Power Graph

Vlad Manea, Sebastian Codrin Dițu, Vlad Andrei Tudose
{vlad.manea, sebastian.ditu, vlad.tudose}@info.uaic.ro

Obiective

Proiectul de față are ca obiective:

1. reprezentarea unui graf neorientat oarecare desenat de utilizator cu obiecte Shape într-un Slide,
2. aplicarea unor algoritmi pe graful identificat și furnizarea rezultatelor lor.

Structură

Aplicația este structurată pentru a reduce la minim duplicarea codului, conform principiilor *don't repeat yourself* și *rule of three*. Astfel, am implementat un modul care conține funcțiile și procedurile necesare managementului matricei de adiacență a grafului, iar cei patru algoritmi se folosesc de acest modul. Aplicația a fost modelată de studenții Vlad Manea și Sebastian Codrin Dițu.

Reprezentare

Reprezentarea grafului a fost implementată de studenții Vlad Manea și Sebastian Codrin Dițu. Graful este dedus din obiectele Shape din slide. Pe scurt, obiectele de tip Rectangle reprezintă vârfurile grafului, iar obiectele de tip Line reprezintă muchiile grafului. Primul pas în construcția grafului este numerotarea vârfurilor prin parcurgerea colecției de obiecte Shape din obiectul Slide specificat. Al doilea pas este înregistrarea muchiilor ca legături între vârfuri. Desigur, pentru ca o muchie să fie înregistrată, obiectul Line asociat trebuie să intersecteze la extremități câte un obiect de tip Rectangle ce reprezintă un vârf. Intersecția este testată de funcția PointInShape.

Graful este reținut folosind obiecte de tip Collection. Am ales această metodă pentru a permite un număr oarecare de vârfuri și muchii în graf. Matricea de adiacență este reținută într-o colecție de colecții mobjsEdges, în care fiecare element mobjsEdges(intIndexI)(intIndexJ) este -1 sau indicele obiectului Line din colecția tuturor obiectelor Shape din obiectul Slide specificat.

Funcția de cost a muchiilor, necesară pentru unii algoritmi, a fost aleasă să fie lungimea liniei definită de obiectul Line al cărui indice este memorat în matricea mobjsEdges. Această lungime este calculată în funcția EuclideanDistance.

Până la momentul de față, nu am găsit o metodă pentru a identifica extremitățile segmentului de tip Line desenat. Concret, un segment poate fi desenat programatic între două puncte specificate, folosind AddLine cu parametrii coordonatele celor două puncte, în ordine cel inițial și final. Dar extremitățile segmentului, o dată desenat, nu mai pot fi preluate folosind VBA. Singurele proprietăți posibile sunt extremele Top, Left și dimensiunile Height, Width care descriu dreptunghiul de dimensiuni minime care are laturile paralele cu axele de coordonate (*bounding box*) care include obiectul.

În acest sens, pentru fiecare colț posibil al segmentului reținem în câte un obiect de tip Collection indicii obiectelor care se intersectează cu respectivul colț. Apoi, pentru a stabili exact muchia grafului, invităm utilizatorul să aleagă perechea de vârfuri între care se dorește a fi muchie. Perechile pot fi de forma (vârfTR, vârfBL) unde vârfTR este indicele de vârf asociat unui obiect Rectangle care se intersectează cu colțul din dreapta sus (Top Right) forma *bounding box* a obiectului Line ce definește muchia respectivă. Corespondent, pentru vârfBL. De asemenea, perechile pot fi de forma (vârfTL, vârfBR), pentru a permite segmentului dat de obiectul Line să aibă pantă negativă.

De asemenea, experimental am arătat că valorile furnizate de proprietățile Top, Left, Width, Height nu sunt consistente când lungimea formei bounding box a unui obiect Line este mai mică decât înălțimea. Această limitare ne-a obligat să alegem grafuri unde obiectele Shape de tip Line nu au această proprietate.

Algoritmi

Am implementat 4 algoritmi în VBA folosind matricea de adiacență asociată grafului.

Arbore parțial de cost minim

Prima problemă, rezolvată de Vlad Andrei Tudose, este cea de a identifica arborele parțial de cost minim, abreviat APM, într-un graf conex. În acest scop, am implementat în procedura ComputeAPM algoritmul Prim. Complexitatea timp a algoritmului este **O**(n^2), unde n este numărul de vârfuri din graf.

Algoritmul este descris în continuare:

1. selectează și vizitează un vârf oarecare din vârfurile 1, 2, ..., n-1, n, fie acesta cel cu indicele 1,
2. Drumul minim de la 1 la orice vârf este reținut ca fiind ∞, mai puțin cel până la el însuși, care este 0,
3. De n ori:
    a. selectează vârful nevizitat cel mai apropiat de componenta conexă de vârfuri vizitate,
    b. vizitează respectivul vârf și "apropie" celelalte vârfuri.

Componente conexe

A doua problemă, rezolvată de Sebastian Codrin Dițu și Vlad Manea, este cea de a identifica într-un graf componentele conexe. Am implementat acest lucru în procedura ComputeConnectedComponents. În acest scop, am parcurs recursiv în adâncime (DFS) graful și am asociat o culoare random (folosind funcția rnd) tuturor muchiilor din fiecare componentă conexă. În procedura DFS, pentru fiecare vârf nevizitat din graf:

1. alege random o culoare pentru muchii,
2. parcurge recursiv componenta conexă din respectivul vârf,
3. colorează cu respectiva culoare muchiile din arborele de parcurgere DFS,
4. colorează cu respectiva culoare muchiile rămase necolorate în respectiva componentă conexă

Desenarea se realizează folosind funcția RGB cu parametri naturali între 0 și 255 aplicată asupra culorii unui Line.

Drum minim între două vârfuri oarecare

A treia problemă, rezolvată de Vlad Manea, este cea de a identifica într-un graf drumurile minime între oricare două vârfuri cu indicii lngStart și lngEnd. Pentru aceasta, am implementat algoritmul Roy Floyd, folosind procedura ComputeRoyFloyd. Complexitatea timp a algoritmului este **O**(n^3), unde n este numărul de vârfuri din graf. Algoritmul a fost implementat după cum urmează:

1. prumul minim între oricare două vârfuri este lungimea muchiei dintre ele sau ∞, dacă muchia nu există,
2. pentru fiecare indice de vârf intermediar lngIndexK:
    a. pentru fiecare indice de vârf de start lngIndexI:
        i. dacă drumul minim dintre lngIndexI și lngIndexK nu este ∞,
            1. pentru fiecare indice de vârf de end lngIndexJ:
                a. dacă drumul de la lngIndexI la lngIndexJ prin lngIndexK e mai scurt,
                    i. alege respectivul drum între lngIndexI și lngIndexJ,
                    ii. reține lngIndexK ca vârf intermediar.

Pentru reconstruirea drumului dintre vârfurile lngStart și lngEnd, folosesc o procedură recursivă GetPath care:

- în cazul elementar, colorează muchia respectivă,
- în cazul general, se autoapelează între (lngStart și vârful intermediar) și apoi între (vârful intermediar și lngEnd), rezultând desenarea pas cu pas de la lngStart la lngEnd a muchiilor de pe drumul de cost minim.

Identificarea arborilor sau a unui circuit într-un graf

A patra problemă, rezolvată de Vlad Manea, este cea de a identifica pentru fiecare componentă conexă a unui graf dacă este arbore (caz în care se va desena cu o culoare verde) sau nu (caz în care se va identifica un circuit cu roșu). Complexitatea timp a algoritmului este **O**(n^2), unde n este numărul de vârfuri din graf.

Algoritmul se rezolvă prin parcurgerea în adâncime a fiecărei componente conexe, similar cu al doilea algoritm. Procedura recursivă care identifică un ciclu, dacă există, este RedDFS și seamănă cu procedura DFS: dacă printre vecinii vârfului curent se găsește un vârf vizitat și care nu este părintele vârfului curent (din arborele rezultat al unei parcurgeri DFS), am identificat un ciclu. La desenare, se consideră ciclu ciclul propriu-zis, împreună cu drumul până la rădăcina din arborele DFS. Dacă respectiva componentă conexă este arbore, se desenează toate muchiile ei cu verde – a se observa că arborele DFS coincide cu componenta în care se parcurge.

În mod normal, se oprește parcurgerea la găsirea unui ciclu prin apelul exit sub. Acest lucru ar împiedica vizitarea vârfurilor rămase nevizitate în componenta conexă respectivă, ceea ce ar fi incorect, deoarece ele ar fi vizitate ulterior ca fiind în altă componentă conexă. Așadar, dacă a fost identificat un ciclu, se face un *switch* între proceduri: se vor parcurge în continuare vârfurile rămase nevizitate din respectiva componentă conexă folosind procedura recursivă DFS. La întoarcerea din recursie, dacă există un ciclu, acesta va fi desenat cu roșu.

Interfață

Elementele de interfață cu utilizatorul au fost implementate de Sebastian Codrin Dițu. Proiectul conține:

- un buton integrat în meniurile Power Point, cu numele powerGraphApp,
- un formular care permite selectarea algoritmului și obiectul Slide de unde începe algoritmul,
- un formular care permite selectarea extremităților muchiei în cazul de ambiguitate prezentat anterior,
- un formular care permite selectarea celor două vârfuri între care se calculează drumul Roy Floyd.

De asemenea, aplicația este proiectată să salveze într-un fișier PDF graful după fiecare pas al fiecărui algoritm. Procedurile de export PDF au fost implementate de Vlad Manea.

Bibliografie

1. **Microsoft Developer Network** Working with Shapes
   http://msdn.microsoft.com/en-us/library/aa221611%28v=office.11%29.aspx
2. **Visual Basic Programming Documentation** Dealing with Shapes
   http://www.visual-basic-dox.net/Que-Absolute.Beginner.s.Guide/0789730766/ch09lev1sec4.html
3. **LQNet** Power Point Object Model
   http://www.lqexcel.com/powerpoint.php
4. **LQNet** Power Point Shape Objects
   http://www.lqexcel.com/pptshapes.php

- **Wikipedia** Don't Repeat Yourself
  http://en.wikipedia.org/wiki/Don%27t_repeat_yourself
- **Wikipedia** Rule of Three
  http://en.wikipedia.org/wiki/Rule_of_three_(programming)

- **iStockPhoto** Tree design elements
  http://www.istockphoto.com/stock-illustration-5476608-tree-design-elements.php
- **Dragon Artz** Vector curly tree design preview
  http://dragonartz.files.wordpress.com/2009/01/_vector-curly-tree-design-preview1-by-dragonart.png