Vilniaus Universitetas

Matematikos ir Informatikos Fakultetas
Informatikos Katedra
III kursas ? grupė

Vardas Pavardaitis

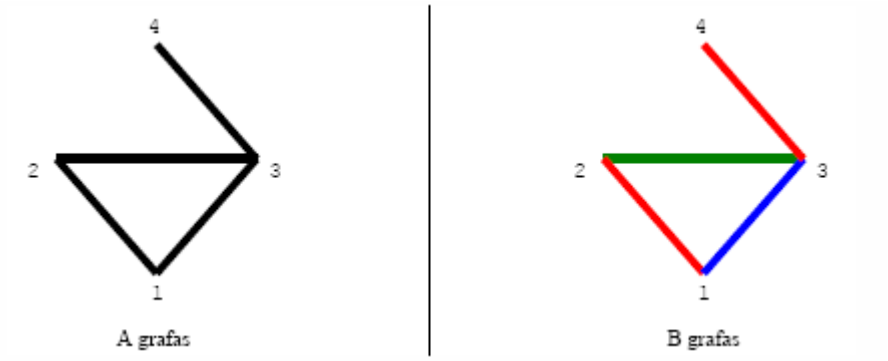
Užduotis 1-??:
Minimalaus briaunų spalvinimo uždavinys

2005

Užduotis: Duotam grafui rasti minimalų spalvų skaičių briaunoms nuspalvinti, taip kad dvi tos pačios spalvos briaunos neturėtų bendro taško.

Ši grafo briaunų spalvinimo problema dažnai taikoma aibės skaidymui į nesikertančias aibes, bangos kelio ilgio parinkimui optiniuose tinkluose bei tvarkaraščiams (planams) sudaryti. Pavyzdžiui, įsivaizduokime, kad mums reikia sudaryti asmenų susitikimų laiko grafiką. Susitikimo trukmė fiksuota. Taigi visi susitikimai galėtų įvykti skirtingu laiku, bet tai užtruktu daug laiko. Kad išvengti „laiko švaistymo“ pamėginkime sudaryti nesikertantį susitikimų tvarkaraštį, kurie galėtu vykti tuo pat metu (lygiagrečiai). Mes galime sukonstruoti tokį grafą, kurio viršūnės būtų žmonės, o briaunos vaizduotų žmonių porą kurie nori susitikti. Spalvų aibės vaizduos skirtingus laiko periodus tvarkaraštyje, o jų skaičius atitiks visų susitikimų trukmę (jei susitikimų trukmė vienoda).

Pavyzdys: Tarkime yra 4 asmenys: Jonas, Petras, Algis ir Kęstas, kurie turi susitikti.
Jonas turi pasitarti su Petru ir Algiu, o Algis su Petru ir Kęstu. Visa tai reikia padaryti per trumpiausią laiką (pokalbio trukmė - 1 valanda). Tegu grafo viršūnės vaizduos asmenis taip: Jonas → 1, Petras → 2, Algis → 3, Kęstas → 4. Grafo briaunos žymi asmenų poras. Šią situaciją vaizduoja A grafas.



Pamėginkime šias poras suskirstyti į kiek įmanoma mažiau aibių, tokias kad tuo pačiu metu tas pats asmuo negalėtų dalyvauti keliuose pokalbiuose. Aibes pažymėkime skirtingomis spalvomis.
9_valanda: {1 – 2, 3 – 4}, 10_valanda: {1 – 3}, 11_valanda: {3 – 2}. Šią situaciją vaizduoja B grafas.
Kiekvienais metais įvairių sporto šakų varžybų tvarkaraščiams sudaryti yra naudojamas grafo briaunų spalvinimo uždavinys.

Algoritmo kintamieji: Uždaviniui spręsti naudosim tokią duomenų struktūrą *Edge*, kuri apibrėžia grafo briauną. Ši struktūra turi du narius (*in* – įėjimo taškas, *out* – išėjimo taškas). Taip pat reikės dviejų dinaminių sąrašų: *edgelist* ir *colors*. Grafą faile vaizduosime briaunų masyvo pavidalu. Sąraše *edgelist* saugosime visas briaunas nuskaitytas iš įvedimo failo. O sąraše *colors* – briaunų sąrašus. Taigi uždavinio tikslas yra duotą briaunų sąrašą *edgelist* suskaidyti į nesikertančius sąrašus, ir patalpinti juos sąraše *colors*. Minimalus briaunų spalvų skaičius yra *colors* dydis t.y. *colors.size()*.

Spalvų paieškos algoritmas: Nuosekliai peržiūrime duota briaunų sarašą *edgelist* imdami po vieną briauną *i* iš šio sąrašo. Paieškos sutrumpinimui naudosime loginį požymį *found*, kuris rodys ar rastas briaunų sąrašas, kuriame galima patalpinti *i*-tają briauną. Kas kart imdami po naują briauną, požymį *found* padarom *false*. Tada tikriname briaunų spalvų sąrašus $colors::edgelist_j$. Jei briauna *i* turi bendrų taškų su esančiomis briaunomis sąraše $colors::edgelist_j$ tada tikriname

colors::edgelist$_{j+1}$. Jei peržiūrėjom visą *colors* ir neradom tokio briaunų sąrašo, kuriame būtu galima patalpinti *i*-tają briauną, tada prie *colors* prijungiame naują briaunų sąrašą su šia briauna. Priešingu atveju, briauną *i* prijungiame prie *colors.edgelist*$_j$, bei nustatome požymį *found* = *true*, ir nutraukiam tolimesnį *i*-tosios briaunos tikrinimą.

Briaunos taškų duotame briaunų sąraše paieškos algoritmas:

```
bool Graph::findEdgeNodes(Edge e, list <Edge> l) {
list <Edge>::iterator i;
for (i = l.begin(); i != l.end(); i++)
if ((i->in == e.in) || (i->out == e.out) ||
(i->in == e.out) || (i->out == e.in)) return true;
return false;
}
```

Minimalaus grafo briaunų spalvų skaičiaus paieškos algoritmas:

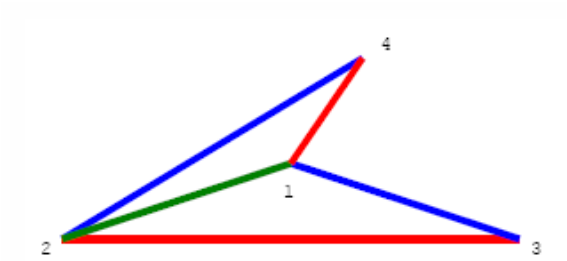
```
.........>
void Graph::calculateColors() {
list <list <Edge> >::iterator n;
list <Edge>::iterator i;
list <Edge> tmp;
bool found;
while (colors.size()) colors.pop_back();
for (i = edgelist.begin(); i != edgelist.end(); i++) {
found = false;
for (n = colors.begin(); n != colors.end(); n++)
if (!findEdgeNodes(*i, *n)) {
n->push_back(*i);
found = true;
break;
}
if (!found) {
tmp.push_back(*i);
colors.push_back(tmp);
tmp.pop_back();
}
}
}
```

Programos naudojimas, parametrai:

```
$ spalvos.exe
Grafo Briaunų Spalvinimas
naudojimas: spalvos.exe failas [-b] [-f] [-d] [-k n] [-g x y]
parametrai:
-b spausdinti briaunų aibes i <stdout>
-f spausdinti briaunų aibes i failą <failas.clr>
-d briaunų dublikatus laikyti skirtingais
-k n skaičiavimą kartoti n kartu
-g x y generuoti x viršūnių, y briaunų atsitiktini grafa į failą <failas>
```

Eksperimentai:

- **Grafas 0**

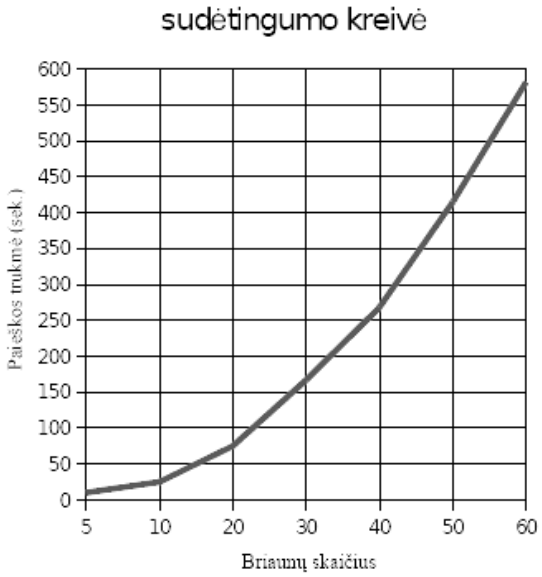


$ spalvos.exe test/graph0.edg -b -k 1000000
spalva #1:
1 -- 4 2 -- 3
spalva #2:
1 -- 3 4 -- 2
spalva #3:
1 -- 2
---
briaunų skaičius grafe : 5
minimalus briaunų spalvų sk.: 3
paieškos trukme : 10 sek.

- **Grafas 1**

$ spalvos.exe test/graph1.edg -b -k 1000000
spalva #1:
1 -- 5 2 -- 3 4 -- 7
spalva #2:
1 -- 3 7 -- 6 5 -- 2
spalva #3:
1 -- 4 6 -- 5
spalva #4:
1 -- 6 3 -- 4
---
briaunų skaičius grafe : 10
minimalus briaunų spalvų sk.: 4
paieškos trukmė : 25 sek.

Uždavinio sudėtingumas:



Apribojimai: C, N (spalvų skaičius) >=1

Iš sudėtingumo kreivės ir lygties matome, kad uždavinio sudėtingumas - eksponentinis. Sudėtingumo analizei buvo naudojami atsitiktinai sugeneruoti 10 viršūnių grafai. Kiekvienas grafo spalvų skaičiavimas buvo kartojamas 1000000 kartų (spalvos.exe failas -k 1000000).

Uždavinio sudėtingumas: $O(N^2 * C)$, kur N – briaunų, C – spalvų skaičius grafe.

Literatūra:

http://www.nada.kth.se/~viggo/wwwcompendium/node18.html
http://www.cs.sunysb.edu/~algorith/files/edge-coloring.shtml