Univerzitet u Banjoj Luci

Prirodno – matematički fakultet

Matematika i informatika – Informatika

Napredni koncepti baza podataka

Neo4J u rješavanju problema ograničenja

Profesor
Dr Marko Đukanović

Student
Jovana Mika 17/19

Sadržaj:

Opis problema

Problemi ograničenja spadaja u klasi matematičkih problema čija rješenja zadovoljavaju niz datih ograničenja. Svakoj promjenljivoj se dodjeljuje vrijednost iz njenog domena, pri čemu sva ograničenja moraju biti zadovoljena.

Definicija: Problem zadovoljenja ograničenja je definisan kao trojka (X,D,C) gdje je:

- $X = \{ X_1, X_2, ..., X_n\}$ – skup varijabli
- $D = \{ D_1, D_2, ..., D_n\}$ – skup domena vrijednosti
- $C = \{ C_1, C_2, ..., C_n\}$ – skup ograničenja

Problem ranca

Problem koji ćemo rješavati je problem ranca koji spada u probleme kombinatorne optimizacije. Cilj je pronaći optimalno rješenje među svim mogućim kombinacijama.

Problem je dobio ime po scenariju u kome imamo ograničen broj predmeta koji se može staviti u ranac fiksne veličine. Svaki predmet ima dva atributa, težinu i vrijednost, a cilj je dobiti što je moguće veću vrijednost u rancu, a da se pri tome ne prekorači ograničenje.

U rješavanju ovog problema koristićemo Neo4J i Graph traversal API.

Ograničenja

Kao što smo već naveli, postoji jedno ograničenje:

- Težina predmeta koje stavljamo u ranac, ne smije premašiti maksimalnu težinu ranca.

Na osnovu datog ograničenja imamo sledeća stanja:

- Početno stanje u kome je ranac prazan, odnosno težina „pokupljenih“ stvari je 0.
- Niz prelaznih stanja koja mogu biti važeća ili nevažeća. Stanje je važeće ako je ukupna težina svih predmeta koji su stavljeni u ranac manja ili jednaka od maksimalne težine ranca. Ako je ukupna težina predmeta veća od maksimalne težine ranca, stanje je nevažeće.

Opis algoritma

Model baze

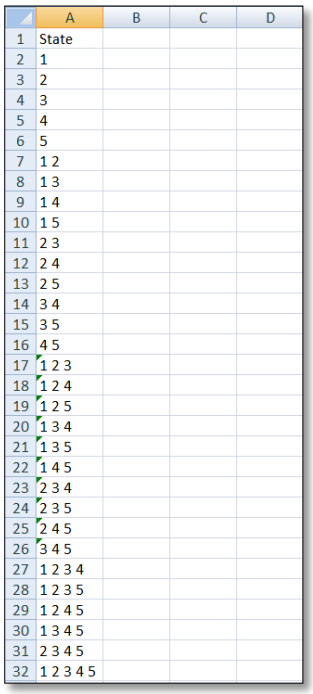
Kako bi riješili dati problem pomoću Neo4J potrebno je da prvo napravimo grafovsku bazu. Predmeti će predstavljati sva moguća stanja, odnosno čvorove grafa, dok će veze predstavljati prelaske u stanja.
Svaki čvor će imati jedan atribut, naziv čvora, koji se sastoji od imena svih prethodno posjećenih čvorova koji su međusobno povezani i odvojeni su razmakom.
Imaćemo dvije vrste čvorova:

1. Početni čvor (*InitialState*) – stanje u kome je ruksak prazan
2. Ostali čvorovi (*State*) – predstavljaju sva moguća stanja u kojima ruksak nije prazan

Svaka veza sadrži informacije o prelasku, a to su:

- Težina, koja se računa kao suma svih težina predmeta do kojih se može doći krenuvši od lista do početnog stanja, prateći tu vezu..
- Profit koji se računa na isti način kao i težina, pri čemu se sumiraju vrijednosti predmeta.
- Naziv veze, koji se formira na sledeći način *uzima_nazivČvora*.

Za kreiranje grafa u Neo4J bazi koristili smo dva csv fajla, *nodes.csv* (za čvorove) i *rel.csv* (za veze) .



	A	B	C	D
1	State			
2	1			
3	2			
4	3			
5	4			
6	5			
7	1 2			
8	1 3			
9	1 4			
10	1 5			
11	2 3			
12	2 4			
13	2 5			
14	3 4			
15	3 5			
16	4 5			
17	1 2 3			
18	1 2 4			
19	1 2 5			
20	1 3 4			
21	1 3 5			
22	1 4 5			
23	2 3 4			
24	2 3 5			
25	2 4 5			
26	3 4 5			
27	1 2 3 4			
28	1 2 3 5			
29	1 2 4 5			
30	1 3 4 5			
31	2 3 4 5			
32	1 2 3 4 5			

Slika 1. Primjer

	A	B	C	D	E
1	Label	Weight	Value	FromNode	ToNode
2	uzima_1	19	22	InitialState	1
3	uzima_2	10	45	InitialState	2
4	uzima_3	30	5	InitialState	3
5	uzima_4	15	30	InitialState	4
6	uzima_5	9	20	InitialState	5
7	uzima_2	29	67	1	1 2
8	uzima_1	29	67	2	1 2
9	uzima_3	49	27	1	1 3
10	uzima_1	49	27	3	1 3
11	uzima_4	34	52	1	1 4
12	uzima_1	34	52	4	1 4
13	uzima_5	28	42	1	1 5
14	uzima_1	28	42	5	1 5
15	uzima_3	40	50	2	2 3
16	uzima_2	40	50	3	2 3
17	uzima_4	25	75	2	2 4
18	uzima_2	25	75	4	2 4
19	uzima_5	19	65	2	2 5
20	uzima_2	19	65	5	2 5
21	uzima_4	45	35	3	3 4
22	uzima_3	45	35	4	3 4
23	uzima_5	39	25	3	3 5
24	uzima_3	39	25	5	3 5
25	uzima_5	24	50	4	4 5
26	uzima_4	24	50	5	4 5
27	uzima_3	59	72	1 2	1 2 3
28	uzima_1	59	72	2 3	1 2 3
29	uzima_2	59	72	1 3	1 2 3
30	uzima_4	44	97	1 2	1 2 4
31	uzima_1	44	97	2 4	1 2 4
32	uzima_2	44	97	1 4	1 2 4

Slika 2. Primjer rel.csv fajla

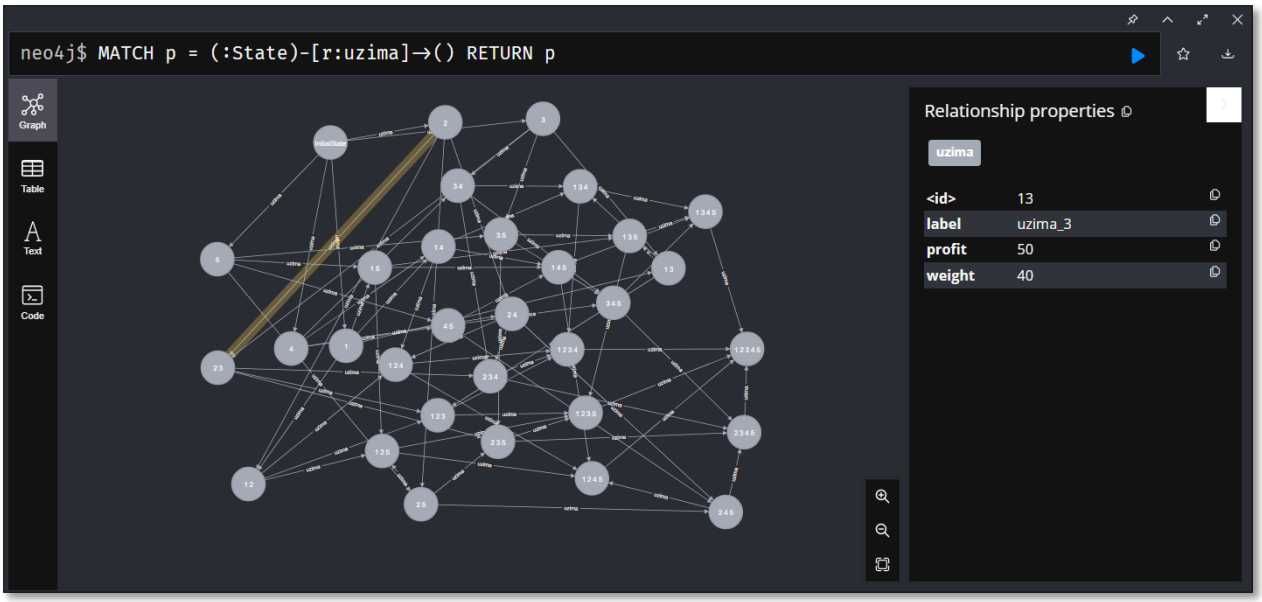
Zatim smo na osnovu ovih fajlova u Neo4J kreirali čvorove i veze između njih.

```
1 LOAD CSV WITH HEADERS FROM 'file:///nodes.csv' as p
2 MERGE (s:State{attribute:p.State})
3 return s
4
```

Slika 3. Kreiranje čvorova u Neo4J-u

```
1 LOAD CSV WITH HEADERS FROM 'file:///rel.csv' as p
2 MATCH (s1:State{attribute:p.FromNode})
3 MATCH (s2:State{attribute:p.ToNode})
4 MERGE (s1) - [:uzima{weight:toInteger(p.Weight),profit:toInteger(p.Value),label:p.Label}]
  → (s2)
5
```

Slika 4. Kreiranje veza u Neo4J-u



Slika 5. Primjer grafovske baze u Neo4J-u

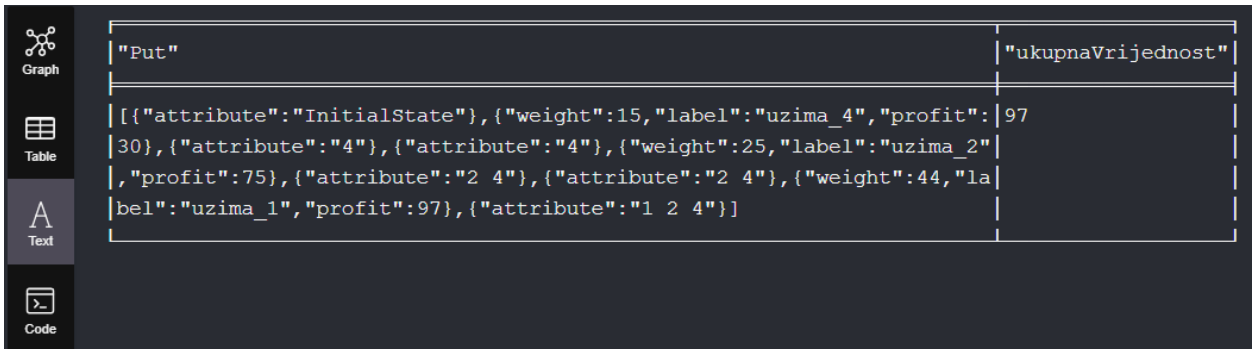
Za pronalaženje puteva koji zadovoljavaju ograničenje koristili smo proceduru koja je pisana uz pomoć Javinog Neo4j API-ja.
Koraci algoritma koji je korišten:

1. Obilazak grafa počinje od početnog čvora.
2. Posjećujemo sve susjedne čvorove, pri čemu se putevi proširuju u dubinu.
3. Provjeravamo da li se uključivanjem susjednog čvora u put krše ograničenja.
4. Ako čvor zadovoljava ograničenja, uključujemo ga u put i nastavljamo obilazak grafa.
5. Ako čvor ne zadovoljava ograničenja, isključujemo ga iz puta i obilazak grafa se ne nastavlja.
6. Iteriramo kroz kolekciju puteva koji zadovoljavaju ograničenja i vraćamo sve puteve kao rezultate.

Implementiranu proceduru pozivamo preko Neo4j upita, a kao parametar prosljeđujemo maksimalnu težinu ruksaka. Rezultate sortiramo po opadajućem poretku, i prikazujemo najbolje rješenje.

```
1 CALL constraint.findValidPaths(48) YIELD path
2 with path, relationships(path) AS rels
3 with path, rels, reduce(acc = 0,r in rels |  r.profit) AS ukupnaVrijednost
4 return path as Put, ukupnaVrijednost
5 order by ukupnaVrijednost desc
6 limit 1
7
```

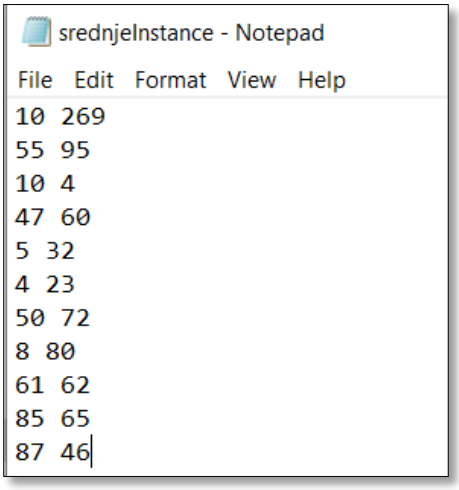
Slika 6. Primjer pozivanja procedure



Slika 7. Prikaz rješenja

Opis instanci i specifikacije računara

Korištene su tri veličine instanci. Male kod kojih je broj predmeta n = 5,srednje za koje je n = 10 i velike n = 14. U prvom redu se navodi broj predmeta i maksimalna težina ruksaka, odnosno odgraničenje. Zatim se u redovima ispod za svaki predmet navode težina i vrijednost predmeta, odvojeni razmakom.



Slika 8. Primjer instance

Specifikacije:

- Procesor: Intel(R) Core(TM) i3-6006U CPU @ 2.00GHz   1.99 GHz
- RAM: 4GB
- Operativni sistem: Windows 10

Poređenje rezultata

Za potrebe poređenja rezultata, pored rješavanja u Neo4j-u korištene su tehnike dinamičkog programiranja i rekurzivni pristup. Svi algoritmi pisani su u programskom jeziku Java.

Veličina instance - ograničenje	Vrijeme izvršavanja	Rezultat
Rekurzivni pristup		
5 - 48	< 1 ms	97
10 - 269	1 ms	431
14 - 678	7 ms	471
Dinamičko programiranje		
5 - 48	1 ms	97
10 - 269	2 ms	431
14 - 678	4 ms	471
Neo4J		
5 - 48	21 ms	97
10 - 269	17374 ms	431
14 - 250	62213 ms	471

Tabela 1. Prikaz rješenja

Na osnovu prikazanih rješenja možemo vidjeti da svi algoritmi vraćaju optimalna rješenja. U slučaju malih i srednjih instanci rekurzivnim pristupom smo najbrže došli do rješenja, dok se za velike instance najbrže do rješenja došlo dinamičkim programiranjem. U svim slučajevima Neo4J se pokazao kao najmanje efikasan. U slučaju malih instanci Neo4J-u je trebalo 21 puta više vremena od preostala dva pristupa, u slučaju srednjih instanci ovaj broj se povećao na čak 17374 puta.

Neka je n broj predmeta, a w kapacitet. Vremeska složenost rekurzivnog algoritma iznosi $O(2^n)$. Za dinamičko programiranje vremenska složenost jednaka je O(n*w), dok je za Neo4j obilazak $O(2^n)$.

Zaključak

Performanse Neo4J-a u slučaju rješavanja problema ograničenja, tačnije problema ranca, su veom loše. Neka je broj premeta n, tada će broj čvorova koji je potreban za kreiranje baze biti 2^n. Dakle, broj čvorova eksponencijalno raste pa pri svakom povećanju veličine instance performance su znatno lošije.

U našem slučaju, za rješavanje problema ograničenja bolje je koristiti druge pristupe.

Literatura

- https://medium.com/@dhananjay.ghanwat/solving-constraint-problems-using-neo4j-88bc3e456bac
- https://neo4j.com/docs/java-reference/current/#tutorial-traversal-java-api
- https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/
- https://en.wikipedia.org/wiki/Knapsack_problem