Arbori minimi de acoperire

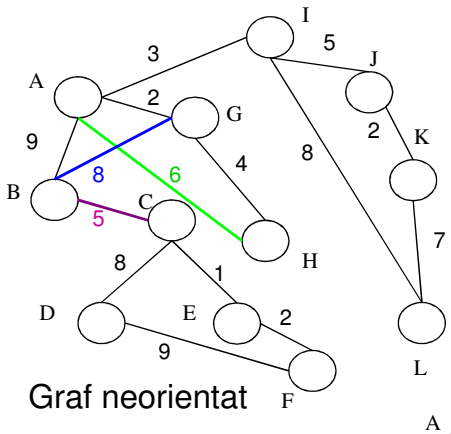
Planul cursului

- Arbori minimi de acoperire
  - Definitie
  - Utilizare
  - Algoritmi
- Operatii cu multimi disjuncte
  - Structuri de date pentru reprezentarea multimilor disjuncte
  - Algoritmi pentru reuniune si cautare
  - Calcul de complexitate

Arbori minimi de acoperire - Definitie si notatii

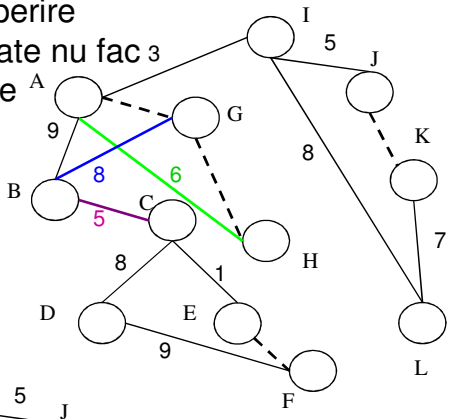
- G=(V,E) graf neorientat si conex;
- w:E->$\Re$ functie de cost
- w(u,v)= costul muchiei (u,v)
- Def. **arbore liber** al lui G este un graf neorientat conex si aciclic Arb=(V',E'); V'$\subseteq$V, E'$\subseteq$E; C(Arb)=Σw(e) e$\in$ E'
- Def. un arbore liber se numeste **arbore de acoperire** daca V'=V
- Def. un arbore de acoperire se numeste **arbore minim de acoperire** Arb$\in$ ARB(G) a.i. C(Arb)=min{C(Arb')|Arb'$\in$ ARB(G)}

Exemple

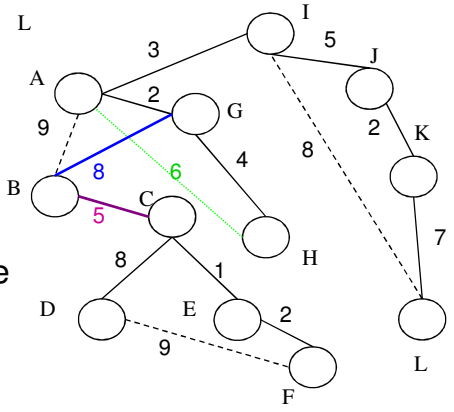


Graf neorientat

Arbore de acoperire
Muchiile punctate nu fac parte din arbore



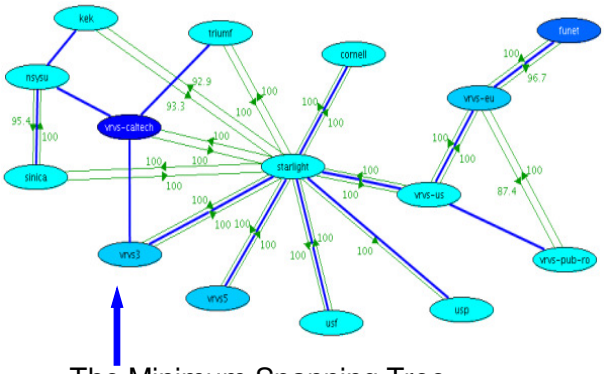
Arbore minim de acoperire
Muchiile punctate au fost eliminate din graf



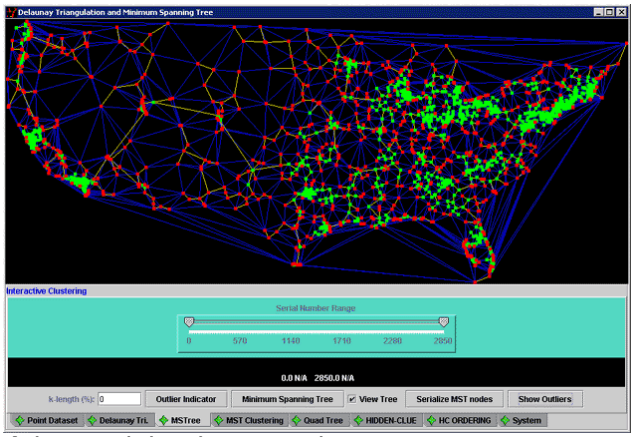
Utilizari

- Proiectarea retelelor
  - Electrice, calculatoare, drumuri
- Clustering
- Algoritmi de aproximare pentru probleme NP-complete

Exemple de utilizare



The Minimum Spanning Tree connections and the peer-to-peer connection quality for a set of VRVS reflectors (caltech) [1]



Arbore minim de acoperire pentru cca 2850 de orase din USA [2]

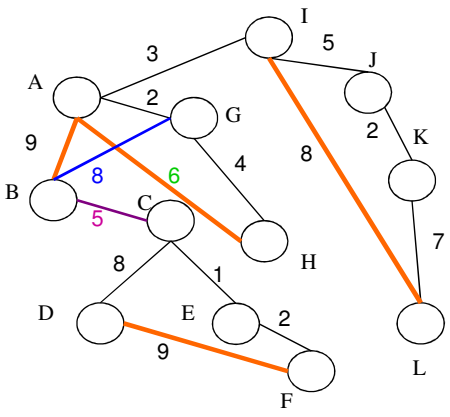
Construire AMA

- Greedy
- Se adaugă *arce sigure* până când nu mai există noduri neconectate

- Tăietura (T, N-T)
- Arc care travesrsează tăietura
- O tăietură ocolește o mulțime B
- Arc ieftin de traversare a unei tăieturi
- Teoremă – Un arc ieftin este sigur
- Corolar – G, B inclus într-un AMA, C componentă conexă în G_B – un arc ieftin ce conectează două componente conexe din G_B este sigur

Proprietati

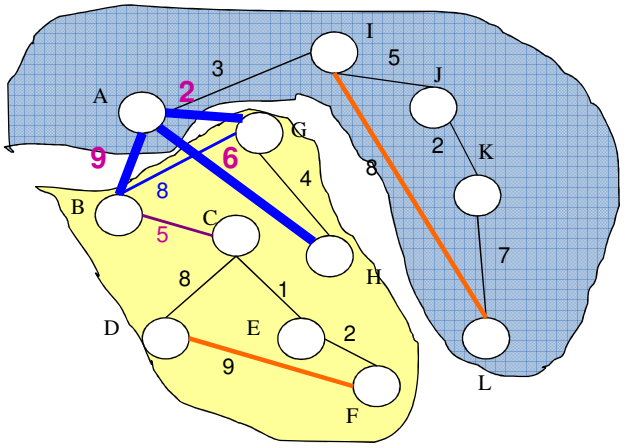
1. G=(V,E), C=(V',E') – ciclu in G; e∈ E' a.i. w(e) = max{w(e')|e'∈ E'}=> e∉ Arb(G) unde Arb(G)=arbore minim de acoperire in G.



•Pp e∈ Arb(G)
•Eliminand e din Arb(G)=>S1, S2 – 2 multimi de muchii
•e∈ E' (ciclu)=>∃e'∈ E' w(e)>w(e') a.i. un capat din e' este in S1 si celalalt in S2
•Arb(G)-e+e' = arbore de acoperire
•=>Cost(Arb(G)-w(e)+w(e')<Cost(Arb(G))=>Arb(G) nu este arbore minim

Proprietati

2. G=(V,E); S=(V',E'), V'⊂V; e=(u,v) a.i. e∉ E' dar (u xor v)∈ V' cu proprietatea ca w(u,v)=min{w(u',v')| (u' xor v')∈ V' } =>(u,v)∈ arbore minim de acoperire Arb

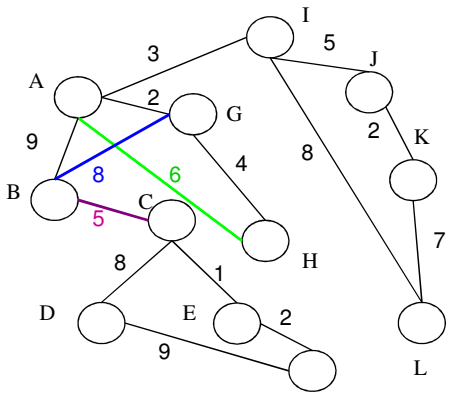


- Pp e∉ Arb(G)
- Arb' = Arb(G) – e' +e (unde e' o muchie similara cu e)
- Arb'= arbore de acoperire
- Cost(Arb')<Cost(Arb)
- => Arb nu este arbore minim

Algoritmul lui Kruskal

- Kruskal(G,w)
  - A=∅;
  - Foreach (v in V)
    - ConstrArb(v)
  - Sorteaza_asc(E,w)
  - Foreach((u,v) in E')
    - If(Arb(u)!=Arb(v))
      - Arb(u)=Arb(u)∪Arb(v)
      - A=A ∪{(u,v)}
  - Return A

Exemplu



- CE -1
- EF -2
- AG-2
- JK-2
- AJ-3
- GH-4
- BC-5
- IJ-5
- AH-6
- KL-7
- BG-8
- CD-8
- IL-8
- AB-9

Corectitudine (I)

- 1. aratam ca muchiile ignorate nu fac parte din Arb
  - Pp (u,v) a.i. Arb(u)=Arb(v)
    - =>(u,v) creeaza un ciclu in Arb(u) (arborii sunt aciclici)
    - w(u,v)=max{w(u',v')|(u',v')∈ Arb(u)} //din faptul ca muchiile sunt sortate ascendent
    - => (din (1)) (u,v)∉ Arb

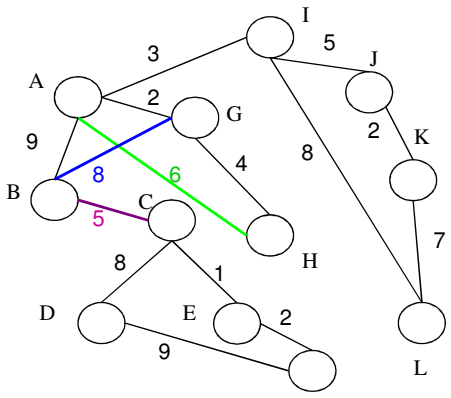
Corectitudine (II)

- 2. aratam ca muchiile pe care le adaugam apartin Arb
  - Arb(u)!=Arb(v)
  - =>(u,v) muchie cu un capat in Arb(u)
  - (u,v) are cel mai mic cost din muchiile cu un capat in u (din faptul ca muchiile sunt sortate crescator)
  - =>(u,v) $\in$ Arb (din (2))

Algoritmul lui Prim

- Prim(G,w,s)
  - A=∅
  - Foreach(u in V)
    - d[u]=∞; p[u]=null
  - d[s]=0;
  - Q=constrQ(V);
  - while(Q!=∅)
    - u=ExtrMin(Q);
    - A=A∪{(u,p[u])}
    - foreach(v∈ succs(u))
      - If(d[v]>w(u,v)
        - » d[v]=w(u,v);
        - » p[v]=u;
  - return A-{s,p(s)}

Exemplu



- IA - AJL
- AG - GBJL
- GH - BJLH
- IJ - BJLK
- JK - BLK
- KL – BL
- GB - BC
- BC – CDE
- CE – DE
- F- DF
- CD - D

Corectitudine

- S=(V',E') multimea varfurilor si muchiilor adaugate deja in arbore inainte de a adauga (u,p[u])
- p[u]$\in$ V', u$\notin$ V'; (u,p[u]) are cost minim dintre muchiile care au un capat in S (conform extrage minim)
- => (din (2)) (u,p[u])$\in$ Arb

Complexitate Prim

- Depinde de implementare (v. Dijkstra)
  - matrice de adiacenta $O(V^2)$
  - heap binar O(ElogV)
  - heap fibonacci O(VlogV+E)
- Concluzii
  - grafuri dese – matrice de adiacenta preferata
  - grafuri rare – heap binar sau fibonacci

Complexitate Kruskal

- elementele algoritmului
  - sortarea muchiilor O(ElogE)=O(ElogV)
  - Arb(u)=Arb(v) – compararea a 2 multimi disjuncte {1,2,3} {4,5,6} – mai precis trebuie identificat daca **2 elemente sunt in acelasi set?**
  - Arb(u)∪Arb(v) – reuniunea a **2 multimi disjuncte intr-una singura**
- => depinde de implementarea multimilor disjuncte

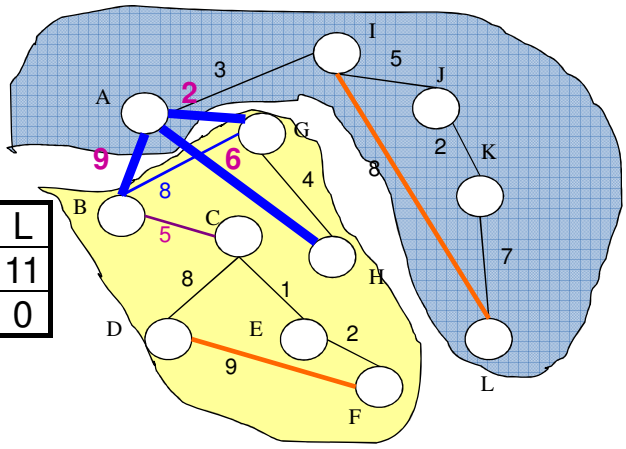
Variante de implementare multimi disjuncte (I) – contraexemplu

- varianta 1 (populara la laborator☹) – nerecomandata☺
- Multimile implementate ca vectori
- equal?(M1, M2)
  - foreach(u in M1)
    - foreach(v in M2)
      - if (u==v) return true
  - return false
- union
  - for(i=length(M1);i<length(M2)+length(M1);i++)
    - M1[i]=M2[i-length(M1)]
  - return M1
- equal? – complexitate V^2
- union – complexitate V
- numarul de apelari – M
- Complexitate totala $M*V^2$

Variante de implementare multimi disjuncte – regasire rapida

- Varianta 2
- M1- vector
- Id- vector de id-uri

A	B	C	D	E	F	G	H	I	J	K	L
0	1	2	3	4	5	6	7	8	9	10	11
0	1	1	1	1	1	1	1	0	0	0	0



- Arb(u)!=Arb(v)
  - O(1)
- Arb(u)=Arb(u)∪Arb(v)
  - O(n)

Variante de implementare multimi disjuncte – regasire rapida

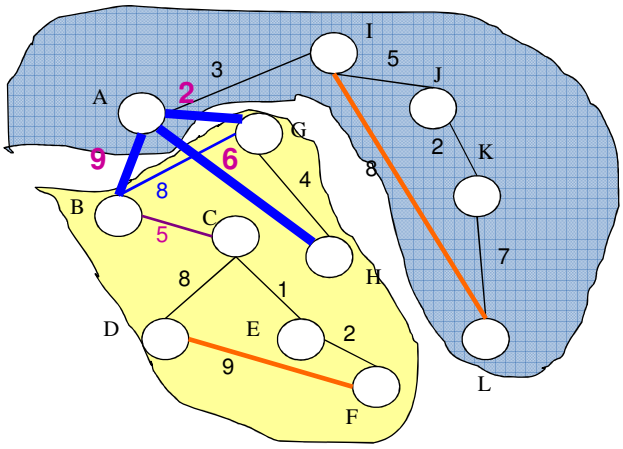
- Varianta 2 – Continuare
- Complexitate
  - O(V*E) //E=numarul de reuniuni
- Inacceptabil pentru grafuri ff mari

Variante de implementare multimi disjuncte – reuniune rapida

- se foloseste tot un vector auxiliar de id-uri

A	B	C	D	E	F	G	H	I	J	K	L
0	1	2	3	4	5	6	7	8	9	10	11
8	1	1	2	2	4	1	6	8	8	9	10

- id[i] reprezinta parintele lui i
- pentru radacina arborelui id[i]=i



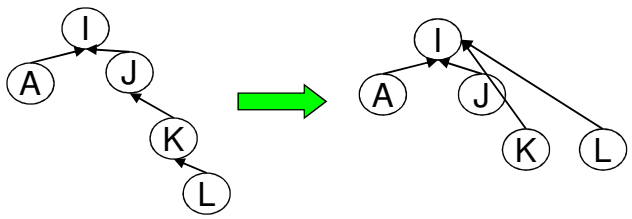
Variante de implementare multimi disjuncte – reuniune rapida

- cautare
  - Arb(u)!=Arb(v)
  - verifica daca 2 noduri au aceeasi radacina
  - Arb(u)
    - if(id[u]==id[id[u]]
      - return u
    - return Arb(u)
- reuniune(u,v)
  - p[v]=u;

Optimizarea reuniunii rapide

- compresia caii
  - Arb(u)
  - if(id[u]==id[id[u]]
    - return u
  - id[u]=Arb(u)
  - return id[u]
- salvarea inaltimii arborelui pentru a minimiza inaltimea arborelui rezultat

A	B	C	D	E	F	G	H	I	J	K	L
0	1	2	3	4	5	6	7	8	9	10	11
8	1	1	2	2	4	1	6	8	8	9	10



Complexitate dupa optimizari

- Orice secventa de E operatii de cautare si reuniune consuma O(V+E*α(V,E))
- α – de cate ori trebuie aplicat log pentru a ajunge la 1
  - in practica este <=5
- => in practica O(E)

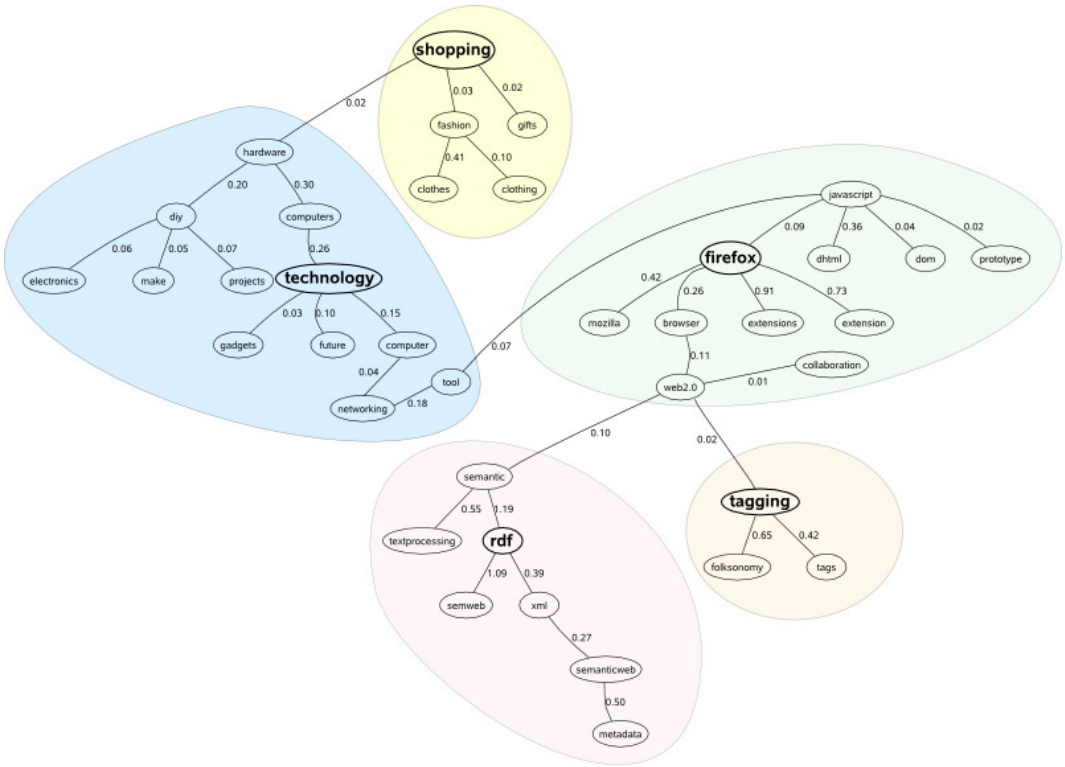
Complexitate Kruskal

- max(complexitate sortare, complexitate operatii multimi)=
- max(O(ElogV), O(E))=O(ElogV)
- => complexitatea algoritmului Kruskal este data de complexitatea sortarii costurilor muchiilor

Aplicatie practica

- k-clustering
  - impartirea unui set de obiecte in grupuri astfel incat obiectele din cadrul unui grup sa fie “apropiate” considerand o “distanta” data
- utilizat in clasificare, cautare (web search de exemplu)
- dandu-se un intreg K sa se imparta grupul de obiecte in K grupuri astfel incat spatiul dintre grupuri sa fie maximizat

Exemplu [5]


shopping
0.02
0.03
0.02
fashion
gifts
0.41
0.10
clothes
clothing
hardware
0.20
0.30
diy
computers
0.06
0.05
0.07
0.26
electronics
make
projects
technology
0.03
0.10
0.15
gadgets
future
computer
0.04
tool
networking
0.18
0.07
javascript
0.09
0.36
0.04
0.02
firefox
dhtml
dom
prototype
0.42
0.26
0.91
0.73
mozilla
browser
extensions
extension
0.11
collaboration
web2.0
0.01
0.10
0.02
semantic
tagging
0.55
1.19
0.65
0.42
textprocessing
rdf
folksonomy
tags
1.09
0.39
semweb
xml
0.27
semanticweb
0.50
metadata

Algoritm

- se formeaza V clustere
- gaseste cele mai apropiate 2 obiecte din clustere diferite
  - uneste cele 2 clustere
- opreste cand au mai ramas k clustere

- =>chiar algoritmul Kruskal

Bibliografie

1. http://monalisa.cacr.caltech.edu/monalisa__Service_Applications__Monitoring_VRVS.html
2. http://www.cobblestoneconcepts.com/ucgis2summer2002/guo/guo.html
3. Introducere in Algoritmi – C. Giumale
4. R. Sedgewick, K Wayne – curs de algoritmi Princeton 2007 www.cs.princeton.edu/~rs/AlgsDS07/ 1 si 14
5. http://www.pui.ch/phred/automated_tag_clustering/