Profajliranje ivica: Knutov algoritam i njegova unapređenja

Seminarski rad u okviru kursa
Verifikacija softvera
Matematički fakultet

Nevena Nikolić, 1021/2018
nevena134@hotmail.com

9. decembar 2018

Sažetak

Proces optimizacije programa predstavlja neizostavan deo razvoja softvera. Postoji veliki broj različitih pomoćnih alata koji su razvijeni u te svrhe. Za potrebe optimizacije mogu se koristiti podaci o broju izvršavanja delova programa koji se dobijaju tehnikom profajliranja. U ovom radu biće detaljnije opisana jedna vrsta kompajlerski zasnovanog profajliranja - profajliranje ivica, kao i algoritmi koji se za to koriste.

Sadržaj

1 Uvod

Profajliranje predstavlja vid dinamičke analize koda, čiji je rezultat skup podataka dobijen izvršavanjem programa sa određenim ulaznim podacima. Profajliranje se može posmatrati i kao ubacivanje dodatnih instrukcija u program sa ciljem prikupljanja podataka o programu za vreme njegovog izvršavanja. Drugačije rečeno, pravi se profil posmatranog programa. Dobijeni podaci nam mogu pomoći u otkrivanju „uskog grla" (tj. dela koda koji se često izvršava), curenja memorije, određivanju pokrivenosti koda datim ulazima, proširivanju skupa testova i još mnogo drugih problema.

Ubacivanja novog koda u već napisan program naziva se instrumentalizacija (eng. *instrumentation*). Na osnovu mesta gde se ubacuje dodatni kôd imamo tri vrste profajliranja:

- Profajliranje putanje (eng. *path profiling*)
- Profajliranje ivica (eng. *edge profiling*)
- Profajliranje bloka (eng. *basic–block profiling*)

Blokovi mogu biti funkcije ili deo koda u kome se ne nalaze instrukcije grananja ili skokova. Ivica predstavlja instrukciju grananja ili skoka kojom se prebacuje tok izvršavanja programa iz jednog bloka u drugi. Ona povezuje dva bloka. U nastavku se fokusiramo na problem dobijanja statističkih podataka o blokovima i ivicama programa koji su se izvršili.

2 Knutov algoritam

Da bismo brojali koliko puta se neki događaj desio prilikom izvršavanja programa, moramo da ubacimo kôd koji će vršiti uvećanje brojača. Ovo uvodi određene troškove. Najjednostavniji način bi bio da za svaki blok ili ivicu ubacimo brojač. Tada bismo imali neophodne podatke, ali bi oni zauzeli previše računarskih resursa.

Cilj nam je da minimizujemo broj umetnutih brojača. Broj izvršavanja svakog bloka možemo dobiti pomoću brojača ivica, tako što sumiramo sve ivice koje ulaze u taj blok. Profajliranje ivica može da se implementira značajno efikasnije, pod pretpostavkom da se radi u fazi kompilacije programa. Jedno takvo rešenje prvi je teoretski uveo Donald Knut (eng. *Donald Knuth*), dobitnik Tjuringove nagrade 1974. godine, i on je pokazao da je broj umetnutih brojača u njegovom rešenju minimalan [4]. Najpre uvodimo nekoliko pojmova, a zatim sledi opis samog algoritma.

2.1 Terminologija

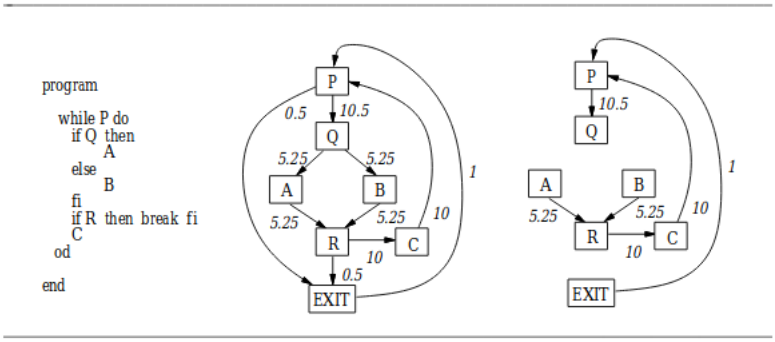
Graf kontrole toka (eng. *control-flow graph*, skraćeno: CFG) je usmereni korenski graf $G = (V, E)$ koji odgovara proceduri programa na sledeći način: svaki čvor iz V predstavlja osnovni blok instrukcija (pravolinijsku sekvencu instrukcija), a svaka ivica iz E predstavlja prebacivanje toka izvršavanja iz jednog osnovnog bloka u drugi. Dodatno, CFG uključuje i specijalan čvor $EXIT$ koji odgovara izlasku iz procedure (*return*). Koreni čvor je prvi osnovni blok u proceduri. U grafu postoji usmerena putanja od korena do svakog čvora i usmerena putanja od svakog čvora do $EXIT$ čvora. Za potrebe algoritma profajliranja, pogodno je ubaciti i ivicu od $EXIT$ čvora do korena, kako bi CFG bio čvrsto povezan graf. $EXIT$ čvor nema drugih sledbenika osim korenog čvora.

Zadavanje težina (eng. *weighting*) grafu G dodeljuje nenegativnu vrednost (celobrojnu ili realnu) svakoj ivici prema Kirhofovom zakonu: za svaki čvor v, zbir težina ivica koje ulaze u čvor v mora biti jednak zbiru težina ivica koje izlaze iz čvora v. Težina čvora jeste suma težina njegovih ulaznih (izlaznih) ivica. Cena nekog skupa ivica i/ili čvorova je suma težina ivica/čvorova od kojih je taj skup sačinjen.

Jedno **izvršavanje** (eng. *execution*) procedure predstavljeno je usmerenom putanjom EX od korena do $EXIT$ čvora kroz CFG. Frekvencija čvora v ili ivice e u nekom izvršavanju EX jeste broj pojavljivanja v ili e u EX. Ukoliko se čvor ili ivica ne pojavljuje u EX, odgovarajuća frekvencija je nula, sa izuzetkom ivice $EXIT \rightarrow koren$, čija je frekvencija za svako izvršavanje jednaka broju pojavljivanja $EXIT$ čvora u tom izvršavanju.

Razapinjuće stablo (eng. *spanning tree*) usmerenog grafa $G = (V, E)$ je podgraf $H = (V, T)$, gde je $T \subseteq E$, takav da je svaki par čvorova iz V povezan jedinstvenom putanjom (odnosno, svaka dva čvora su povezana u H i H ne sadrži cikluse). **Maksimalno** razapinjuće stablo težinskog grafa je ono razapinjuće stablo čije grane imaju maksimalnu cenu (razapinjuće stablo nekog grafa nije jedinstveno!). Postoji više algoritama za efikasno nalaženje maksimalnog razapinjućeg stabla [5].

Primer 2.1 *Slika 1 ilustruje prethodne definicije.*



Slika 1: Program, odgovarajući težinski CFG i maksimalno razapinjuće stablo

2.2 Opis algoritma

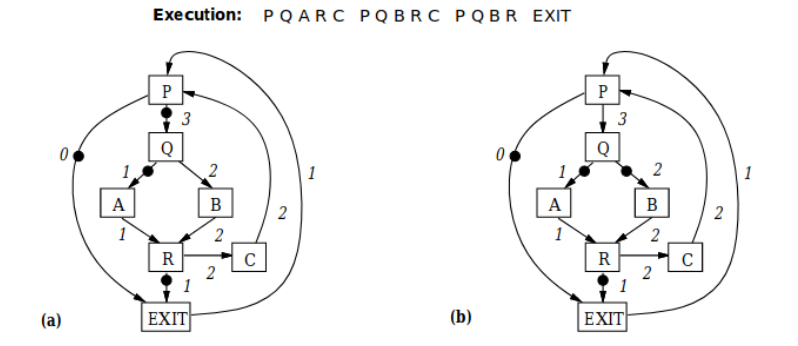
Označimo kraće problem profajliranja ivica sa $Eprof(Ecnt)$: odrediti postavljanje brojača $Ecnt$ (skup nekih ivica) u CFG G tako da se frekvencija svake ivice u bilo kom izvršavanju može jednoznačno utvrditi na osnovu CFG i izmerenih frekvencija ivica u $Ecnt$. Knut je pokazao kako iz Kirhofovog zakona sledi da postavljanje brojača ivica $Ecnt$ rešava $Eprof(Ecnt)$ za CFG $G = (V, E)$ akko $(E - Ecnt)$ ne sadrži (neusmereni) ciklus [3]. Kako razapinjuće stablo grafa kontrole toka predstavlja maksimalan podskup ivica bez ciklusa, sledi da je $Ecnt$ rešenje minimalne veličine problema $Eprof(Ecnt)$ akko je $(E - Ecnt)$ razapinjuće stablo od G. Dakle, minimalan broj brojača neophodnih za rešavanje $Eprof(Ecnt)$ je $|E| - (|V| - 1)$.

Da bismo videli kako ovakvo postavljanje rešava problem nalaženja frekvencija svih ivica, posmatrajmo CFG G i skup $Ecnt$ takav da je $E - Ecnt$ razapinjuće stablo grafa G. Neka svaka ivica e iz $Ecnt$ ima pridružen brojač koji je inicijalno postavljen na 0 i uvećava se svaki put kada se e

izvrši. Ako je čvor v list u razapinjućem stablu (drugim rečima, samo jedna ivica stabla je susedna čvoru v), onda su sve ostale ivice susedne čvoru v u skupu $Ecnt$. Kako frekvencije ivica za neko izvršavanje zadovoljavaju Kirhofov zakon, neizmerena frekvencija ivice je jedinstveno određena jednačinom toka za v i poznatim frekvencijama ostalih ulaznih i izlaznih ivica čvora v. Preostale ivice sa nepoznatim frekvencijama i dalje formiraju stablo, pa se postupak može ponavljati sve dok frekvencije svih ivica iz $E - Ecnt$ nisu jedinstveno određene. Ako $E - Ecnt$ ne sadrži cikluse a nije razapinjuće stablo, onda je $E - Ecnt$ šuma. Navedeni pristup može se primeniti na svako stablo posebno kako bi se odredile frekvencije ivica iz $E - Ecnt$.

Bilo koji od dobro poznatih algoritama za računanje maksimalnog razapinjućeg stabla će efikasno pronaći ovakvo stablo za graf kontrole toka, uz poštovanje težina. Ivice koje ne pripadaju dobijenom stablu rešavaju problem $Eprof(Ecnt)$ i minimizuju cenu skupa $Ecnt$. Kao rezultat, brojači su postavljeni u delovima sa nižom frekvencijom izvršavanja u CFG. Imamo garanciju da brojač nikad neće biti postavljen na ivici koja spaja $EXIT$ čvor sa korenom (za proizvoljne CFG i zadate težine uvek postoji maksimalno razapinjuće stablo koje uključuje pomenutu ivicu); izračunata frekvencija te ivice zapravo predstavlja broj izvršavanja procedure kojoj odgovara dati CFG.

Primer 2.2 *Slika 2 ilustruje kako se frekvencije ivica skupa $E - Ecnt$ mogu izračunati na osnovu frekvencija ivica skupa $Ecnt$. Crne tačke identifikuju ivice skupa $Ecnt$, dok su preostale ivice - ivice skupa $E - Ecnt$ i one formiraju razapinjuće stablo. Za dato izvršavanje prikazane su frekvencije svih ivica. Neka je P koren razapinjućeg stabla. Čvor Q je list razapinjućeg stabla i ima jednačinu toka ($P \to Q = Q \to A + Q \to B$). Kako su frekvencije ivica $P \to Q$ i $Q \to A$ poznate, možemo zameniti te vrednosti u pomenutoj jednačini i tako dobiti frekvenciju ivice $Q \to B$. Sada kada je poznata frekvencija ivice $Q \to B$, frekvenciju ivice $B \to R$ možemo izračunati iz jednačine toka čvora B i tako dalje. Za težine date na Slici 1, postavljanje brojača prikazano na Slici 2(a) ima cenu 16.75. Slika 2(b) prikazuje postavljanje brojača dobijeno na osnovu maksimalnog razapinjućeg stabla i ono ima cenu 11.5.*



Slika 2: Rešavanje $Eprof(Ecnt)$ pomoću razapinjućeg stabla

Algoritam prikazan na Slici 3 koristi post-order obilazak razapinjućeg stabla $E - Ecnt$ za propagiranje frekvencija ivica skupa $Ecnt$ na neinstrumentalizovane ivice. Procedura DFS izračunava frekvenciju jedne ivice razapinjućeg stabla.

```
global
    G: control-flow graph
    E: edges of G
    cnt: array[edge] of integer      /* for each edge e in Ecnt, cnt[e] = frequency of e in execution */

procedure propagate_counts(Ecnt: set of edges)
begin
    for each e ∈ E - Ecnt do cnt[e] := 0 od
    DFS(Ecnt, root-vertex(G), NULL)
end

procedure DFS(Ecnt: set of edges; v: vertex; e: edge)
let   IN(v) = { e' | e' ∈ E and v = tgt(e') }   and   OUT(v) = { e' | e' ∈ E and v = src(e') }   in
    in_sum := 0
    for each e' ∈ IN(v) do
        if (e' ≠ e) and e' ∈ E - Ecnt then DFS(Ecnt, src(e'), e') fi
        in_sum := in_sum + cnt[e']
    od
    out_sum := 0
    for each e' ∈ OUT(v) do
        if (e' ≠ e) and e' ∈ E - Ecnt then DFS(Ecnt, tgt(e'), e') fi
        out_sum := out_sum + cnt[e']
    od
    if e ≠ NULL then cnt[e] := max(in_sum, out_sum) - min(in_sum, out_sum) fi
ni
```

Slika 3: Algoritam za računanje frekvencija ivica koje pripadaju razapinjućem stablu na osnovu frekvencija ivica skupa $Ecnt$

Napomena: Postupak profajliranja je opisan za jednan graf kontrole toka, koji odgovara jednoj proceduri programa. Za programe koji imaju više procedura, algoritam zahteva par izmena [2].

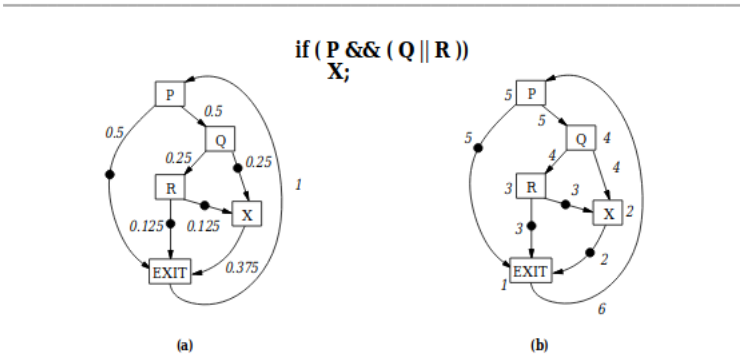
3 Unapređenja

Knutovo rešenje može istrumentalizovati isti broj ivica, ali na različite načine. Standardni algoritmi računanja razapinjućeg stabla mogu vratiti različita stabla u zavisnosti od redosleda obrade grana. Da bi profajliranje bilo efikasno, kôd za instrumentalizaciju bi trebalo smestiti u one delove koji se ne izvršavaju često. Čini se da je za pronalaženje tih informacija potrebno profajliranje. Tomas Bal (eng. *Tomas Ball*) i Džejms Larus (eng. *James R. Larus*) su prikazali način kako da se proceni koji skup ivica je najoptimalniji korišćenjem jednostavne heuristike za zadavanje težina ivicama, zasnovane na analizi grafa kontrole toka [2].

Osnovna ideja je zadavanje manjih težina onim ivicama koje su dublje ugnježdene u uslovnim kontrolnim strukturama, s obzirom da će se ovi delovi ređe izvršavati. Uopšteno, svaka putanja kroz petlju zahteva instrumentalizaciju. Unutar petlje koja sadrži uslove, i dalje bismo želeli da instrumentalizacija bude što je moguće dublje ugnježdena.

Primer 3.1 *Za graf kontrole toka na Slici 4, heuristika će generisati težine prikazane u (a).*

Postoje skupi metodi za generisanje težina zasnovani na matricama, a prednost ove heuristike je što zahteva samo pretragu u dubinu (eng. *depth-first search*) i topološki obilazak (eng. *topological traversal*) grafa kontrole toka.



Slika 4: Fragment programa, (a) njegov CFG sa težinama koje zadovoljavaju Kirhofov zakon i optimalno postavljanje brojača (ivice sa crnim tačkama), (b) zadavanje težina koristeći post-order nabrajanje čvorova (težina ivice je post-order broj njenog izvornog čvora) i suboptimalno postavljanje brojača koje je rezultat nalaženja maksimalnog razapinjućeg stabla koje poštuje te težine

Heuristika se sastoji iz nekoliko koraka. Prvo, pretragom u dubinu grafa kontrole toka od njegovog korena identifikujemo ivice povratka tog grafa. Heuristika potom koristi topološki obilazak grafa dobijenog od polaznog grafa izbacivanjem ivica povratka da računa težine. Koriste se **prirodne petlje** (eng. *natural loops*) za identifikaciju petlji i ivica izlaska iz petlje [1]. Prirodna petlja neke ivice povratka $x \to y$ se definiše na sledeći način:

$$nat - loop(x \to y) = \{y\} \cup \{w | \text{postoji usmerena putanja od } w \text{ do } x \text{ koja ne sadrži } y\}$$

Čvor je **ulazak u petlju** (eng. *loop-entry*) ako je on cilj jedne ili više ivica povratka. Prirodna petlja ulaska u petlju y, u oznaci $nat - loop(y)$, je jednostavno unija svih prirodnih petlji $nat - loop(x \to y)$, gde je $x \to y$ ivica povratka. Ako su a i b različiti čvorovi ulaska u petlju, onda su skupovi $nat - loop(a)$ i $nat - loop(b)$ ili disjunktni ili je jedan u potpunosti sadržan u drugom. Ovo svojstvo ugnježdenosti se koristi da bi se definisale **ivice izlaska iz petlje** (eng. *loop-exit edges*) neke petlje sa ulaskom y:

$$loop - exits(y) = \{a \to b \in E | a \in nat - loop(y) \text{ i } b \notin nat - loop(y)\}$$

Ivica $a \to b$ je ivica izlaska iz petlje ako postoji ulazak u petlju y tako da je $a \to b \in loop - exits(y)$.

Heuristika pretpostavlja da je broj iteracija svake petlje jednak LOOP_MULTIPLIER (za našu implementaciju 10) i svaka grana nekog iskaza može biti izabrana sa istom verovatnoćom. Ivicama izlaska iz petlje se posebno rukuje, kako je opisano u nastavku. Težina ivice od čvora $EXIT$ do korena je fiksirana na 1 i ne menja se. Ova ivica se ne tretira kao ivica povratka, iako je identifikovana kao takva pretragom u dubinu.

Pravila koja slede opisuju kako se računaju težine čvorova i ivica:

1. Težina čvora jednaka je sumi težina njegovih ulaznih ivica koje nisu ivice povratka
2. Ako je čvor v ulazak u petlju sa težinom W i $N = |loop - exits(v)|$, onda svaka ivica skupa $loop - exits(v)$ ima težinu W/N
3. Ako je čvor v ulazak u petlju, neka je W težina čvora v pomnožena sa LOOP_MULTIPLIER, inače neka je W težina čvora v. Ako je W_E suma težina izlaznih ivica čvora v koje su ivice izlaska iz petlje, onda svaka izlazna ivica čvora v koja nije ivica izlaska iz petlje ima težinu $(W - W_E)/N$, gde je N ukupan broj izlaznih ivica od v koje nisu ivice izlazska iz petlje.

Ova pravila se primenjuju u pojedinačnom topološkom obilasku grafa dobijenog od grafa kontrole toka izbacivanjem ivica povratka. Jednoj ivici (možda ivici povratka) dodeljuje se težina na osnovu prvog pravila koje za nju važi u obilasku, kako sledi. Kada se čvor v prvi put poseti u toku obilaska, težine njegovih ulaznih ivica koje nisu ivice povratka su poznate. Prvo pravilo određuje težinu čvora v. Ako je čvor v ulazak u petlju, onda se koristi drugo pravilo kako bi se dodelile težine svakoj ivici skupa $loop - exits(v)$. Konačno, treće pravilo određuje težinu svake izlazne ivice čvora v koja nije ivica izlaska iz petlje.

4 Zaključak

Opisani algoritmi optimizuju ubacivanje koda za instrumentalizaciju, uz poštovanje težina grafa kontrole toka. Empirijski rezulati na stvarnim programima pokazuju da ovi algoritmi uspešno smanjuju troškove instrumentalizacije. Ipak, ostaje nekoliko otvorenih pitanja: (1) Da li postoji efikasan algoritam koji će optimalno rešiti problem frekvencija čvorova na osnovu skupa brojača ivica ili ovaj problem ostaje težak za rešavanje? (2) Da li postoje bolje sheme zadavanja težina koje bi preciznije uputile postavljanje koda za instrumentalizaciju?

„Computer programming is an art dorm, like the creation of poetry or music"

Donald E. Knuth

Literatura

[1] R. Sethi A. Aho and J. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, Reading, MA, 1986.

[2] Thomas Ball and James R.Larus. Optimally profiling and tracing programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(4):1319–1360, 1994.

[3] D. E. Knuth. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Addison-Wesley, Reading, MA, 2 edition, 1973.

[4] Donald E. Knuth and Francis R. Stevenson. Optimal measurement points for program frequency counts. *U: BIT Numerical Mathematics*, 3(13):313–322, 1973.

[5] D. W. Wall. Predicting program behavior using real or estimated profiles. In *Proceedings of the SIG-PLAN 91 Conference on Programming Language Design and Implementation*, pages 59–70, Toronto, June 1991. ACM SIGPLAN Notices.