Metode optimizacije

Vježba 5

Dana je udaljenost između gradova u csv formatu. Napišite program kojim ćete obići n gradova, svaki grad samo jednom uz povratak u polazišni grad, tako da napravite što kraći put:

1. koristeći brute-force,
2. koristeći algoritam najbližeg susjeda,
3. koristeći algoritam sortiranih susjeda.

Algoritmi su opisani u slajdovima sa predavanja. Za sva tri algoritma procijenite složenost i izmjerite vrijeme izvršavanja. Testirajte na datasetu `distance.csv`. Gradove koje želite obići unosi korisnik ili ih možete zapisati u neku datoteku, pa testni dio programa čita iz nje.

Čitanje datoteke

Datoteka je zapisana u `csv` formatu. Kod čitanja, zapisuje se u jednu od struktura pogodnih za zapis grafa, a po potrebi se pozivaju funkcije za konverziju iz vježbe 4.

Brute-force

Generiraju se sve permutacije skupa $\{0, 1, 2, \ldots, n-1\}$, gdje je n broj gradova.

```
from itertools import permutations
perms = list(permutations(range(n), n))
```

Za svaku permutaciju računa se trošak puta.

Primjer: za matricu

$$A = \begin{bmatrix} 0 & 160 & 3082 & 1639 \\ 160 & 0 & 2766 & 1465 \\ 3082 & 2766 & 0 & 3312 \\ 1639 & 1465 & 3312 & 0 \end{bmatrix}$$

i permutaciju

$$(0, 1, 3, 2)$$

trošak je

$$a_{01} + a_{13} + a_{32} + a_{20} = 8019$$

Među svim tako izračunatim troškovima, nađemo najmanji.

Možda je najlakše računati trošak koristeći matricu susjedstva. Ako ste graf zapisali u dictionary ovakvog oblika `g = {0 : [(1,160), (2,3082), (3,1639)],...}`, onda je trošak `g[0][1] + g[1][3] + g[3][2] + g[2][0]`.

Algoritam najbližih susjeda

Posjećene gradove spremamo u listu `visited`. Za grad koji trenutno gledamo (zadnji dodan u `visited`), uzimamo sve susjede koji nisu u visited i među njima tražimo onaj najmanje udaljenosti. Kad ga nađemo, njega stavljamo u `visited` i nastavljamo dalje.

Ako radite sa matricom susjedstva, najjednostavnije je staviti da je udaljenost od tekućeg grada do svih posjećenih 0, pa se traži najmanji pozitivni broj u odgovarajućem retku matrice.

Algoritam sortiranih bridova

Napravimo konverziju grafa u listu susjeda, tj. listu bridova. Za gornji primjer to je `edges = [(0,1,160), (0,2,3082), (0,3,1639), (1,2,2766), (1,3,11465), (2,3,3312)]]`. Sortiramo po trecem elementu u uređenoj trojci `edges.sort(key = lambda x : x[2])`

Krećemo od praznog grafa. Ponavljamo sljedeći postupak dok svi vrhovi nisu obiđeni.

- Odaberemo brid najmanje težine i dodamo ga u graf.
- Provjerimo ima li koji vrh stupanj 3 (riješeno u vježbi 4) i postoji li ciklus u grafu (koristiti pretragu u dubinu ili širinu). Ako ne, dodamo brid u graf.

Za graf je najjednostavnije koristiti dictionary `g={}`. Onda odabrani brid ovako dodajemo u graf: `g[edge[0]] = [(edge[1], edge[2])]`