Matematički fakultet

Projekat iz predmeta konstrukcija i analiza algoritama 2

Školska 2018/2019

# Grejemov algoritam

*Student:*
Uroš Poznan 269/2013

*Mentori:*
dr Vesna Marinković
Mirko Spasić

Decembar, 2018

# 1 Problem

Sam problem koji je postavljen pred nas je iz oblasti geometrije, a sam algoritam tj. rešenje našeg problema spada u grupu geometrijskih algoritama. Prvo, definisaćemo i razjasniti neke osnovne pojmove kako bi čitaocu bila jasna cela problematika, a potom i rešenje. Objekti sa kojima radimo su tačke, vektori, duži i mnogouglovi.

**Tačka** je predstavljena svojim koordinatama u ravni, odnosno, u kodu, strukturom point koja sadrži koordinate x i y. **Vektor** je, takodje, svojim koordinatama i za to, u kodu, koristimo istu strukturu point koja sadrži koordinate x i y.**Duž** se zadaje parom tačaka koje predstavljaju njegove krajeve. **Put** P je niz tačaka $p_1, p_2 \dots p_k$ i duži $p_1 - p_2, p_2 - p_3, \dots, p_{k-1} - p_k$ koje ih povezuju.

**Prost mnogougao** je takav mnogougao kod koga odgovarajući put nema preseka sa samim sobom; odnosno, jedine ivice koje imaju zajedničke tačke su susedne ivice sa njihovim zajedničkim temenom.

**Konveksni mnogougao** je mnogougao za koji važi da, ma koje dve tačke iz unutrašnjosti mnogougla spojimo u duž, sve tačke te duži će ostati u unutrašnjosti mnogougla. **Konveksni omotač** datog skupa tačaka definiše se kao najmanji konveksni mnogougao koji sadrži sve tačke datog skupa . [4]

---

**Zadatak:** *Dato je n tačaka u ravni. Konstruisati konveksni omotač za njih.*

---

# 2 Rešenje

Algoritam koji će biti prikazan nalazi se u literaturi pod imenom *Grejemov algoritam*. Naziv nosi po *Ronaldu Grejemu*(eng. Ronald Graham) koji je ovaj algoritam objavio 1972. godine. Ovaj algoritam nam daje rešenje nalaženja konveksnog omotača sa najboljim performansama.

Zadatak možemo preformulisati na sledeći način: *odrediti podskup zadatog skupa tačaka tako da taj podskup predstavlja najmanji konveksni mnogougao,* *a sve tačke zadatog skupa tačaka koje nisu elementi tog podskupa se nalaze u unutrašnjosti tog mnogougla.* [3]

Ulaz za Grejemov algoritam je skup od $n$ tačaka za koje se traži konveksni omotač, a izlaz je skup od $k$ tačaka ($n \geq 3, k \geq 3, m \leq n$) koje, kada se spoje, redom, predstavljaju mnogougao koji čini konveksni omotač.

**ulaz:** $p_1, p_2, ...p_n$ (skup tačaka u ravni)
**izlaz:** $q_1, q_2, ...q_k$ (konveksni omotač tačaka $p_1, p_2, ...p_n$)

Prvo što radimo je uređivanje tačaka već poznatim agoritmom **Prost_mnogougao** [1], koji zadati niz tačaka reorganizuje tako da tačke, respektivno, predstavljaju susedna temena mnogougla.

Jednim prolaskom kroz niz nalazimo tačku koja ima najmanju $y$ koordinatu, u slučaju da ima više takvih, biramo onu sa najvećom $x$ koordinatom, i označavamo tako da nam to bude tačka $p_1$ tj. stavljamo je na prvo mesto u nizu.

Zatim sortiramo ostale tačke $p_i$ u nizu rastuće prema uglovima koji prava $p_1 - p_i$ zaklapa sa nekom proizvoljnom pravom, u našem slucaju, sa apscisom. Tačnije, za tačke $p_1$ i $p_i$ pravimo vektor $\overrightarrow{p_1p_i}$ i računamo uglove sa apscisom (vektor $\overrightarrow{e_1} = (1, 0)$) iz formule za skalarni proizvod vektora $\overrightarrow{p_1p_i}$ i $\overrightarrow{e_1}$. Sada imamo niz sortiran tako da, idući od $p_1$ do $p_n$ dobijamo prost mnogougao. Potom inicijalizujemo rezultujući skup tačaka i u njega ubacujemo tačke $p_1, p_2$ i $p_3$ tj. dobijamo prve tri tačke našeg konveksnog omotača $q_1, q_2$ i $q_3$

Ideja je da, u nastavku, redom, u rezultujući skup ubacujemo tačku po tačku $p_i$ uz proveru da li nam neka od prethodno ubačenih tačaka kvari uslov konveksnosti. Ukoliko tačka kvari uslov, izbacujemo je iz rezultujućeg skupa, u skup ubacujemo tekuću tačku $p_i$ i prelazimo na sledeću tačku u nizu.

*Uslov konveksnosti* nam se kvari ukoliko je ugao u unutrašnjosti mnogougla izmedju dve susedne stranice veći ili jednak $\pi$, u geometrijskom smislu.

Utvrđivanje da li 3 tačke formiraju "levi" ili "desni zaokret" ne zahteva stvarno računanje ugla izmedju dve duži i može biti izvedeno samo korišćenjem proste aritmetike. Za tri tačke $p_1 = (x_1, y_1), p_2 = (x_2, y_2)$ i $p_3 = (x_3, y_3)$, dovoljno je naći vektorski proizvod $\overrightarrow{p_1p_2}$ i $\overrightarrow{p_1p_3}$, koji se određuje izrazom $(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$. [2]

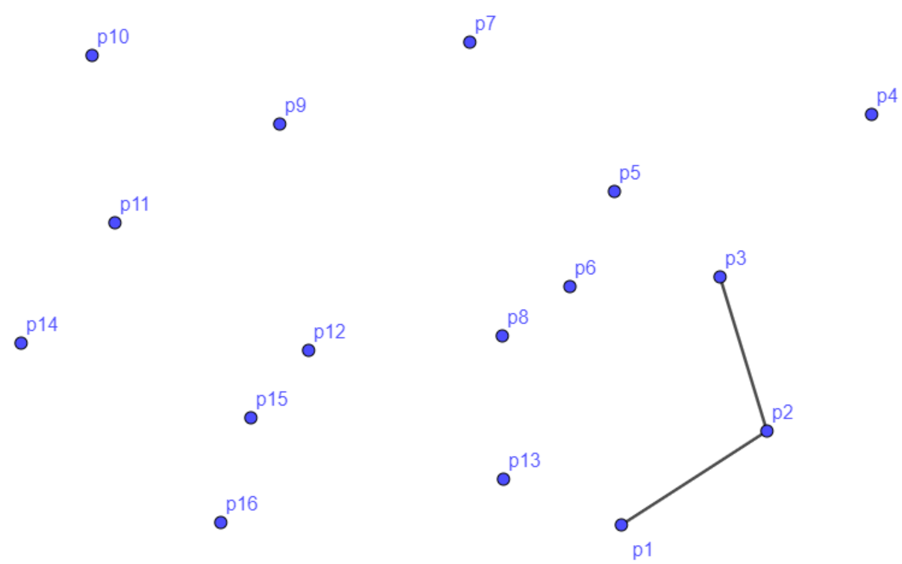
Ako je rezultat 0, tačke su kolinearne; ako je pozitivan, tri tačke formiraju "levi zaokret" ili direktnu orijentaciju, u suprotnom, formira se "desni zaokret" ili obrnuta orijentacija (za tačke koje redom numerišemo u direkt-

noj orijenataciji).
Kao što je prethodno pomenuto, u rezultujući skup tačaka ubacujemo tačke $p_1, p_2$ i $p_3$. Ukoliko je ugao izmedju pravih $p_i - q_m$ i $q_m - q_{m-1} > 0$, znamo da ti kraci u unutrasnjosti mnogougla zaklapaju ugao veći ili jednak $\pi$ i iz rezultujućeg skupa izbacujemo tačku $q_m$. Ukoliko, pak, to nije slučaj, postojeći rezultujući skup samo proširujemo tekućom tačkom $p_i$, bez prethodnih brisanja.

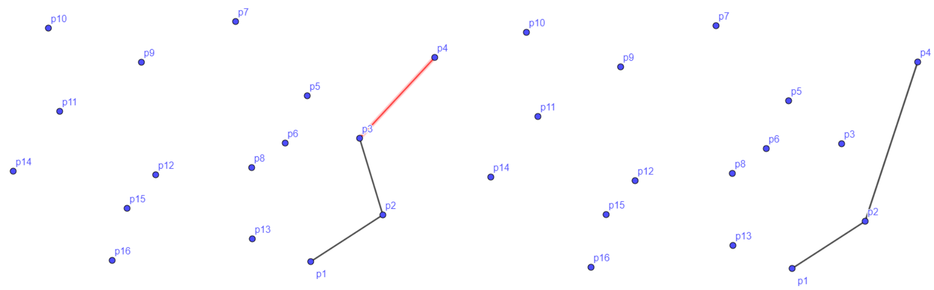
Kada zatvorimo krug, tj kada nam tekuća tačka postane $p_1$, to nam ujedno predstavlja i poslednju iteraciju.

## 2.1 Vizuelni prikaz algoritma za proizvoljan skup tačaka

U nastavku će biti vizuelno prikazani koraci izvršavanja algoritma. Crvenom bojom se boji duž koja se trenutno razmatra, odnosno čiji je početak tačka koja je poslednja ubačena u rezultujući skup, a kraj tekuća tačka $p_i$. Isprekidane linije se sustiču u tački koja je upravo izbačena iz rezultujućeg skupa tačaka, dok su crnom linijom povezane one tačke koje su trenutno u rezultujućem skupu.

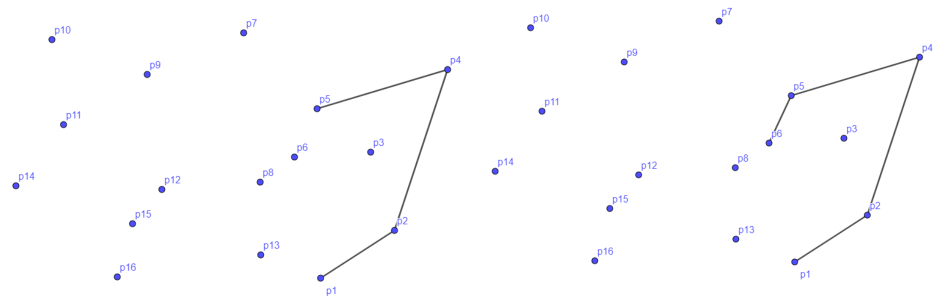



Slika 1: Početno stanje, pre ulaska u petlju



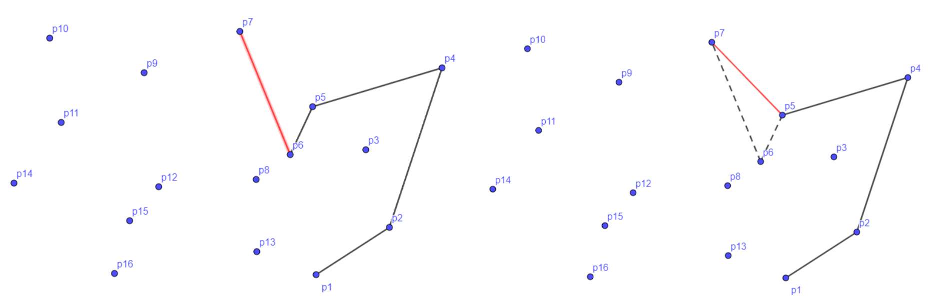

(a) Korak 2 (b) Korak 3

Slika 2: Trenutno stanje



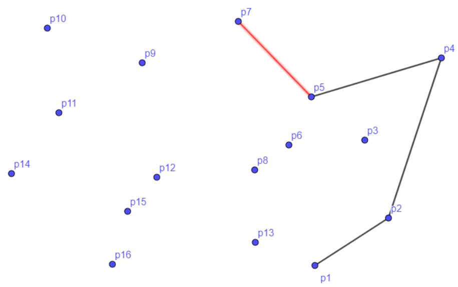

(a) Korak 4 (b) Korak 5

Slika 3: Trenutno stanje

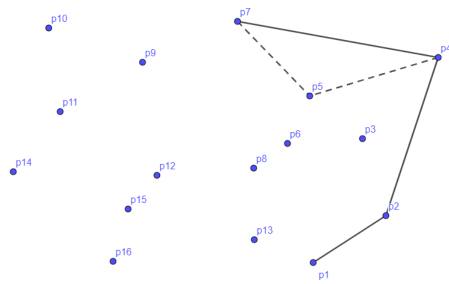



(a) Korak 6 (b) Korak 7

Slika 4: Trenutno stanje

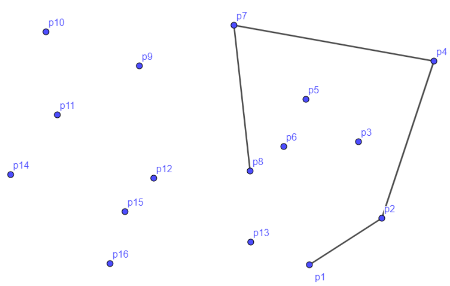



(a) Korak 8

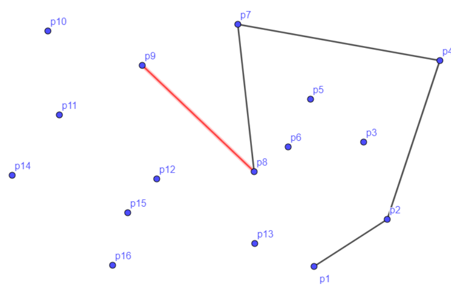



(b) Korak 9

Slika 5: Trenutno stanje

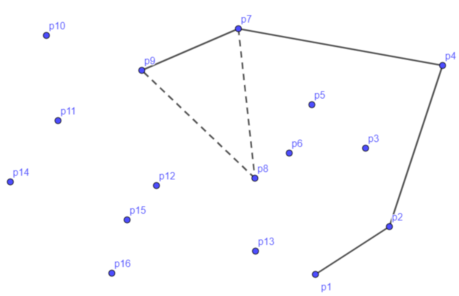



(a) Korak 10

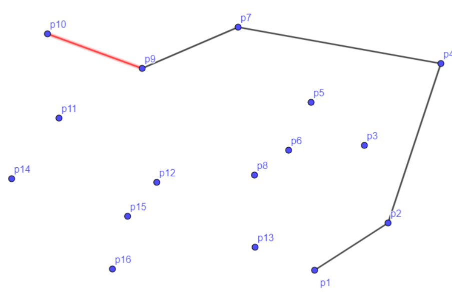



(b) Korak 11

Slika 6: Trenutno stanje

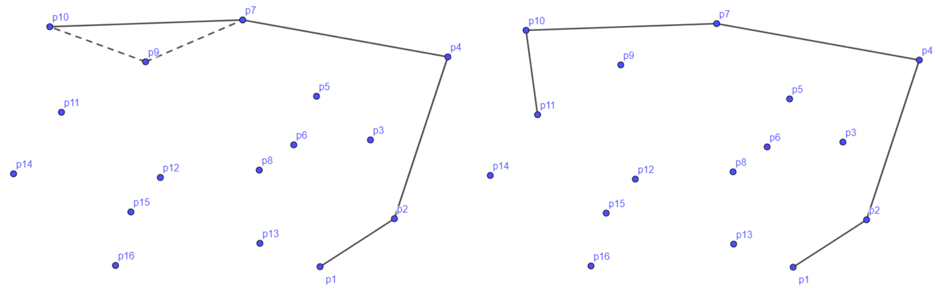



(a) Korak 12



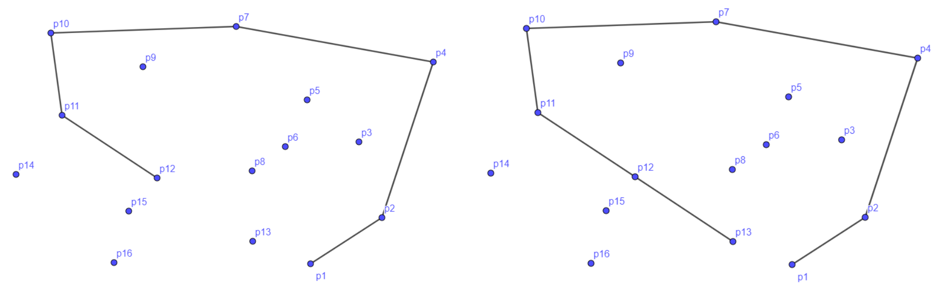

(b) Korak 13

Slika 7: Trenutno stanje



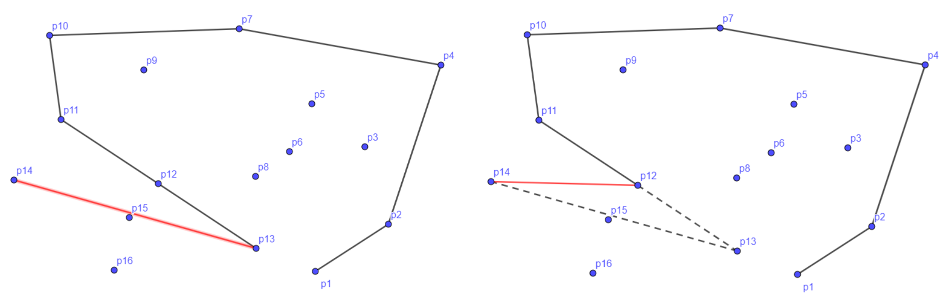

(a) Korak 14 (b) Korak 15

Slika 8: Trenutno stanje



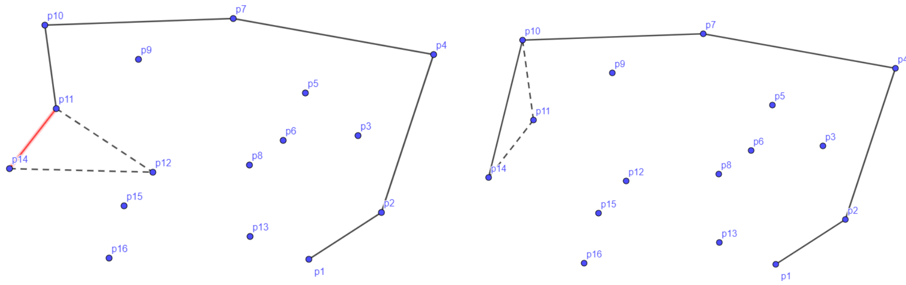

(a) Korak 16 (b) Korak 17

Slika 9: Trenutno stanje



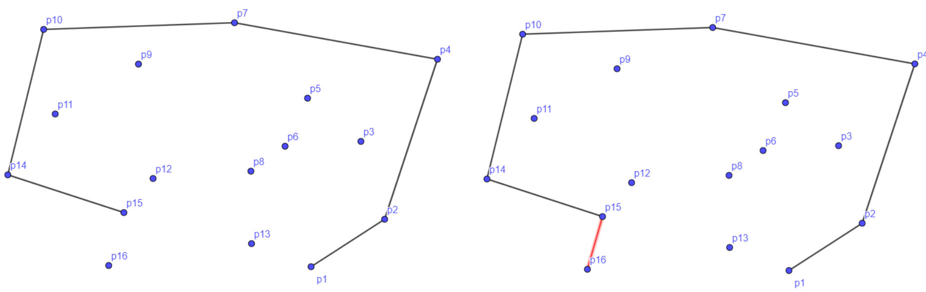

(a) Korak 18 (b) Korak 19

Slika 10: Trenutno stanje



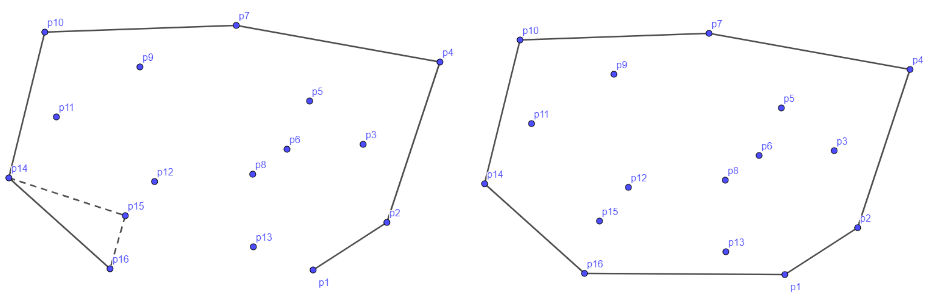

(a) Korak 20 (b) Korak 21

Slika 11: Trenutno stanje




(a) Korak 22 (b) Korak 23

Slika 12: Trenutno stanje




(a) Korak 24 (b) Korak 25

Slika 13: Trenutno stanje

# 3 Složenost

Prvi deo algoritma, u kome se formira prost mnogougao, ima složenost $O(n \cdot logn)$. Imamo jedan prolazak kroz sve tačke, u potrazi za tačkom $p_1$, što nam je složenosti O(n). Zatim, imamo sortiranje niza algoritmom sortiranja (u našem kodu *Qsort*) koje je prosečne složenosti $O(n \cdot logn)$, a sortiranje obavljamo prema uglovima koje računamo u konstantnom vremenu. Dakle $O(n + n \cdot logn) = O(n \cdot logn)$. U nastavku, iako se, na prvi pogled, čini da je složenost petlje $O(n^2)$, jer se za svaku tačku vraćamo unazad kako bismo proverili da li ona sa nekim od prethodnih obrazuje desni zaokret, vremenska složenost je zapravo $O(n)$, jer se svaka tačka razmatra samo jednom (ili se kod nje završava izvršavanje unutrašnje petlje ili se izbacuje iz daljeg razmatranja), tačnije ubacuje se, odnosno izbacuje iz rezultujućeg vektor najviše jednom. Prema tome, sveukupna vremenska složenost je $O(n \cdot logn)$, jer ova vrednost početnog sortiranja preovlađuje nad operacijama za stvarno određivanje konveksnog omotača.

# 4 Rezultati

U ovom poglavlju predstavićemo rezultate merenja brzine algoritma tj. našeg programa za različite ulaze, odnosno za ulaze različitih veličina. Time želimo da potvrdimo složenost algoritma koju smo procenili u prethodnom poglavlju i koja iznosi $O(n \cdot log(n))$.

Ono što znamo za složenost $O(n \cdot log(n))$ je da sa povećanjem ulaza raste nešto brže od linearne složenosti $O(n)$, a mnogo sporije od kvadratne složenosti $O(n^2)$.

Kako vidimo iz naše tablice i test primera, osetna razlika u promeni vremena pri povećanju ulaza, vidi se na tri poslednje vrste u Tabeli 1. Gde vidimo da se sa dupliranjem broja tačaka vreme izvršavanja povećava malo više od 2 puta. Te na ovom malom skupu primera vidimo da je rast malo brži od linearnog. Naravno, kada bismo testirali za mnogo veće ulaze, ta razlika bi bila sve uočljivija (kako $n \to \infty$), ali nikada ne bismo došli do kvadratne složenosti.

Nažalost, kako nas bibliotečka funkcija *qsort*, koju smo iskoristili i koja za najveću dužinu niza koji sortira uzima vrednost koja staje u *int*, sprečava da povećamo broj tačaka i tako testiramo algoritam, moraćemo da se zadovoljimo ovim rezultatima. Naravno, iako se podrazumeva, nije na odmet pomenuti da rezultati merenja mogu da variraju u zavisnosti od koordinata tačaka koje smo uzeli za testiranje i zato skaliranje koje dobijemo pri povećanju ulaza ne može precizno da se prikaže.

| Tačke | Milisekunde |
|---|---|
| 1 000 | 54.06 |
| 2 000 | 56.88 |
| 4 000 | 66.99 |
| 8 000 | 90.38 |
| 16 000 | 154.9 |
| 32 000 | 225.5 |
| 64 000 | 373.1 |
| 128 000 | 700.4 |
| 256 000 | 1429 |
| 512 000 | 2649 |
| 1 024 000 | 5820 |
| 2 048 000 | 11820 |

Tabela 1: Tabela odnosa veličine ulaza i brzine izvršavanja programa

# 5 Literatura


[1] Nikola Ajzenhamer. Konstrukcija i analiza algoritama 2. http://nikolaajzenhamer.rs/pdf/kiaa2-v.pdf. (Accessed on 12/17/2018).

[2] Vukmirović Srdjan. Geometrija1. http://poincare.matf.bg.ac.rs/~vsrdjan/geometrija1.pdf. (Accessed on 12/17/2018).

[3] Miodrag Zivkovic. Algoritmi. *Matematicki fakultet, Beograd*, 316, 2000.

[4] Miodrag Zivkovic and Vesna Marinkovic. Konstrukcija i analiza algoritama 2. http://poincare.matf.bg.ac.rs/~vesnap/kaa2/kaa2.pdf. (Accessed on 12/17/2018).