Univerza v Ljubljani
Fakulteta za matematiko in fiziko

Voronoijevi diagrami

RAČUNALNIŠTVO 2

Avtor naloge:

Gal Zakrajšek

Ljubljana, 2022

Kazalo

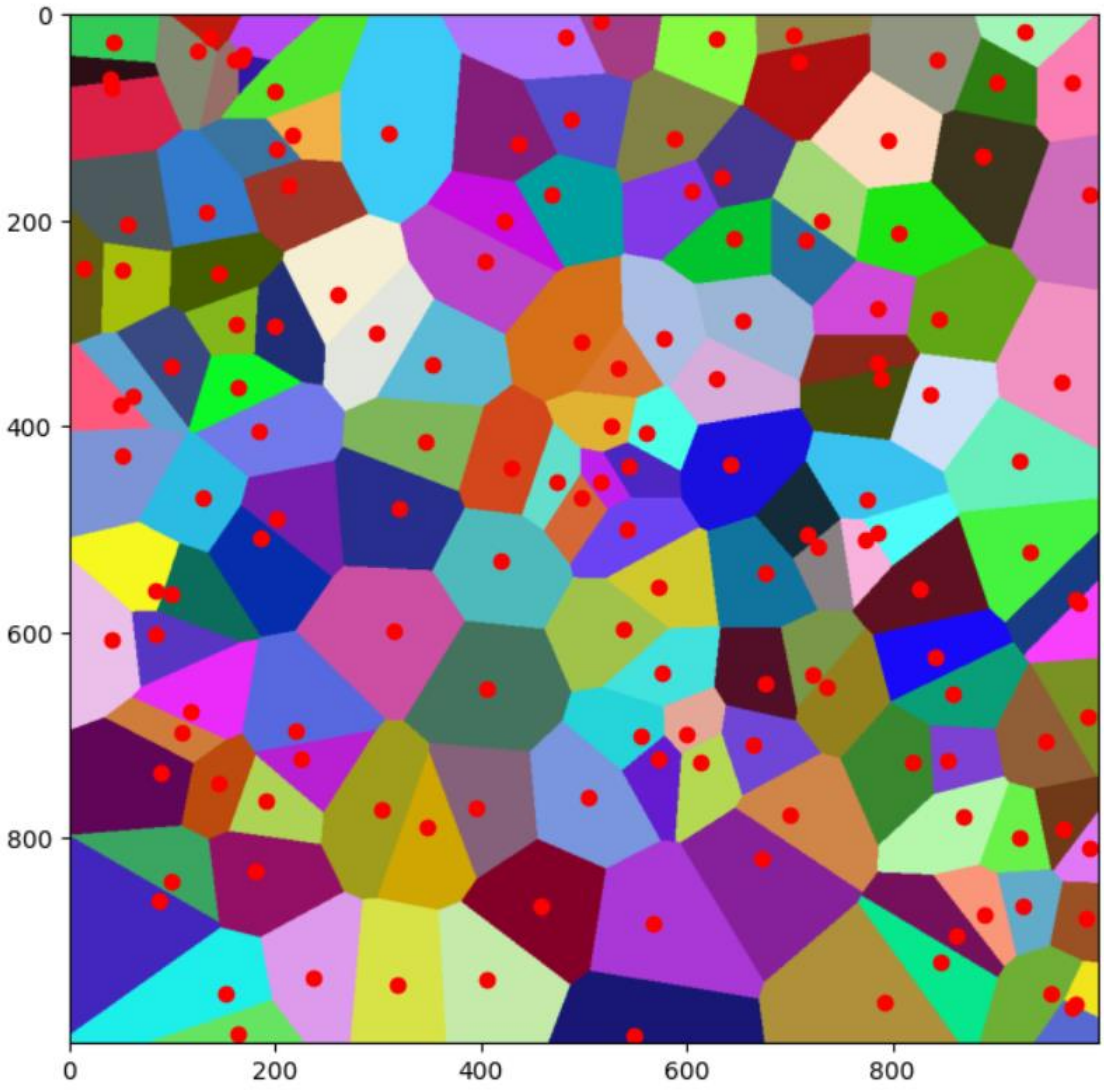
1. Uvod

Voronoijev diagram predstavlja ravnino razdeljeno na območja. Ta območja so oblikovana s pomočjo središčnih točk, ki jih bomo v nadaljevanju imenovali semena. Vsako seme ima okoli sebe območje, ki vsebuje vse točke, ki so po razdalji najbližje temu semenu. Za lažjo predstavo, bomo na vseh slikah obarvali območja vsako z svojo barvo.

Pogledali si bomo tri različne načine, s katerimi pridemo do Voronoijevih diagramov. Prvi način bo naivno oziroma z ''brute force'' metodo. Drugi algoritem, pa bo nekakšna izboljšava prvega. Tretji najbolj kompleksen način, bo tako imenovana Delaunayeva triangulacija, ki v osnovi rešuje drug problem. Izkaže se, da če gledamo dual te triangulacije, dobimo ravno Voronoijeve diagrame.

Delaunayeva triangulacija se v današnjem času veliko uporablja pri računalniški grafiki, saj algoritem v osnovi razdeli ploskev na skupek trikotnikov. Današnje grafične kartice vse 3D modele hranijo v obliki trikotnikov, saj jih najhitreje izrisujejo in ravno tukaj pride prav Delaunayeva triangulacija.

Na spodnji sliki si poglejmo primer Voronoijevega diagrama. Vsaka rdeča pika predstavlja eno seme, vse okoli pa so območja pobarvana z različnimi barvami. Vsako območje pripada svojemu semenu.



2. Zgodovina

Ideja o Voronoijevih diagramih se je v zgodovini večkrat pojavila, prvi pa jih je definiral Georgy Feodosevich Voronoy, ki je leta 1908 objavil članek o n-dimenzionalnih Voronoijevih diagramih. Voronoy, se je rodil leta 1868 in je že v srednji šoli napisal svoj prvi matematični članek, ter sodeloval z različnimi matematičnimi profesorji.

3. Naivna ''Brute force'' metoda

Že samo ime naivna o metodi pove, da bo preprosta za razumeti a časovno zelo potratna. Ravnino tukaj glejmo kot mrežo točk. Lahko si predstavljamo piksle na ekranu.

Torej imamo mrežo $N \times N$ točk na kateri je M semen. Za vsako točko na ravnini pogledamo razdaljo do vseh M semen in najdemo tistega, kateremu je najbližje. Ko preverimo vsa semena, bomo zagotovo vedeli h kateremu semenu spada ta točka in jo pobarvamo z njegovo barvo. Potem nadaljujemo na naslednji točki in ponovno pogledamo razdaljo do vseh M semen.

Že kratek opis nam pove, da je dela ogromno, saj je za vsako točko potrebno izračunati M razdalj. Vseh teh točk pa je $N \times N$, zato je tudi časovna zahtevnost algoritma $O(N^2 * M)$.

4. Jump Flood algoritem

Pri tem algoritmu bomo delovali ravno nasprotno kot pri naivni metodi, saj se ne bomo premikali, po še ne pobarvanih točkah, ampak bomo začeli barvati iz semen. Po nekaj iteracijah te metode, bomo prišli do enakega rezultata. Algoritem, ki ga bomo pogledali ni stoodstotno natančen, vendar pa je število napak zanemarljivo majhno. Kasneje bomo pogledali tudi nekaj izboljšav algoritma, ki zmanjša število napak.

4.1. Postopek

Ponovno imamo mrežo $N \times N$ točk in pa M semen. Za naš algoritem, je potrebno definirati še korak K:

$$K \in \frac{N}{2}, \frac{N}{4}, \dots, 1$$

Kot je razvidno, se bo ta korak v vsaki iteraciji zmanjšal za polovico, dokler ne pride do vrednosti 1.

Algoritem se v eni iteraciji sprehodi čez vsa semena in pogleda vse njihove sosede, ki jih določi z pomočjo koraka K. Sosede bomo definirali z Q in jih določali na način:

$$Q = (x + i, y + j), kjer\ sta\ i, j \in -k, 0, k$$

To pomeni, da iz vsakega semena pogledamo v 8 smeri. V primeru, da z korakom pridemo izven naše mreže, tistega soseda izpustimo.

Ko obravnavamo soseda, se lahko pojavita dve možnosti:

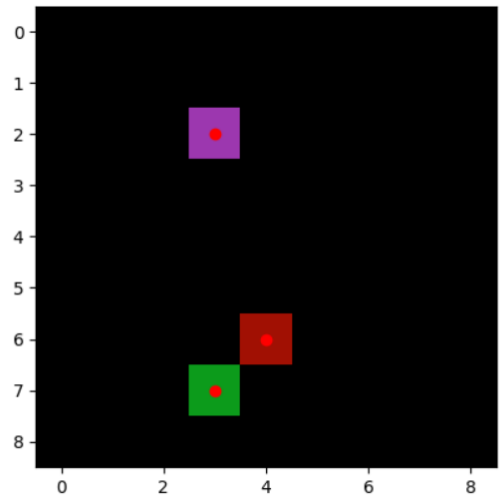
- ***Q* še nima barve**: Pobarvamo ga z barvo, ki jo ima trenutno seme. Q dodamo med semena, in ga bomo v naslednji iteraciji upoštevali kot seme.
- ***Q* že ima barvo**: Primerjamo razdaljo med Q in trenutnim semenom in pa razdaljo med Q in tistim semenom, h kateremu trenutno spada Q. V kolikor je razdalja trenutnega semena in Q manjša, barvo spremenimo. V nasprotnem primeru ohranimo barvo Q.

Algoritem ima časovno zahtevnost $O(N^2 * \log_2 N)$. Ko to primerjamo z časovno zahtevnostjo naivne metode, vidimo, da imamo enkrat N^2 množen z M drugič pa z $\log_2 N$. Iz tega je razvidno, da pri majhnem številu semen rezultat hitreje dobimo z naivno metodo, vendar ko se število semen poveča, je algoritem Jump flood hitrejši.

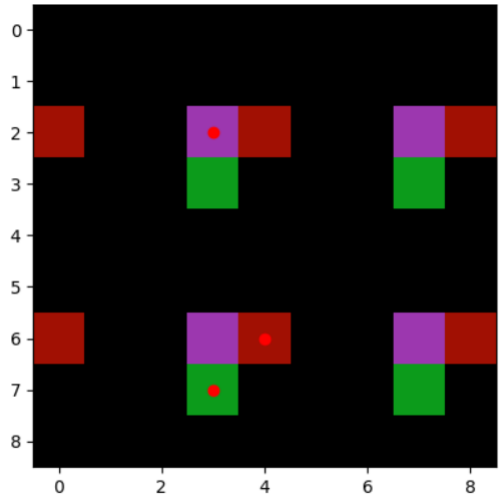
Za lažjo predstavo, kako deluje barvanje našega platna z Jump flood, si poglejmo dva primera:

1. Primer : 8×8 točk in pa 3 semena.

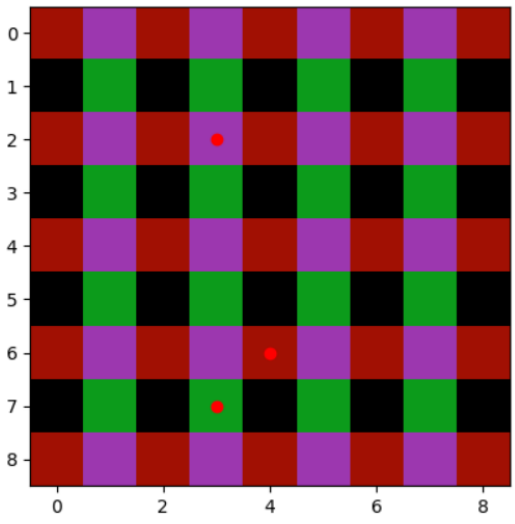
1.iteracija



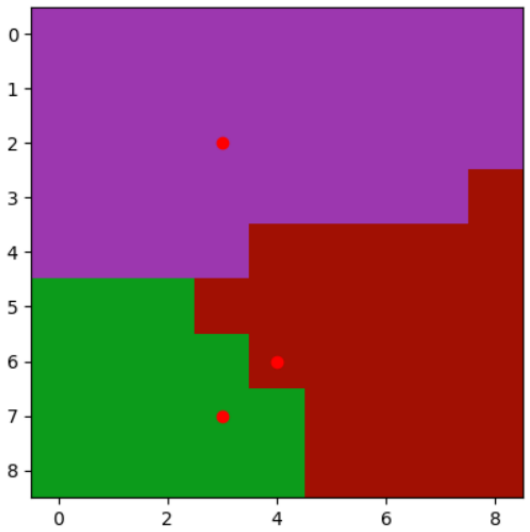
2.iteracija



3.iteracija

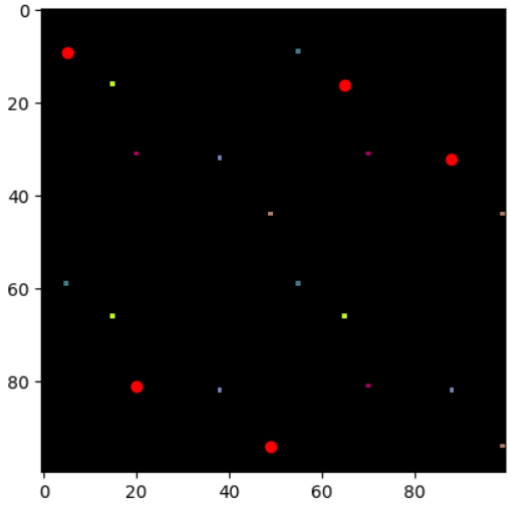


4.iteracija

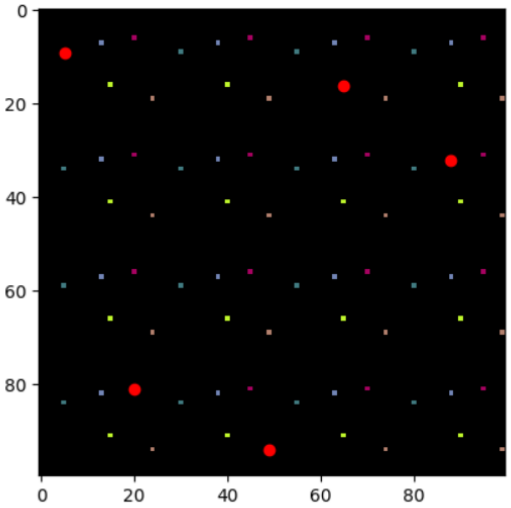


2. Primer : 100×100 točk in pa 5 semen.

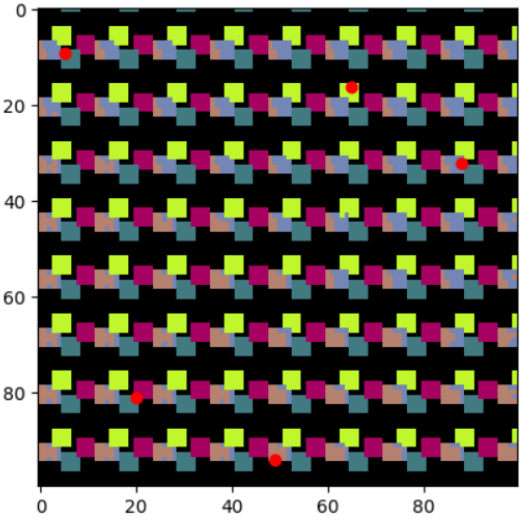
1.iteracija



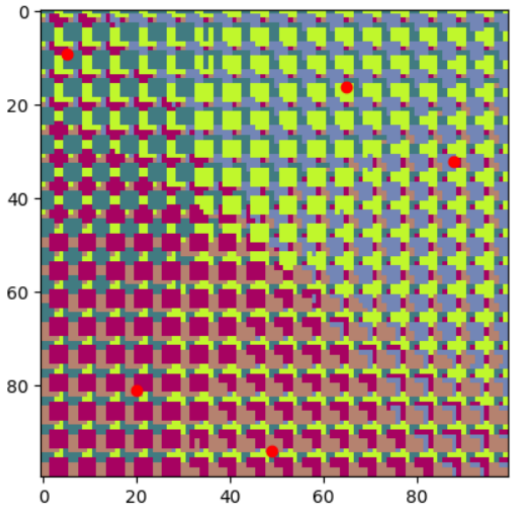
2.iteracija



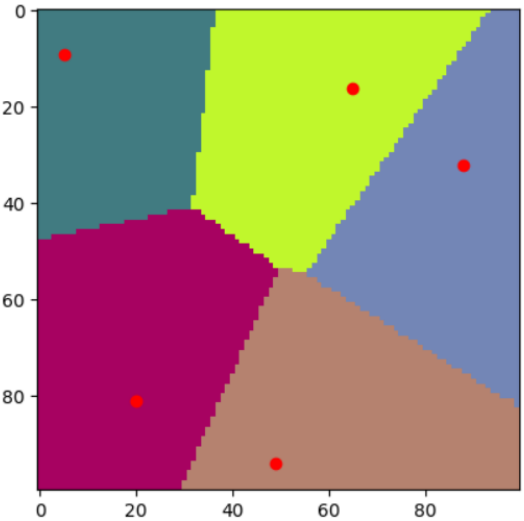
3.iteracija



4.iteracija



5.iteracija

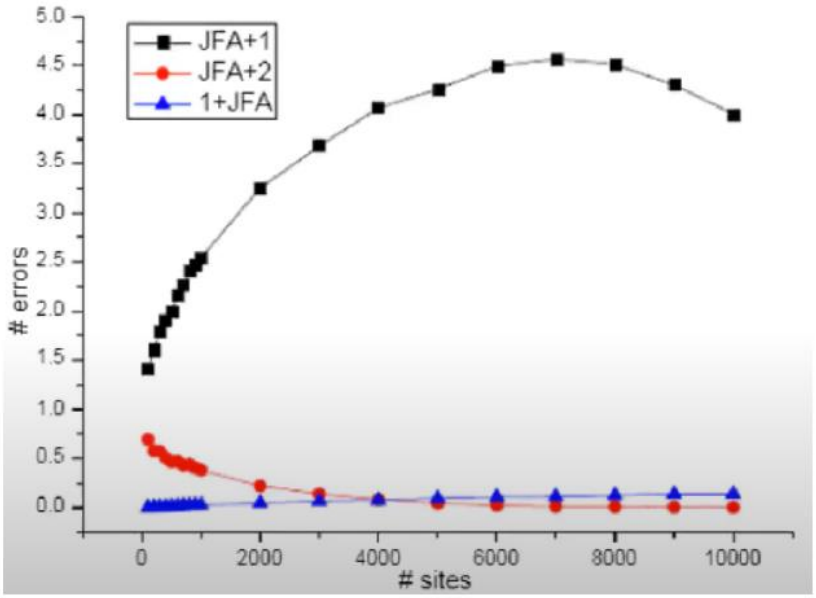


4.2. Napake in različice algoritma

Pri jump flood algoritmu uporabljamo skok K in tako hitreje barvamo celotno ravnino, vendar pa zaradi teh skokov pride tudi do napak. V večini primerov je njihovo število zanemarljivo, saj imamo ogromno število točk in semen. Če se v takih primerih pojavi kakšna napačno označena točka, to ne igra bistvene vloge. Kljub temu pa obstajajo načini, da število napak zmanjšamo. Vse nadgradnje algoritma dodajo nek korak med naše osnovne korake K in tako poskrbijo, da se določena območja natančneje pregledajo. Poglejmo si tri nadgradnje in korake, ki jih dodamo:

- **JFA + 1** : $K \in \frac{N}{2}, \frac{N}{4}, \dots, 1, 1$
- **JFA +2** : $K \in \frac{N}{2}, \frac{N}{4}, \dots, 1, 2, 1$
- **1 + JFA** : $K \in 1, \frac{N}{2}, \frac{N}{4}, \dots, 1$

Opazimo, da algoritmi dodajo korak, ki pregleda najbližje sosede ponovno in s tem odpravijo napake, ki ležijo ob meji dveh različnih območij. V naslednjem grafu je razvidno, kako se število napak spreminja glede na število semen.



5. Delaunay-jeva triangulacija

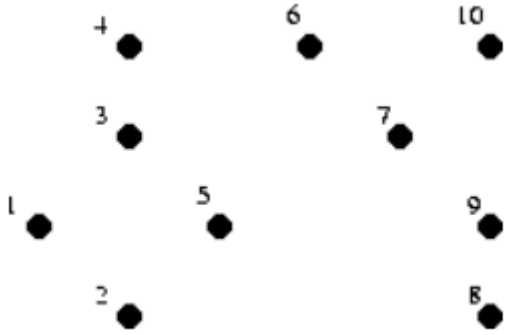
Še hitrejša in bolj učinkovita metoda za pridobivanje Voronoijevih diagramov pa je Delaunay-eva triangulacija. V nadaljevanju si bomo ogledali algoritem, ki iz danih točk ustvari triangulacijo. Sledi še kako potem iz dobljene triangulacije pridemo do naših Voronoijevih diagramov.

5.1. Definicija

V ravnini imamo točke A, B, C in D. Poglejmo dva trikotnika ΔABD in ΔBCD, ki imata skupno stranico BD. Trikotnika skupaj sestavljata štirikotnik, katerega razpolavlja stranica BD. Delaunay-ova triangulacija velja, če nasprotna kota nista presekana z stranico BD in skupaj tvorita kot, ki je manjši od 180°. Pravilo mora veljati za vse pare trikotnikov v množici, ki imajo skupno stranico.

5.2. Algoritem

Algoritem deluje po načinu deli in vladaj, kar pomeni, da osnovni problem razdelimo na manjše pod probleme in jih na koncu rekurzivno združimo v celotno rešitev. Zdaj pa si poglejmo algoritem po korakih. Vse korake bomo demonstrirali na spodnjem grafu:



5.2.1. Urejanje točk

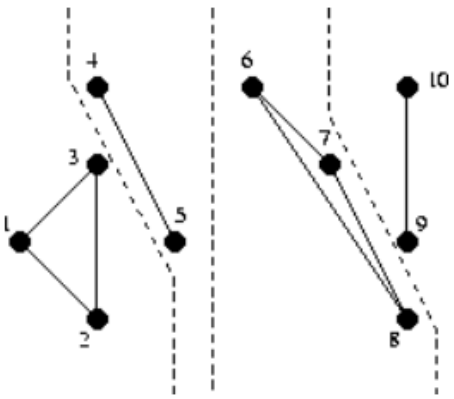
Pred začetkom algoritma, je potrebno točke urediti na naslednji način. Uredimo jih po njihovi x koordinati. V primeru, da ima več točk enako x koordinato, jih uredimo še po y koordinati. Točke na zgornji sliki so že pravilno urejene.

5.2.2. Rekurzivno deljenje

Zdaj množico točk rekurzivno delimo na dele oziroma celice, ki vsebujejo do 3 točke.

To dobimo tako, da med dvema sredinskima točkama naredimo mejo. Postopek ponavljamo dokler nam ne ostanejo celice, kjer se pojavita najmanj 2 točki ali največ 3 točke. Ko dosežemo takšno stanje v vsaki celici povežemo točke med sabo. Tako bomo v celici z dvema točkama dobili daljico, z tremi točkami pa trikotnik.

Stanje po končanem rekurzivnem deljenju na zgornjem grafu:



5.2.3. Rekurzivno lepljenje

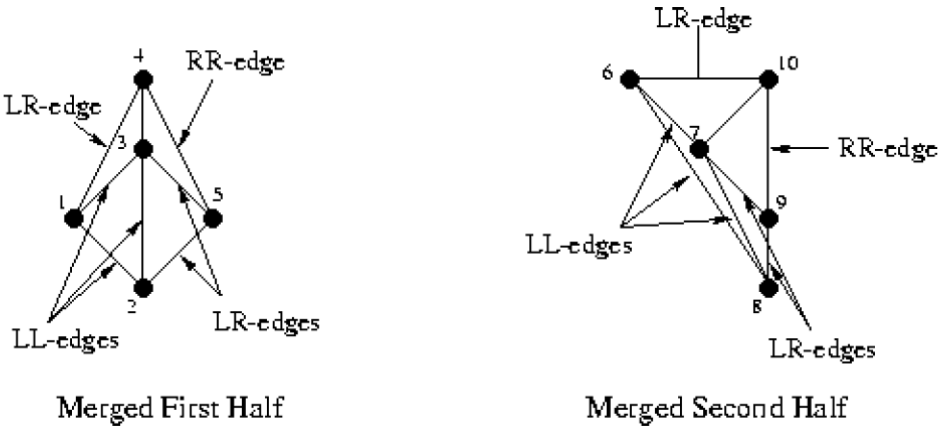
Sledi najbolj kompleksni del algoritma, saj bomo celice ponovno spajali skupaj tako, da bomo ohranjali Delaunay-evo triangulacijo. Ustvarjali bomo nove povezave, ki bodo zadoščale pravilom triangulacije in tvorile nove trikotnike. Rekurzivno lepimo celice po istem zaporedju, kot smo jih delili.

Spajanje

Torej spajamo dve celici, ki sta si sosednji. Imamo levo in desno celico in tukaj bomo zaradi lažjega razumevanja definirali izraze za skupine povezav:

- LL : Vse povezave, ki se začnejo in končajo v levi celici
- RR : Vse povezave, ki se začnejo in končajo v desni celici
- LR : Nove povezave, ki nastajajo med spajanjem leve in desne celice.

Pri spajanju se lahko zgodi primer, ko bomo izbrisali kakšno povezavo v LL ali RR. Nikoli pa ne bomo ustvarili kakšne nove LL ali RR povezave. Če na zgornjem grafu naredimo prvi korak spajanja dobimo takšen rezultat:

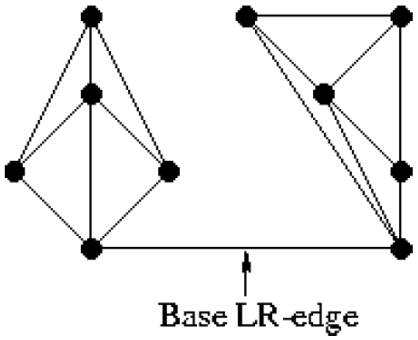


Merged First Half Merged Second Half

Pri zgornjem primeru, je bilo prvo spajanje zelo preprosto, saj smo samo ustvarili vse možne povezave med dvema celicama. Zato si raje poglejmo podrobneje naslednji korak, kjer spajamo nastali celici.

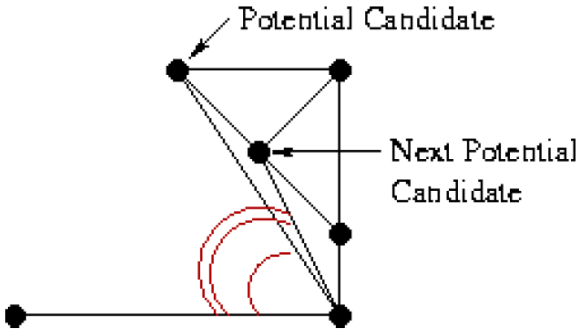
Ustvarjanje LR povezave.

Bazna LR povezava je povezava, ki povezuje točki v levi in desni celici, le te imata najmanjši y koordinati, kar pomeni, da je povezava najnižja. Prav tako povezava ne sme sekati nobene LL ali RR povezave.



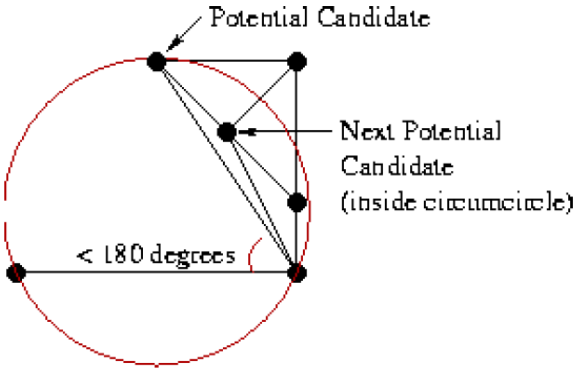
Ko imamo bazno LR povezavo, se premikamo navzgor in določamo naslednjo LR povezavo, ki bo nad bazno. Pri tem poskrbimo, da bo novo nastala povezava zagotovo vsebovala eno od dveh točk, ki sta v bazni LR povezavi, druga točka, pa mora biti na drugi strani.

Izbiro si olajšamo tako, da izberemo dve točki kandidatki. Ena je iz leve, druga pa iz desne celice. Če si kot primer pogledamo desno stran. Kandidatke izbiramo zaporedoma po velikosti kota, ki je med LR bazno povezavo in povezavo desne točke z desno kandidatko.



Za vsako kandidatko preverimo, če zadostuje pogojem:

I. Kot med bazno LR povezavo in povezavo desne točke do kandidatke mora biti manj kot 180°.
II. Očrtana krožnica trikotnika, ki jo tvorita obe točki bazne LR povezave in kandidatka, ne sme vsebovati naslednje potencialne kandidatke.



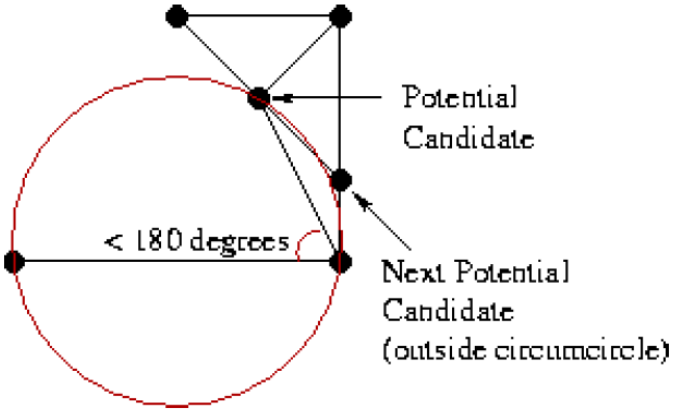
Preverimo zgornje pogoje in ugotovimo, da kandidatka zadostuje prvemu pogoju drugemu pa ne. Pri temu postopku, se lahko srečamo z tremi možnostmi:

- **I. in II. sta izpolnjena**: Kandidatka postane naša končna kandidatka oziroma izbranka
- **I. ni izpolnjen**: Ne izberemo nobene kandidatke na desni strani
- **I. izpolnjen II. pa ne**: Izbrišemo povezavo RR, ki jo tvori desna točna bazne LR povezave in kandidatka.

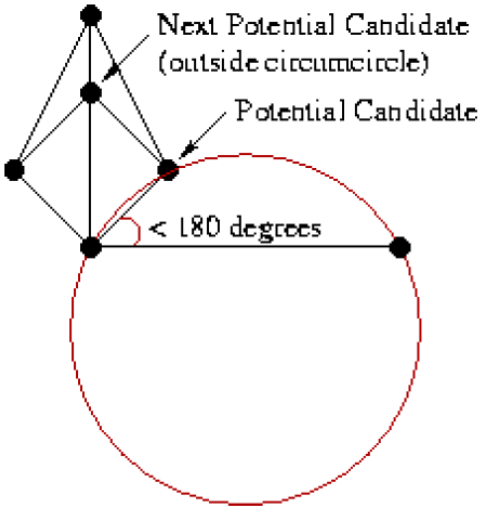
Zgornji postopek izvajamo na vseh kandidatnih točkah, dokler:

1. Ne dobimo končne kandidatke oziroma izbranke
2. Pregledamo vse kandidatke in ugotovimo, da nobena ni ustrezna.

Ko to naredimo na našem primeru, izgleda nov graf sledeče:



Postopek ponovimo tudi na levi strani in se držimo zgoraj naštetih pravil:



Ob končanem postopku določanja kandidatk na obeh straneh, lahko pridemo do treh zaključkov:

1. **Na nobeni strani nismo našli izbranke:** Zaključimo, ker je spajanje končano
2. **Dobimo izbranko le na eni strani:** Naredimo novo povezavo med izbranko in pa točko bazne LR povezave, ki je na drugi strani kot izbrana točka.
3. **Dobimo dve izbranki:** V tem primeru imamo dodatno preverjanje, da ugotovimo, katera izbranka je boljša.

Preverjanje obeh izbrank

Očrtamo krog trikotniku, sestavljenemu iz točk bazne LR povezave in pa leve izbranke. Če krog ne vsebuje desne izbranke, potem leva izbrana točka določa novo povezavo med njo in desno točko bazne LR povezave. Nasprotno velja za levo izbranko.

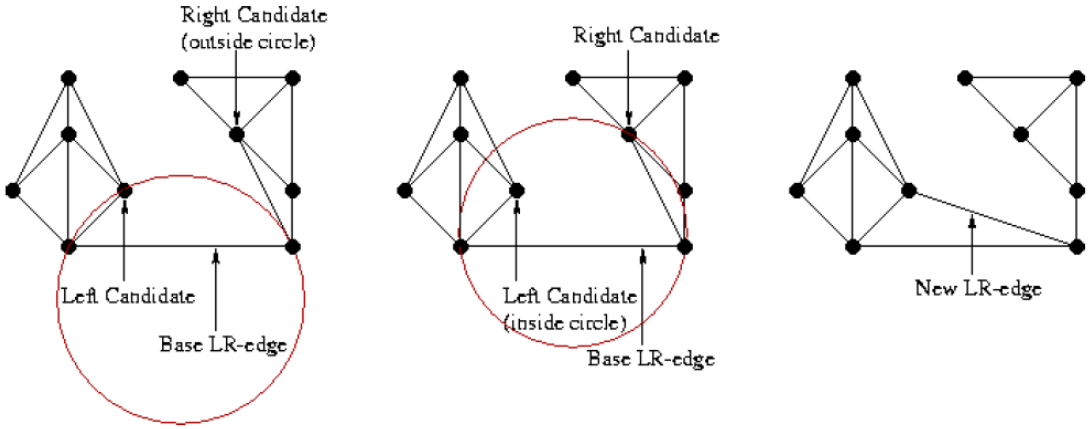
Pri preizkusu izbrancev, se vprašamo, če se lahko zgodi, da noben pogoj ne velja. Na to odgovorita naslednja izreka:

Izrek
Po zagotovljenem obstoju Delaunay-eve triangulacije, bo vsaj eden zadostoval zgornjem pogoju.

Izrek
Po unikatnosti Delaunay-eve triangulacije, bo natanko eden zadostoval pogoju. (Izjema so, če so vse 4 točke ko-planarne.)

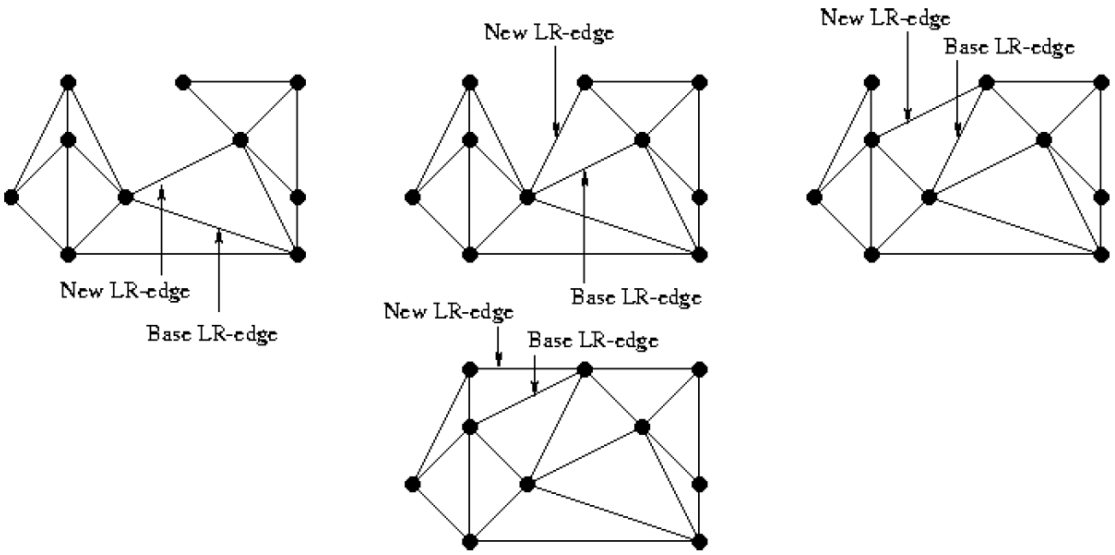
Dokaza izrekov zaradi kompleksnosti izpustimo.

Zdaj si poglejmo preverjanje kandidatov na našem grafu:



Iz slike je razvidno, da ko vzamemo levo izbranko, dobimo očrtan krog, ki ne vsebuje desne izbranke in je zato leva izbrana točka tista, ki določa novo povezavo. Ta povezava je zdaj naša nova bazna LR povezava. Postopek nato ponavljamo, dokler ne pridemo do bazne LR povezave, kjer ne bomo našli kandidatk.

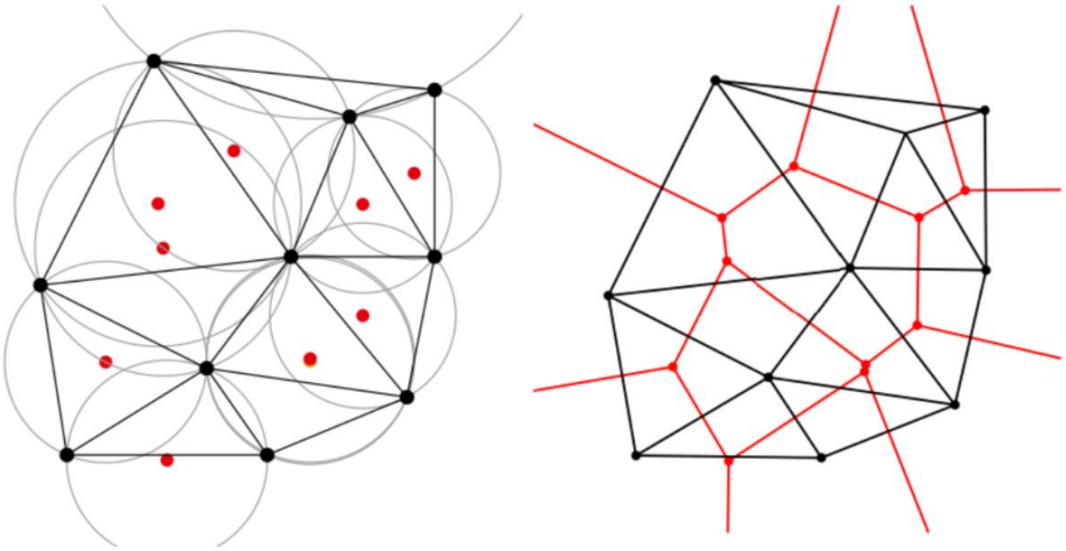
Vse LR povezave, ki jih dodamo v našem grafu pa so sledeče:



Zadnja slika je tudi končna Delaunay-eva triangulacija našega grafa.

5.3. Povezovanje točk

Z zgornjim algoritmom pridemo do končne Delaunay-eve triangulacije. Zdaj pa si poglejmo še postopek, kako iz triangulacije dobimo Voronoijev diagram. Poznamo več postopkov, saj je malce odvisno od implementacije algoritma za pridobivanje Delaunay-eve triangulacije. Ogledali si bomo dva, ki za rešitev uporabljata krožnice, katere dobimo med ustvarjanjem triangulacije in si jih lahko že med samim algoritmom shranjujemo. Najprej pa si za lažjo predstavo oglejmo sliko, kjer je na levi končna Delaunay-eva triangulacija z vsemi orisanimi krožnicami. Na desni strani pa imamo Voronoiev diagram, ki ga dobimo ko povežemo vse točke.



5.3.1. Sosednji trikotniki

Pri tem načinu, si med samo triangulacijo sproti shranjujemo krožnice in trikotnike, ki so nastali med postopkom. Potem se lotimo povezovanja.

Začnemo s poljubnim trikotnikom in poiščemo vse sosednje trikotnike, ki imajo eno stranico isto kot izbrani. Najdemo lahko največ tri trikotnike, lahko pa tudi manj. Središče očrtane krožnice izbranega trikotnika povežemo z vsemi središči očrtanih krožnic sosednjih trikotnikov. V primeru, da izbranemu trikotniku najdemo manj kot tri sosede, bomo tisti stranici, kjer soseda ni, narisali simetralo, ki bo na eni strani potekala do središča očrtane krožnice, na drugi strani pa do roba ravnine.

Postopek ponavljamo za vse trikotnike in na koncu nove povezave tvorijo Voronoijev diagram.

5.3.2. Simetrale stranic

Pri drugem načinu ne potrebujemo shranjenih nobenih trikotnikov. Vse kar potrebujemo so končne povezave triangulacije in pa središča očrtanih krožnic. Sprehodimo se čez vse povezave in vsaki narišemo simetralo stranice. Vsaka simetrala bo sekala eno ali dve središči krožnic, zato postopek ločimo na dve možnosti:

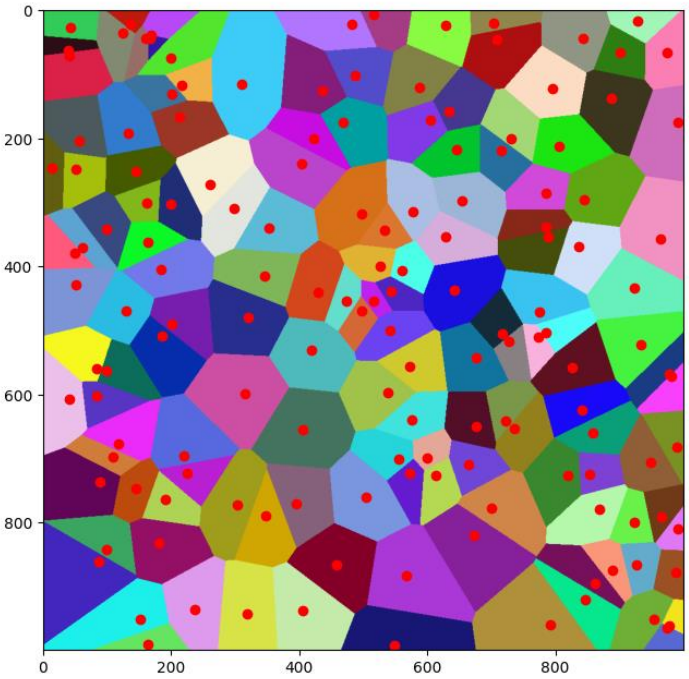
- **Simetrala seka dve središči:** Ohranimo samo del simetrale med obema točkama
- **Simetrala seka eno središče**: Ohranimo del simetrale, ki je med središčem in povezave, kateri simetrala pripada. Na drugi strani povezave ohranimo celotno simetralo, ki poteka od povezave pa vse do konca ravnine.

Ko postopek ponovimo za vse povezave, bodo vse ohranjene simetrale tvorile Voronoiev diagram.

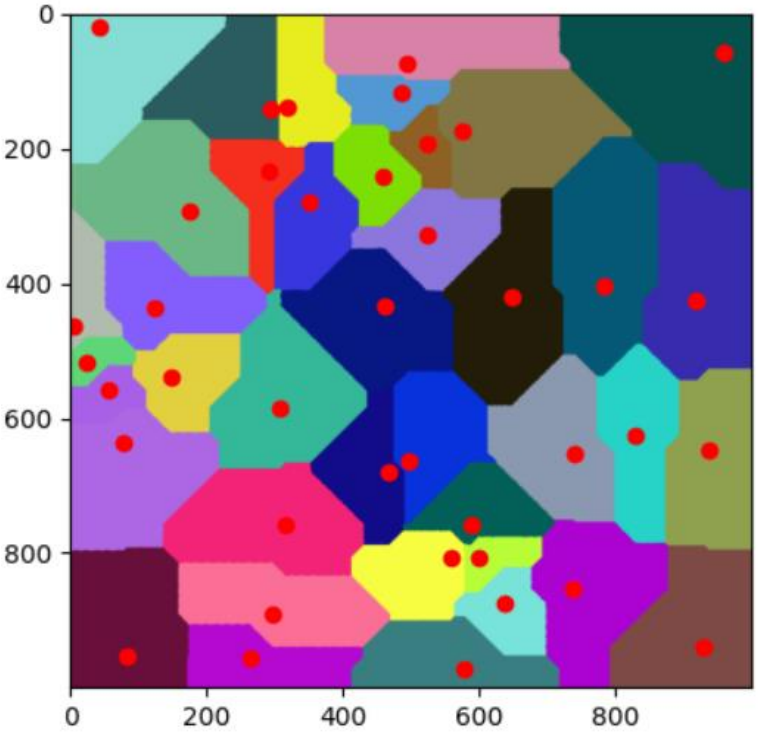
6. Različne metrike

Kot zanimivost si poglejmo še par zgledov Voronoijevih diagramov v različnih metrikah:

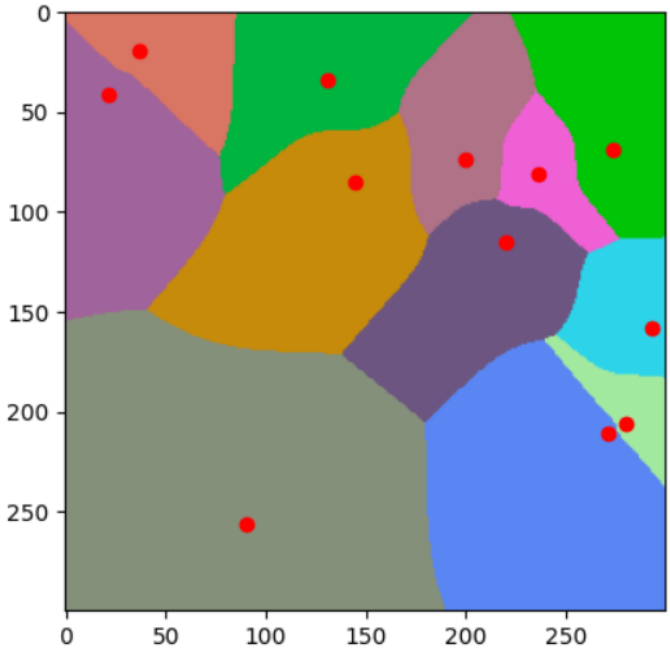
Evklidska metrika $d(A,B) = \sqrt{|a_1 - b_1|^2 + |a_2 - b_2|^2}$



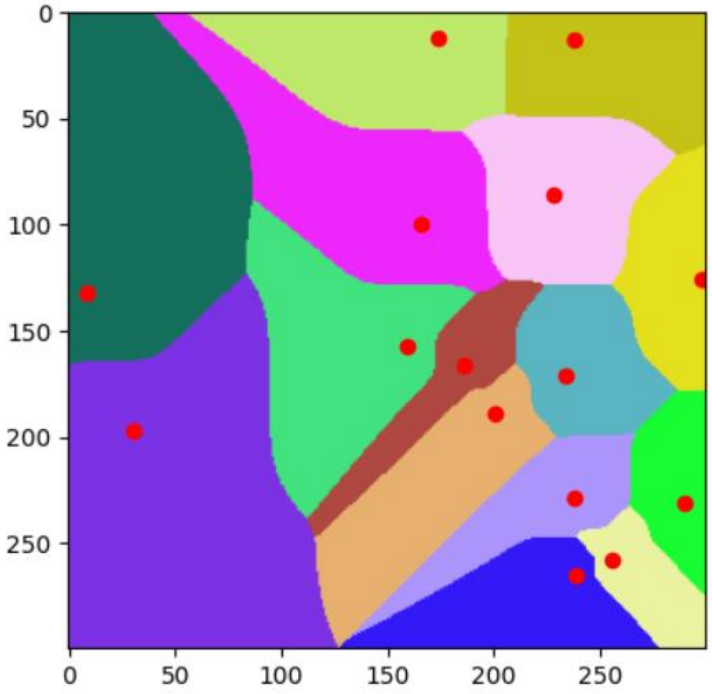
Manhattanska metrika $d(A,B) = |a_1 - b_1| + |a_2 - b_2|$



K4 metrika $d(A,B) = \sqrt[4]{|a_1 - b_1|^4 + |a_2 - b_2|^4}$



K6 metrika $d(A,B) = \sqrt[6]{|a_1 - b_1|^6 + |a_2 - b_2|^6}$



7. Primeri uporabe

Voronoievi diagrami se uporabljajo v zelo različnih oblikah in so zelo razširjeni. Nekatere uporabe je preprosto razumeti, saj takoj vidimo, kako bi jih uporabili, medtem ko so nekatere uporabe zelo kompleksne in jih je težko razumeti. Zato si raje oglejmo nekaj enostavnih:

Načrtovanje rasti in razvoja gozdov

V primeru, da na več mestih zasadimo gozdove, lahko z pomočjo Voronoievih diagramov ugotovimo, kje se bodo gozdovi združili in kako bo na koncu izgledal skupni gozd. Seveda, nekatere vrste se širijo hitreje kot druge, zato lahko v Voronoijevih diagramih dodamo tudi nekakšne uteži, ki napovejo, kako hitro se širi nek del gozda. To se potem pri računanju razdalj upošteva in lahko vseeno dobimo pravilne izračune.

Načrtovanje poti v robotiki

Če pomislimo, da so vsi predmeti v nekem prostoru semena za naš Voronoiev diagram. Končne povezave, ki tvorijo diagram nam opišejo poti, ki so kar se da oddaljene od vseh ovir v prostoru. Če imamo robota, ki bo sledil tem potem, se bo kar se da na široko izognil vsem predmetom, kar zmanjša možnost za trk.

Najbližja ustanova

Predstavljamo si, da imamo v nekem mestu več policijskih postaj. Da bi vsaka postaja imela približno enako dela, si lahko z Voronoievimi diagrami pomagamo razdeliti mesto na enaka območja okoli postaj in bo tako delo razporejeno, kar se da pravično.

Resnična zgodba prihaja iz mesta Melbourne, kjer ima vsak otrok po zakonu zastonj šolanje v ustanovi, ki je najbližje njegovemu prebivališču. Tam so z pomočjo Voronoievih diagramov razdelili mesto in objavili zemljevid, da so lahko starši ugotovili, v katero šolo naj vpišejo otroka.

Iskanje najbližjega telefonskega oddajnika

Imamo mobilni telefon in želimo kar se da dober telefonski signal. Ena možnost je, da telefon preveri moč signala do vseh oddajnikov, s katerimi ima trenutno stik. Lahko pa ima že v naprej določen zemljevid, kjer so označeni oddajniki in tudi narejen Voronoiev diagram. Potem lahko telefon na podlagi lokacije ugotovi, z katerim oddajnikom naj komunicira, saj mu je ta najbližje oziroma ima za to lokacijo najboljši signal.

8. Viri

- Voronoi diagram. https://en.wikipedia.org/wiki/Voronoi_diagram(Dostopno 2.9.2022)

- Jump flooding algorithm. https://en.wikipedia.org/wiki/Jump_flooding_algorithm(Dostopno 2.9.2022)

- Peterson, S. *Computing constrained Delaunay triangulations*. http://www.geom.uiuc.edu/~samuelp/del_project.html#problem(Dostopno 2.9.2022)

- Delaunay triangulation. https://en.wikipedia.org/wiki/Delaunay_triangulation(Dostopno 2.9.2022)