 

Kruskal algoritmas minimaliam grafo karkasui rasti

Lukas Karmanovas

Matematikos ir informatikos fakultetas

Gegužės 17d., 2016

Turinys

[1 Įvadas 3](#_Toc451237340)

[1.1 Užduotis 3](#_Toc451237341)

[1.2 Užduoties aprašymas 3](#_Toc451237342)

[2 Algoritmų kūrimas 3](#_Toc451237343)

[2.1 Duomenų struktūros 4](#_Toc451237344)

[2.2 Kruskal algoritmas su komponenčių masyvu 4](#_Toc451237345)

[2.3 Kruskal algoritmas naudojant medžius 5](#_Toc451237346)

[3 Algoritmų analizė 6](#_Toc451237347)

# Įvadas

## Užduotis[[1]](#footnote-1)

**Duota:** Neorientuotas svorinis grafas *G*, turintis *n* viršūnių ir *m* briaunų su svoriais *w(e*).

**Rasti:** Grafo *G* karkasą (Spanning tree, angl.) *T*, kurio svoris būtų mažiausias iš visų galimų grafo *G* karkasų.

## Užduoties aprašymas

Kruskal algoritmas yra minimalaus grafo karkaso radimo algoritmas, kuris suranda briaunas mažiausio įmanomo svorio kurios jungia medžius miške.

**Apibrėžimas 1:** Duotas neorientuotas grafas *G = (V,E)*, be ciklis pogrfas grafo *G* yra vadinamas **mišku**, ir sujungtas beciklis pografis *G* yra vadinamas **medžiu**.

**Apibrėžimas 2:** Duotas neorientuotas grafas *G = (V,E)*, pografis *T* yra vadinamas grafo *G* karkasu, jeigu pografis *T* yra sujungtas, be ciklas ir jungia visas grafo *G* viršūnes.

**Apibrėžimas 3:** Duotas neorientuotas grafas *G = (V,E)*, ir teigiama svorio funkcija , tada *T* yra minimalus grafo *G* karkasas jeigu *T* yra grafo *G* karkasas ir yra mažiausias iš grafo *G* karkasų.

Kruskal algoritmas yra godus algoritmas grafų teorijoje, suranda minimalaus grafo karkaso sujungtame svoriniame grafe didinant verte kiekviename žingsnyje. Kitais žodžiais sakant jis suranda pografį briaunų, iš kurių suformuotas medis įskaitant visas viršūnes, kur bendras svoris visų medžio briaunų yra minimalus. Jeigu grafas yra nesujungtas tada yra surandamas minimalus miško karkasas (minimalus grafo karkasas kiekvienai sujungtai komponentei).

Šis algoritmas pirmą kartą pasirodė [*Proceedings of the American Mathematical Society*](https://en.wikipedia.org/wiki/Proceedings_of_the_American_Mathematical_Society), 1956 metais ir buvo parašytas Džiozefo Kruskalo[[2]](#footnote-2)

# Algoritmų kūrimas

Mūsų tikslas yra sukurti Kruskal algoritmą, kuris galėtų surasti bet kokio svorinio grafo *G* minimalų grafo karkasą. Šiam tikslui mes naudosime dvi algoritmų kūrimo strategijas:

* Su komponenčių masyvu (viršūnių žymėmis)[[3]](#footnote-3)
* Naudojant medžius (sąrašus)[[4]](#footnote-4)

## Duomenų struktūros

Abiem algoritmams naudosime neorientuotą svorinį grafą *G*, kuris yra sudarytas iš:

* Viršūnių;
* Briaunų;

Paduodami duomenys yra iš tekstinio failo kuriame yra sugeneruotos viršūnės su briaunomis ir svoriais. Kiekviena duomenų eilutė yra sudaryta iš pradinio taško (taškas iš kurio išeina briauna), galinio taško (taškas į kurį ateina briauna) ir briaunos svorio. Pirmose dviejuose eilutėse yra viršūnių skaičius ir briaunų skaičius.

## Programos paleidimas

Turime keturis programos failus, trys yra programos kodai ir vienas tekstinis duomenų failas. Pradėkime nuo lengviausio tai yra Generatoriaus klasė, kuri yra skirta sukurti tekstinį failą su duomenimis. Ji sukuria pirmoje eilutėje Viršūnių skaičių, antroje eilutėje gaunamas Briaunų skaičius. Likusiose eilutėse mes turime po tris elementus pirmi du yra Viršūnės, o paskutinis yra double tipo Svoris. Generator.java sukuria NewFile.txt kuriame visi duomenys yra.

Antras programinis failas yra AAlgorithm.java jis skaičiuoja a variantą Su komponenčių masyvu. Programa atidaro NewFile.txt ir iš jo pasiima duomenis, taigi nereikia vartotojui nieko įvedinėti užtenka tiesiog paleisti programą.

Trečias programinis failas yra Kruskal.java jis skirtas apskaičiuoti b variantą Naudojant medžius. Programa atidaro NewFile.txt ir iš jo pasiima duomenis, taigi nereikia vartotojui nieko įvesti užtenka tiesiog paleisti programą.

**SVARBU**: programos turi būti paleidžiamos iš komandinės eilutės (jeigu paleidžiami tiesiai jar failai). Pirmiausia paleidžiame Generator.jar (įvedame į komandinę eiutę: java -jar Generator.jar) suvedame reikiamus duomenis (programa pati parašys ko jai reikia). Tada kataloge programos turi susikurti NewFile.txt su duomenims. Dabar galima paleisti vieną iš algoritmų a) Su komponenčių masyvu (java -jar AAlgorithm.jar); b) Naudojant medžius (java -jar Krustal.jar).

## Kruskal algoritmas su komponenčių masyvu

Svorinis grafas *G* yra grafas, kur kiekviena briauna *e* turi susijusį realų skaičių pavadinimu svoris. Minimalus grafo karkasas svorinio grafo yra grafo karkasas turintis mažiausią įmanomą bendrą svorį. Šį algoritmą galima apibrėžti tokiais šešiais žingsniais:

1. Turimas sujungtas grafas *G*, kurio visos viršūnės yra *v1, v2, ..., vn*.
2. Sąrašas visų viršūnių yra išdėliotas didėjimo tvarka pagal svorius.
3. Pridedama etiketė *L(i)* prie kiekvienos viršūnės *vi* ir inicializuojama *L(i)* į atskiras reikšmes nustatant *L(i) = i (i = 1, 2, ..., n)*
4. Tegu rezultatų grafas *T* būna inicializacijos metu tuščias.
5. Paimti sekančią laisvą briauną *e* iš briaunų sąrašo iš 2) dalies. Tarkime *vi0* ir *vj0* yra viršūnės briaunos *e*.
   1. IF , tada pridedame briauną *e* į *T* ir tada pakeičiame visus *L(k)*, kurie yra lygus *L(i0)* arba *L(j0)* į mažiausią reikšmę *min(L(i0)*, *L(j0))*.
   2. IF , praleidžiame briauną *e*.
6. Kartojame 5) punktą tol kol būsime perėję per visas briaunas.

Algoritmo implementacija Java kalba:

**void** Kruskal(){  
 Edge result[] = **new** Edge[**V**];  
 **int** labels[] = **new int**[**V**];  
 **int** i, resultIncrease = 0;  
 **for** (i = 0; i < **V**; i++){labels[i] = i+1;}  
 quicksort(**edgeArray**, 0, **E**-1);  
 **for** (i = 0; i < **E**; i++){  
 **int** labelStart = labels[**edgeArray**[i].**start**];  
 **int** labelDestination = labels[**edgeArray**[i].**destination**];  
 **if** (labelStart != labelDestination){  
 result[resultIncrease] = **edgeArray**[i];  
 resultIncrease += 1;  
 **if**(labelStart < labelDestination){  
 labels[**edgeArray**[i].**destination**] = labelStart;  
 **for**(**int** j = 0; j < **V**; j++){  
 **if**(labels[j] == labelDestination){labels[j] = labelStart;}  
 }  
 }  
 **else**{  
 labels[**edgeArray**[i].**start**] = labelDestination;  
 **for**(**int** j = 0; j < **V**; j++){  
 **if**(labels[j] == labelStart){labels[j] = labelDestination;}  
 }  
 }  
 }  
 }  
 **double** fullWeight = 0;  
 **for** (i = 0; i < resultIncrease; i++){  
 fullWeight += result[i].**weight**;  
 }  
 System.***out***.println(**"Weight full "** + fullWeight);  
}

Išrykiavimo algoritmas užtrunka *O(E2)*. Trečias žingsnis užiema *O(V log V)*. Taigi bendrai algoritma užiema *O((E2 + V) log V)*.

## Kruskal algoritmas naudojant medžius

Algoritmas:

1. Išrykiuoti visas briaunas iš grafo *G* svorių didėjimo tvarka
2. For kiekvienai briaunai *ei* = (u, v), i = 1,2, ..., m do

S1 = findSet(u)

S2 = findSet(v)

If S1 = S2 then

Do nothing

Else

Union(S1, S2)

Endif

**void** KruskalAlgorithm(){  
 Edge result[] = **new** Edge[**V**];  
 **int** i;  
 **int** e = 0;  
 **for**(i = 0; i < **V**; i++){  
 result[i] = **new** Edge();  
 }  
 quicksort(**edge**, 0, **E** - 1);  
 Vertex[] vertexSet = **new** Vertex[**V**];  
 **for** (i = 0; i < **V**; i++) {  
 vertexSet[i] = **new** Vertex();  
 vertexSet[i].**parent** = i;  
 vertexSet[i].**rank** = 0;  
 }  
 i = 0;  
 **while**(e < **V** - 1){  
 Edge nextEdge;  
 nextEdge = **edge**[i++];  
 **int** u = findSet(vertexSet, nextEdge.**start**);  
 **int** v = findSet(vertexSet, nextEdge.**destination**);  
 **if** (u != v){  
 result[e++] = nextEdge;  
 Union(vertexSet, u, v);  
 }  
 }  
 **double** fullWeight = 0;  
 **for** (i = 0; i < e; i++) {  
*// System.out.println(result[i].start + " - " + result[i].destination+  
// " weight " + result[i].weight);* fullWeight += result[i].**weight**;  
 }  
 System.***out***.println(**"Result weight "** + fullWeight);  
 }

Algoritmo sudėtingumas yra *O(E2 log E)* arba *O(E2 log V)*

# Algoritmų analizė

Buvo atlikti 88 bandymai iš viso, su kiekvienu algoritmu po 44 bandymus. Pirmiausiai buvo sugeneruoti vienodi duomenys abiem algoritmams su tais pačiais duomenimis buvo atlikta po 11 bandymų. Buvo naudoti duomenys v =100, e = 400; v =2000, e = 8000; v =10000, e = 40000; v =100000, e = 400000; (laikas matuojamas milisekundėmis)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 100x400 | | | 2000x8000 | | | | 10000x40000 | | | | | 1000000x4000000 | | |
| B | A | | B | | A | | B | | | A | | B | A | |
| 46 | 45 | | 243 | | 236 | | 488 | | | 623 | | 1663 | 9910 | |
| 46 | 42 | | 223 | | 243 | | 429 | | | 668 | | 1652 | 9901 | |
| 37 | 59 | | 216 | | 266 | | 502 | | | 619 | | 1742 | 9826 | |
| 37 | 44 | | 212 | | 247 | | 440 | | | 595 | | 1709 | 10107 | |
| 48 | 49 | | 206 | | 256 | | 447 | | | 595 | | 1802 | 9896 | |
| 41 | 41 | | 202 | | 262 | | 447 | | | 616 | | 1653 | 9799 | |
| 48 | 37 | | 202 | | 267 | | 514 | | | 662 | | 1629 | 10344 | |
| 44 | 36 | | 248 | | 245 | | 359 | | | 592 | | 1667 | 9788 | |
| 43 | 50 | | 237 | | 257 | | 433 | | | 636 | | 1704 | 10541 | |
| 46 | 45 | | 191 | | 259 | | 432 | | | 594 | | 1593 | 10040 | |
| 46 | 43 | | 206 | | 240 | | 437 | | | 619 | | 1935 | 9967 | |
| Vidutinis laikas | | | | | | | | | | | | | | |
| 43,81818 | | 44,63636 | | 216,9091 | | 252,5455 | | 448 | 619,9091 | | 1704,455 | | | 10010,82 |

Taigi matome, kad Kruskal algoritmas su komponenčių masyvu su dideliais duomenimis daug lėčiau susidoroja negu Kruskal algoritmas naudojantis medžius. Tai matome tiek iš grafiko tiek iš abiejų algoritmų sudėtingumų.

# Išvados

Buvo atlikti 88 bandymai su skirtingais duomenimis, kiekvieną iš duomenų patikrinant po 11 kartų. Atlikus bandymus buvo suapvalintas laikas (ms). Pastebėta, kad b) algoritmas veikia greičiau su dideliais duomenis (dideliu skaičiumi viršūnių ir briaunų). B) algoritmas yra greitesnis dėl to, kad jame nėra pereidinėjama beveik 3 kartus per visus elementus. A) algoritme mes matome, kad pereiname pirmiausia per visas briaunas, o paskui per visas viršūnes, tuo tarpu b) algoritme pereiname vieną kartą per visas briaunas.

1. 10 užduotis - http://www.mif.vu.lt/~valdas/ALGORITMAI/Laboratorinis\_darbas2015/Uzduotys2015/ [↑](#footnote-ref-1)
2. [Joseph Kruskal](https://en.wikipedia.org/wiki/Joseph_Kruskal) [↑](#footnote-ref-2)
3. J.A. Bondy and U.S.R. Murty, Graph Theory with Applications, 5th edition, North-Holl and, Amsterdam, 1982, pp. 36—40. Ir T. Dalby, Discrete Mathematics, Lecture Notes, 2001, pp. 88—93. [↑](#footnote-ref-3)
4. T.H. Cormen,C.E. Leiserson and R.L. Rivest, Introduction to Algorithms, 2nd edition, MIT Press,Cambridge, MA, pp. 476--484, 425—431 (žr. Cormen.pdf) ir E. Cohen, Lecture 3, 1997. [↑](#footnote-ref-4)