VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS INSTITUTAS
INFORMATIKOS KATEDRA

Kursinis projektas

Rikiavimo tobulinimas genetiniais algoritmais

(Improving sorting with genetic algorithms)

Atliko: 4 kurso 2 grupės studentas
Deividas Zaleskis (parašas)

Darbo vadovas:
Irmantas Radavičius (parašas)

Vilnius
2022

Turinys

Įvadas

Duomenų rikiavimas yra vienas pamatinių informatikos uždavinių. Matematiškai jis formuluojamas taip: duotai baigtinei palyginamų elementų sekai $S = (s_1, s_2, ..., s_n)$ pateikti tokį kėlinį, kad duotosios sekos elementai būtų išdėstyti monotonine (didėjančia arba mažėjančia) tvarka [RB13]. Efektyvus šio uždavinio sprendimas buvo svarbus, kai informatikos mokslo sąvoka dar neegzistavo - skaitmeninio rikiavimo algoritmas (angl. radix sort) buvo naudojamas perforuotų kortelių rikiavimui jau 1923 metais. Atsiradus kompiuteriams, rikiavimo uždavinys tapo dar aktualesnis ir buvo laikomas vienu pagrindinių diskrečių uždavinių, kuriuos turėtų gebėti spręsti kompiuteris [Knu70]. Efektyvus rikiavimo uždavinio sprendimas dažnai padeda pagrindą efektyviam kito uždavinio sprendimui, pavyzdžiui, atliekant paiešką sąraše, naivią paiešką tikrinant visus elementus iš eilės galima pakeisti dvejetaine paieška, jei sąrašas yra išrikiuotas ir sumažinti uždavinio sprendimo laiko sudėtingumą iš $O(n)$ į $O(log\, n)$. Nepaisant to, jog šis uždavinys yra nagrinėjamas nuo pat informatikos mokslo pradžios, nauji rikiavimo algoritmai ir įvairūs patobulinimai egzistuojantiems algoritmams yra kuriami ir dabar.

Rikiavimo uždaviniui spręsti egzistuoja labai įvairių algoritmų. Dažniausiai jie yra klasifikuojami pagal šiuos kriterijus: rėmimąsi palyginimu (palyginimu paremti algoritmai gauna informaciją apie duomenis tik remdamiesi palyginimo operacijomis), laiko sudėtingumą (optimalūs palyginimu paremti algoritmai blogiausiu atveju turi $O(n\, log\, n)$ laiko sudėtingumą), atminties sudėtingumą (optimaliu atveju - $O(1)$), stabilumą (stabilūs algoritmai nekeičia lygių elementų tvarkos).

Šelo rikiavimo algoritmas (angl. Shellsort, toliau - Šelo algoritmas) [She59] yra paremtas palyginimu, nenaudojantis papildomos atminties ir nestabilus. Šelo algoritmą galima laikyti rikiavimo įterpimu modifikacija, kuri lygina ne gretimus, o toliau vienas nuo kito esančius elementus, taip paspartindama jų perkėlimą į galutinę poziciją. Pagrindinė algoritmo idėja - išskaidyti rikiuojamą seką S į posekius $S_1, S_2, ..., S_n$, kur kiekvienas posekis $S_i = (s_i, s_{i+h}, s_{i+2h}, ...)$ yra sekos S elementai, kurių pozicija skiriasi h. Išrikiavus visus sekos S posekius S_i su tarpu h, seka tampa h-išrikiuota. Remiantis tuo, jog sekai S esant h-išrikiuota ir ją k-išrikiavus, ji lieka h-išrikiuota [GK72], galima kiekvieną algoritmo iteraciją mažinti tarpą, taip vis didinant sekos S išrikiuotumą. Įprastai paskutinėje iteracijoje atliekamas rikiavimas su tarpu 1, kas užtikrina jog bus atliekamas rikiavimas įterpimu ir seka bus pilnai išrikiuota.

Šelo algoritmo efektyvumas tiesiogiai priklauso nuo pasirinktos tarpų sekos [Ciu01; Sed96]. Yra įrodyta, kad Šelo algoritmo laiko sudėtingumo blogiausiu atveju apatinė riba yra $\Omega(\frac{n\, log^2\, n}{log\, log\, n^2})$ [PPS92], taigi jis nėra asimptotiškai optimalus. Tiesa, kol kas nėra rasta seka, su kuria Šelo algoritmas pasiektų šią apatinę ribą. Kiek žinoma autoriui, asimptotiškai geriausia tarpų seka yra rasta Pratt, kuri yra formos 2^p3^p ir turi $\Theta(n\, log^2\, n)$ laiko sudėtingumą blogiausiu atveju [Pra72], tačiau praktikoje ji veikia lėčiau už Ciura [Ciu01] ar Tokuda [Tok92] eksperimentiškai rastas sekas.

Darbo **tikslas**: pritaikyti genetinius algoritmus rikiavimo algoritmų generavimui.

Darbo uždaviniai:

- Atlikti Šelo algoritmo ir jo variantų literatūros analizę.

- Atlikti genetinių algoritmų literatūros analizę.
- Nustatyti kriterijus rikiavimo algoritmų efektyvumui įvertinti.
- Paruošti aplinką rikiavimo algoritmų generavimui.
- Pasitelkiant genetinius algoritmus sugeneruoti rikiavimo algoritmus.
- Paruošti aplinką rikiavimo algoritmų tarpusavio palyginimui.
- Atliekant eksperimentus įvertinti sugeneruotų ir klasikinių rikiavimo algoritmų efektyvumą.

Šis darbas sudarytas iš 7 skyrių. Pirmame skyriuje atliekama Šelo algoritmo ir jo variantų literatūros analizė. Antrame skyriuje atliekama genetinių algoritmų literatūros analizė. Trečiame skyriuje nustatomi kriterijai rikiavimo algoritmų efektyvumui įvertinti. Ketvirtame skyriuje paruošiama aplinka rikiavimo algoritmų generavimui. Penktame skyriuje pasitelkiant genetinius algoritmus generuojami rikiavimo algoritmai. Šeštame skyriuje paruošiama aplinka rikiavimo algoritmų tarpusavio palyginimui. Septintame skyriuje atliekant eksperimentus įvertinamas sugeneruotų ir klasikinių rikiavimo algoritmų efektyvumas.

1. Šelo algoritmas ir jo variantai

Šis skyrius sudarytas iš 2 poskyrių. Pirmame poskyryje nagrinėjamas Šelo algoritmas. Antrame poskyryje aptariami Šelo algoritmo variantai.

1.1. Šelo algoritmas

Vadovėlinis Šelo algoritmas [She59] (toliau - VŠA) yra vienas iš seniausių (D. L. Shell paskelbtas 1959 m.) ir geriausiai žinomų rikiavimo algoritmų. Taip pat jis yra ir vienas iš paprasčiausiai įgyvendinamų, ką galima pastebėti iš pseudokodo, pateikiamo 1 algoritme.

Algorithm 1 Vadovėlinis Šelo rikiavimo algoritmas

```
1:  foreach gap in H do
2:      for i ← gap to N − 1 do
3:          j ← i
4:          temp ← S[i]
5:          while j > gap and S[j − gap] > S[j] do
6:              S[j] ← S[j − gap]
7:              j ← j − gap
8:          end while
9:          S[j] ← temp
10:     end for
11: end for
```

1.2. Šelo algoritmo variantai

Šelo algoritmas yra unikalus savo variantų gausa. Literatūroje Šelo algoritmo variantais vadinamos ir šio algoritmo implementacijos, kuriose naudojamos kitokios nei Šelo pasiūlytos tarpų sekos, ir implementacijos, kurių kodas skiriasi nuo vadovėlinės versijos. Šiame darbe nagrinėsime tuos variantus, kurių kodas skiriasi nuo vadovėlinės versijos.

Kaip Šelo algoritmo varianto pavyzdį galime pateikti patobulintą Šelo algoritmą (toliau - PŠA) [RB13]. Šio algoritmo pseudokodas pateikiamas žemiau, 2 algoritme. Autorių teigimu [RB13], PŠA vidutiniškai atlieka 40-80% mažiau priskyrimų ir veikia 20% greičiau, nei VŠA. Šį skirtumą galima paaiškinti tuo, jog vykdant vidinį vadovėlinio Šelo algoritmo ciklą (1 algoritmo 5-8 eilutės), 5 eilutėje yra tikrinama, ar $S[j]$ jau yra tinkamoje pozicijoje. Jei $S[j]$ jau yra tinkamoje pozicijoje, vidinis ciklas nėra vykdomas ir jokių elementų pozicijos nėra keičiamos, tačiau 4 ir 9 eilutėse vis tiek yra veltui atliekami du priskyrimai. Tuo tarpu PŠA prieš vykdydamas bet kokius kitus žingsnius patikrina, ar elementas $S[j]$ jau yra tinkamoje pozicijoje (žr. 2 algoritmo 3 eil.) ir taip sumažina atliekamų priskyrimų skaičių.

Algorithm 2 Patobulintas Šelo rikiavimo algoritmas

```
1: foreach gap in H do
2:     for i ← gap to N − 1 do
3:         if S[i − gap] > S[i] then
4:             j ← i
5:             temp ← S[i]
6:             repeat
7:                 S[j] ← S[j − gap]
8:                 j ← j − gap
9:             until j ≤ gap or S[j − gap] ≤ S[j]
10:            S[j] ← temp
11:        end if
12:    end for
13: end for
```

Nesunku pastebėti, jog VŠA ir PŠA struktūra yra tokia pati, ir algoritmai skiriasi tik tuo, kaip atliekama elementų rikiavimo logika. Šelo algoritmo variantams būdingą struktūrą toliau vadinsime Šelo algoritmo karkasu, o karkaso viduje atliekamą rikiavimo logiką - perėjimu (angl. pass). VŠA taikomą perėjimą toliau vadinsime įterpimo perėjimu, o PŠA taikomą perėjimą - patobulintu įterpimo perėjimu. Šelo algoritmo karkaso apibrėžimas pateikiamas pseudokodu 3 algoritme.

Algorithm 3 Šelo algoritmo karkasas

```
1: foreach gap in H do
2:     perform pass with gap
3: end for
```

Dobosiewicz vienas pirmųjų pastebėjo, jog pasitelkiant Šelo algoritmo karkasą ir pakeitus rikiavimo logiką (perėjimą) taip pat galima sukonstruoti pakankamai efektyvų algoritmą [Dob+80]. Dobosiewicz taikytas perėjimas yra labai panašus į burbuliuko rikiavimo algoritmo (angl. bubble sort) atliekamas operacijas: einama iš kairės į dešinę, palyginant ir (jei reikia) sukeičiant elementus vietomis. Todėl literatūroje šis perėjimas dažnai vadinamas burbuliuko perėjimu (angl. bubble pass) [Sed96]. Jo pseudokodas pateikiamas 4 algoritme.

Algorithm 4 Burbuliuko perėjimas

```
1: for i ← 0 to N − gap − 1 do
2:     if S[i] > S[i + gap] then
3:         swap(S[i], S[i + gap])
4:     end if
5: end for
```

Tiesa, burbuliuko metodą galima nežymiai patobulinti, suteikiant jam daugiau simetrijos ir atliekant perėjimą tiek iš kairės į dešinę, tiek iš dešinės į kairę. Tokiu būdu dešinėje esantys elementai greičiau pasieks savo galutinę poziciją. Šis metodas primena kokteilio purtymą, todėl literatūroje dažnai vadinamas kokteilio plaktuvo rikiavimu (angl. cocktail shaker sort). Šio algoritmo taikomą perėjimą, kurį toliau vadinsime purtymo perėjimu (angl. shake pass), integravus į Šelo algoritmo karkasą taip pat gaunamas gana įdomus algoritmas [IS86]. Purtymo perėjimo pseudokodas pateikiamas 5 algoritme.

Algorithm 5 Purtymo perėjimas

1: **for** $i \leftarrow 0$ **to** $N - gap - 1$ **do**
2: **if** $S[i] > S[i + gap]$ **then**
3: $swap(S[i], S[i + gap])$
4: **end if**
5: **end for**
6: **for** $i \leftarrow N - gap - 1$ **to** 0 **do**
7: **if** $S[i] > S[i + gap]$ **then**
8: $swap(S[i], S[i + gap])$
9: **end if**
10: **end for**

Dar viena burbuliuko algoritmo modifikacija yra nelyginis-lyginis rikiavimas (angl. odd-even sort arba brick sort). Šio algoritmo perėjimo idėja - išrikiuoti visas nelyginių/lyginių indeksų gretimų elementų poras, o tada atlikti tą patį visoms lyginių/nelyginių indeksų gretimų elementų poroms. Šį perėjimą, kurį toliau vadinsime plytos perėjimu (angl. brick pass) [Sed96], nesunkiai galima pritaikyti ir Šelo algoritmo karkasui, kintamuoju pakeitus originaliame algoritme taikytą tarpą 1 [Lem94]. Plytos perėjimo pseudokodas yra pateikiamas 6 algoritme.

Algorithm 6 Plytos perėjimas

1: **for** $i \leftarrow gap$ **to** $N - gap - 1$ **step** $2 * gap$ **do**
2: **if** $S[i] > S[i + gap]$ **then**
3: $swap(S[i], S[i + gap])$
4: **end if**
5: **end for**
6: **for** $i \leftarrow 0$ **to** $N - gap - 1$ **step** $2 * gap$ **do**
7: **if** $S[i] > S[i + gap]$ **then**
8: $swap(S[i], S[i + gap])$
9: **end if**
10: **end for**

Išvados

Išvadose visą darbą sukišam į porą puslapių, tad čia gana svarbi dalis.

Literatūra

[Ciu01] Marcin Ciura. Best increments for the average case of shellsort. *International Symposium on Fundamentals of Computation Theory*, p.p. 106–117. Springer, 2001.

[Dob+80] Wlodzimierz Dobosiewicz ir k.t. An efficient variation of bubble sort, 1980.

[GK72] David Gale ir Richard M. Karp. A phenomenon in the theory of sorting. *Journal of Computer and System Sciences*, 6(2):103–115, 1972. ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(72)80016-3. URL: https://www.sciencedirect.com/science/article/pii/S0022000072800163.

[IS86] Janet Incerpi ir Robert Sedgewick. *Practical variations of shellsort*. Disertacija, INRIA, 1986.

[Knu70] Donald E. Knuth. Von Neumann's First Computer Program. *ACM Comput. Surv.*, 2(4):247–260, 1970-12. ISSN: 0360-0300. DOI: 10.1145/356580.356581. URL: https://doi.org/10.1145/356580.356581.

[Lem94] P Lemke. The performance of randomized Shellsort-like network sorting algorithms. *SCAMP working paper P18/94*. Institute for Defense Analysis, 1994.

[PPS92] C. G. Plaxton, B. Poonen ir T. Suel. Improved lower bounds for Shellsort. *Proceedings., 33rd Annual Symposium on Foundations of Computer Science*, p.p. 226–235, 1992. DOI: 10.1109/SFCS.1992.267769.

[Pra72] Vaughan R Pratt. Shellsort and sorting networks. Tech. atask., STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1972.

[RB13] Irmantas Radavičius ir Mykolas Baranauskas. An empirical study of the gap sequences for Shell sort. *Lietuvos matematikos rinkinys*, 54(A):61–66, 2013-12. DOI: 10.15388/LMR.A.2013.14. URL: https://www.journals.vu.lt/LMR/article/view/14899.

[Sed96] Robert Sedgewick. Analysis of Shellsort and related algorithms. *European Symposium on Algorithms*, p.p. 1–11. Springer, 1996.

[She59] D. L. Shell. A High-Speed Sorting Procedure. *Commun. ACM*, 2(7):30–32, 1959-07. ISSN: 0001-0782. DOI: 10.1145/368370.368387. URL: https://doi.org/10.1145/368370.368387.

[Tok92] Naoyuki Tokuda. An Improved Shellsort. *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture - Information Processing '92, Volume 1 - Volume I*, p.p. 449–457, NLD. North-Holland Publishing Co., 1992. ISBN: 044489747X.