PyGDB — analiza strukture izvornog koda

Projekt iz kolegija Napredne baze podataka

Luka Šimek

Zagreb, 14. svibnja 2024.*

Sadržaj

*datum zadnje izmjene: 14. svibnja 2024.

1 Uvod

1.1 Motivacija

Suvremeni softverski paketi često u sebi sadrže nepregledno veliku količinu raznih elemenata, od modula, klasa i funkcija do samog broja linija koda. Kao korisnicima, često nam se korisno malo bolje upoznati s arhitekturom paketa s kojim radimo, bilo da je zbog što kvalitetnijeg korištenja njegovih funkcionalnosti ili debuggiranja vlastitog koda.

S tim ciljem, pogodno bi bilo imati način za analizirati strukturu nekog paketa vizualno i potencijalno iz ptičje perspektive. Također, htjeli bismo da ta analiza bude fleksibilna — kao što smo naveli, cjelokupni sadržaj paketa može biti vrlo nepregledan, stoga želimo istovremeno biti u mogućnosti vizualizirati različite module u paketu, ali i podatke o tek nekolicini konkretnih funkcija.

Primijetimo da se spomenuta struktura prirodno preslikava na graf. Za vrhove grafa uzimamo različite objekte (potpakete, module, klase, funkcije i ostale varijable), dok su bridovi dani njihovim raznim odnosima (klasa je definirana u modulu, funkcija je metoda klase, jedna klasa naslijeđuje od druge itd.)

Naši zahtjevi — sakupljanje velike količine podataka o paketu ili više njih, fleksibilno dohvaćanje raznih dijelova tih podataka i modeliranje preko grafa navode nas na korištenje grafovske baze podataka. Konkretno, u ovom projektu koristit ćemo Neo4j. Jedan od razloga za to je da je riječ o trenutno najpoznatijem sustavu za upravljanje grafovskim bazama podataka. Cypher, njegov jezik za upite, sličan je poznatom SQL-u u sinktaksi, ali i sam po sebi vrlo kvalitetan i pogodan za upite nad grafovskim bazama. Dodatno, Neo4j Browser u sebi već ima ugrađeno korisničko sučelje i mogućnosti vizualizacije rezultata upita.

1.2 Python i njegove značajke

Iako različiti programski jezici mogu dijeliti implicitnu shemu takve grafovske baze, algoritam za transformaciju programskog koda u graf nužno će ovisiti o programskom jeziku u kojemu je kod pisan. Ni zajednička shema nije nužna. Primjerice, već smo nekoliko puta spomenuli klase, no klase nisu dio svih programskih jezika.

U ovom projektu koristit ćemo Python i baviti se paketima pisanima, barem prvenstveno, u Pythonu. Jedan razlog svakako je u popularnosti jezika — Python je popularan jezik općenite uporabe i koristi se u velikom broju otvorenih paketa. Posebno spomenimo cijeli *data science* ekosustav u kojemu je Python zauzeo jednu od vodećih uloga u izradi biblioteka za znanstveno računanje, obradu i vizualizaciju podataka, statistiku i statističko učenje.

Činjenica da je Python dinamički tipiziran predstavlja izazov, ali i povećava smisao ovakvog projekta. U Pythonu ne postoje varijable u tipičnom smislu, već postoje samo *imena* koja možemo pridružiti objektima. To pridruživanje je potpuno fleksibilno. Primjerice, sljedeći kod je sasvim pravilan:

```
class A:
  pass

a = A()
A = 3
```

Dakle, dinamički je određeno na koji se objekt odnosti oznaka `A`. U ovom projektu zadržavamo se na statičkoj analizi koda — dinamička analiza bila bi skuplja vremenski i memorijski te bi bila dosta kompleksnija. Zauzvrat, naši rezultati ne moraju uvijek biti pouzdani, ali se situacija kao iznad gotovo nikad neće naći u praksi, među poznatim i kvalitetno napravljenim paketima.

Treći razlog za Python je njegova opsežna standardna biblioteka koja će nam omogućiti interakciju sa strukturama podataka koje inače koristi samo prevoditelj. Više o tome ćemo reći u sljedećem poglavlju.

Literatura

[1] Luka Šimek (lsimek2). *nbp-projekt*. 2024. URL: https://github.com/lsimek2/nbp-projekt.

[2] *ast – Abstract Syntax Trees*. Python Software Foundation. URL: https://docs.python.org/3/library/ast.html (pogledano 14. 5. 2024.).