu ne intereseaza modul in
                //care sunt impartiti banii, ci doar posibilitatea de a fi impartiti in mod egal
        return 1
    return 0
```

b) Algoritm nedeterminist

```
partition(list, n)
    A, B = 0
    i = choice(1...n)
        if A <= B
            A = A + list[ i ]
        else
            B = B + list[ i ]
    if A == B
        success
    fail
```

c) Numim Q problema impartirii banilor intre hoti. Cum problema poate fi rezolvata cu un algoritm nedeterminist tractabil => Q $\in$ NP (1)

Stim ca problema Pizza, data ca hint, este din NPC. Aceasta este un caz general al problemei Q, in care jumatate din suma ce trebuie impartita intre hoti este egala cu pretul pizzei. Astfel, aceasta esta este reductibila in timp polinomial => Q este NP_dura (2)

Din (1) si (2) => Q este NPC => un programator nu poate implementa un algoritm determinist de complexitate polinomiala care sa resolve problema intr-un timp rezonabil

d) plata(list, n, price)

```
bancnote[n]
j = 0
sum = 0
while (i < n || sum != price) //O(n)
    if (sum + list[ i ] < price)
        sum += list[ i ]
        bancnote[ j ] = list[ i ]
        j++
    else
        i++
return bancnote
```

e) Complexitatea temporala a algoritmilor

Problema 3

```
int least(int max, int m[][])
    min = INT_MAX, least = INT_MAX
    for i = 0 .. n -1
        if m[x][i] != 0 && visited[i] != 0
            if m[x][i] + m[i][x] < min
                min = m[i][0] + m[x][i]
                kmin  = m[x][i]
                least = i
    if min != INT_MAX
        cost += kmin
    return least

void minCost(int x, int m[i][i])
    y = least(x, m)
    visited[x] = 1
    cities[pos] = x + 1
    pos++
    if y == INT_MAX
        y = 0
        cost += m[x][y]
        cities[pos] = x + 1
        pos++
        return
    minCost(y, m)
```

Demonstratie:

Un ciclu Hamiltonian se obtine folosind algoritmul lui Prim. Diferenta dintre TSP si Prim este ca TSP se uita doar la vecinii nodului curent si il alege pe cel cu costul cel mai mic, dar urmand aceasta cale este posibil sa nu obtina costul total minim, in timp ce Prim ia minimul din orice nod. Astfel, costul ciclului format se poate mari de cel mult 2*costul celui mai scurt ciclu Hamiltonian.

Problema 4

a)Explicare cod:

Vom forma un arbore binar, pe masura ce citim din fisierul input.in. Totodata, vom obtine un vector de varibile impreuna cu numarul acestora (sunt necesare pentru a genera multimile de intrari).

Vom converti arborele astfel incat sa contina forma CNF a formulei din fisierul de intrare. In functia de convertire luam in considerare toate cazurile posibile. Daca intalnim o frunza sau un “si”, nu le schimbam. Daca intalnim un “sau”, executam operatiile in functie de cazul intalnit. Daca intalnim o operatorul de negatie tratam cazul in care negam un “si”, un “sau”, o variabila sau avem o dubla negatie.

Dupa ce am ajuns la CNF, generam seturi de intrari (in functie de numarul de variabile), si le testam pana gasim unul care rezolva circuitul. Daca niciun set nu a rezolvat circuitul, afisam mesajul “IMPOSIBIL”.