Univerzitet u Sarajevu
Elektrotehnički fakultet
Predmet: Optimizacija resursa
Akademska godina: 2020/2021
Student: Samra Mujčinović
Datum: 10.1.2021.

Izvještaj sa laboratorijske vježbe 9

Zadatak 1

Ideja za rješenje zadatka

U ovoj laboratorijskoj vježbi je bilo potrebno implementirati algoritam tabu pretraživanja za rješavanje problema trgovačkog putnika. U ovu svrhu korišten je prethodno implementiran algoritam tabu pretraživanja sa određenim izmjenama. Problem trgovačkog putnika (TSP) je pronaći najkraću, a najučinkovitiju rutu za kretanje osobe s obzirom na popis određenih odredišta. To je poznati algoritamski problem u području računarstva i operacionih istraživanja. Očito je da postoji mnogo različitih ruta za odabir, ali pronalaženje najboljeg - onog za koji će trebati najmanje udaljenosti ili troškova - ono je za što su matematičari i informatičari decenijama pokušavali riješiti. Problem se može riješiti analizom svake povratne rute kako bi se utvrdila najkraća. Međutim, kako se broj odredišta povećava, odgovarajući broj povratnih putovanja premašuje mogućnosti čak i najbržih računala. S 10 odredišta može biti više od 300 000 permutacija i kombinacija povratnih putovanja. S 15 odredišta, broj mogućih ruta mogao bi premašiti 87 milijardi. Problem pronalaženja početne permuntacije u ovoj vježbi smo riješili korištenjem metoda najbližeg susjeda. Ovo je možda najjednostavnija TSP heuristika. Ključ ove metode je uvijek posjetiti najbliže odredište, a zatim se vratiti u prvi grad kada se posjete svi drugi gradovi. Da bismo riješili TSP ovom metodom, biramo slučajni grad, a zatim tražimo najbliži neposjećeni grad i tu se nastavlja pretraga. Nakon što posjetimo sve gradove, moramo se vratiti u prvi grad. Ovaj algoritam brzo daje učinkovito kratku rutu. Za N gradova nasumično raspoređenih, algoritam u prosjeku daje put 25% duži od najkraćeg mogućeg puta. Međutim, postoje mnoge posebno uređene gradske distribucije zbog kojih ovaj algoritam daje najgoru rutu.

Vodič kroz kôd/simulaciju

Za realizaciju ovog zadatka, korišten je prethodno implementirani algoritam tabu pretraživanja, prikazan na sljedećoj slici:

```
def TS(x0, graf, N, L):
    trenutnoRj = x0
    T = []
    T.append(x0)
    i = 0
    ukupnaUdaljenost = dajUdaljenost(x0, graf)
    boljeRjesenje = x0
    while i < N:
        i += 1
        T.append(boljeRjesenje)
        boljeRjesenje, udaljenost = dajBoljeRjesenje(boljeRjesenje, graf, T)
        if i == N:
            return trenutnoRj, ukupnaUdaljenost
        else:
            if i >= L and i < N:
                T.pop(0)
            if ukupnaUdaljenost > udaljenost:
                trenutnoRj = boljeRjesenje
                ukupnaUdaljenost = udaljenost
```

Ovdje smo iskoristili tabu pretraživanje tako da nađemo permuntaciju koja daje najkracu udaljenost, tačnije u tabu listu smo dodavali permuntacije i računali ukupnu udaljenost za svaku novu permuntaciju koja nije prethodno dodana u tabu listu, te smo na taj način izbjegavali zapadanje algoritma u provjeravanje neke permuntacije koja nije optimalna. Ovaj dio algoritma je implementiran u sljedećoj funkciji:

```
def dajBoljeRjesenje(x0, graf, T):
    ukupnaUdaljenost = float('inf')
    trenutnoRj = []
    for i in range(len(x0)):
        n = i + 1
        for j in range(n, len(x0)):
            novoRj = copy.deepcopy(x0)
            novoRj[i], novoRj[j] = novoRj[j], novoRj[i]
            if novoRj not in T:
                udaljenost = dajUdaljenost(novoRj, graf)
                if ukupnaUdaljenost > udaljenost:
                    trenutnoRj = novoRj
                    ukupnaUdaljenost = udaljenost
    return trenutnoRj, ukupnaUdaljenost
```

Također na prethodnim isječcima koda možemo vidjeti da je dodana i još jedna pomoćna funkcija koja vraća ukupnu udaljenost čvorova i njen kod je dat na slici ispod:

```
def dajUdaljenost(x0, graf):
    udaljenost = 0
    vel = len(x0) - 1
    i = 0
    while i < vel:
        udaljenost += graf[x0[i]][x0[i + 1]]
        i += 1
    udaljenost += graf[x0[len(x0) - 1]][x0[0]]
    return udaljenost
```

Konačno, za izbor početne permuntacije, definisali smo metodu *greedyTSP* koja bira najbliži neposjećeni čvor I vraća tu permuntaciju I njenu ukupnu udaljenost:

```
def greedyTSP(sviCvorovi, graf):
    #print("svi cvorovi: ", sviCvorovi)
    trenutniCvor = sviCvorovi[0]
    x0 = []
    while trenutniCvor not in x0:
        x0.append(trenutniCvor)
        cvorovi = list(set(sviCvorovi).difference(set(x0)))
        if len(cvorovi) != 0:
            #nadji najblizi cvor u listi cvorova
            vel = graf[trenutniCvor][cvorovi[0]]
            najbliziCvor = cvorovi[0]
            for i in range(1, len(cvorovi)):
                vel1 = graf[trenutniCvor][cvorovi[i]]
                if vel1 < vel:
                    vel = vel1
                    najbliziCvor = cvorovi[i]
            trenutniCvor = najbliziCvor
    udaljenost = dajUdaljenost(x0, graf)
    return x0, udaljenost
```

Rezultati kôda/simulacije

Za testiranje implementacije algortima koristili smo testne grafove iz drugog zadatka i rezultati koda će biti prikazani u nastavku za svaki graf pojedinačno:

```
N = 10

W1 = [[0, 3, 3, 8, 6],
      [3, 0, 7, 2, 8],
      [3, 7, 0, 5, 2],
      [8, 2, 5, 0, 1],
      [6, 8, 2, 1, 0]]

W2 = [[0, 2, 2, 2, 7, 5, 2],
      [2, 0, 4, 7, 9, 1, 3],
      [2, 4, 0, 3, 6, 6, 5],
      [2, 7, 3, 0, 4, 9, 6],
      [7, 9, 6, 4, 0, 4, 9],
      [5, 1, 6, 9, 4, 0, 2],
      [2, 3, 5, 6, 9, 2, 0]]

W3 = [[ 0,28, 7,15, 9,24, 8,30,29,16],
      [28, 0,28, 7,12,14, 2,26,19, 0],
      [ 7,28, 0,15,20, 1,19, 0, 3,17],
      [15, 7,15, 0,19,30, 6,17, 0,13],
      [ 9,12,20,19, 0,11,30,18, 0, 2],
      [24,14, 1,30,11, 0, 1,29,29,13],
      [ 8, 2,19, 6,30, 1, 0,25,30,22],
      [30,26, 0,17,18,29,25, 0, 6,26],
      [29,19, 3, 0, 0,29,30, 6, 0,29],
      [16, 0,17,13, 2,13,22,26,29, 0]]
```

```
Pocetna permuntacija cvorova za graf W1: [0, 1, 3, 4, 2]
Ukupna udaljenost za ovaj slucaj: 11
Pocetna permuntacija cvorova za graf W2: [0, 1, 5, 6, 2, 3, 4]
Ukupna udaljenost za ovaj slucaj: 24
Pocetna permuntacija cvorova za graf W3: [0, 2, 7, 8, 3, 6, 5, 4, 9, 1]
Ukupna udaljenost za ovaj slucaj: 61


Najbolja permuntacija cvorova za graf W1: [0, 1, 3, 4, 2]
Ukupna udaljenost za ovaj slucaj: 11
Najbolja permuntacija cvorova za graf W2: [5, 1, 6, 0, 2, 3, 4]
Ukupna udaljenost za ovaj slucaj: 19
Najbolja permuntacija cvorova za graf W3: [5, 2, 7, 8, 3, 1, 9, 4, 0, 6]
Ukupna udaljenost za ovaj slucaj: 34
```

Zaključak

Na osnovu testnih grafova testirali smo nas algoritam i uvjerili smo se u ispravnost algoritma. Moguća poboljšanja algoritma bi se mogla postići koristeći neki drugi metod za generisanje početne permuntacije, jer je nerijetko moguće da nam pohlepni algoritam da najgori slučaj.

Zadatak 2

Ovaj zadatak je objedinjen u prvom zadatku, a rezultat njegove simulacije je prikazan u paragrafu “Rezultat koda/simulacije” u prvom zadatku.