README

TEMA2 – ANALIZA ALGORITMILOR

LUPU ANDREEA – 321CB

Pentru a verifica daca exista un ciclu Hamiltonian intr-un graf trebuie sa verificam daca exista un ciclu in acest graf astfel incat toate nodurile grafului sa fie vizitate o singura data. Pentru a face aceste verificari trebuie sa ne asiguram ca sunt indeplinite o serie de conditii:

Pentru a ne asigura ca drumul ales este un ciclu si ca fiecare nod este vizitat o singura data trebuie sa ne asiguram ca din fiecare nod pleaca exact doua muchii si ca din fiecare nod se poate ajunge la nodul cu numarul 1. De asemenea, este obligatoriu ca toate nodurile grafului sa aiba cel putin doua noduri adiacente(sa existe o muchie prin care sa se ajunga la acel nod si o muchie prin care sa se continue drumul cu nodul urmator) pentru a se putea forma un ciclu in care toate nodurile sa fie vizitate o singura data.

De asemenea, trebuie sa existe o conditie privind orientarea drumului(ciclului) , astfel: daca exista o cale de parcurgere(ciclu) ce presupune o muchie de la un nod “x” la un nod “y”, atunci drumul va putea fi parcurs si in sens invers, cuprinzand muchia de la “y” la “x”(deoarece trebuie sa fie ciclu=>fie exista ambele muchii, fie niciuna).

De asemenea, daca cea mai scurta cale de a ajunge de la un nod in nodul cu numarul 1 are lungimea 1(o muchie), atunci trebuie neaparat ca acest nod sa fie adiacent cu nodul numarul 1. Daca un nod nu e adiacent cu nodul numarul 1, atunci cea mai scurta cale de a ajunge de la nodul respectiv la nodul cu numarul 1 e imposibil sa fie egala cu 1(o muchie).

De asemenea, daca de la un nod se poate ajunge pe cel mai scurt drum la nodul cu numarul 1 intr-un numar de “i” muchii, atunci trebuie sigur ca de la unul din vecinii nodului(in conditiile in care muchia care le leaga apartine drumului ales care trebuie sa fie ciclu) sa se poata ajunge la nodul cu numarul 1 printr-un drum(cel mai scurt) de lungimea “i-1” muchii si nu trebuie sa existe posibilitatea ca de la nodul curent sa se poata ajunge la nodul cu numarul 1 printr-un numar mai mic de “i” muchii.

Detalii legate de implementare:

Pentru a tine evidenta nodurilor si a muchiilor care le leaga am ales sa folosesc o matrice in care indicii de linie si indicii de coloana reprezinta numarul nodurilor (incepand cu1, ignorand linia si coloana 0 a matricii) si marchez cu “1” nodurile adiacente(de ex: daca nodul i si nodul j sunt adiacente, atunci voi complete in matrice la pozitiile (i,j) si (j, i) cu “1”)

Prima data verific daca toate nodurile au cel putin doua noduri adiacente, astfel: pentru fiecare linie a matricii parcurg fiecare coloana si numar de cate ori am valoare “1” pe acea linie(adica pentru fiecare nod=linie a matricii calculez cate noduri adiacente exista) si daca intalnesc noduri care au mai putin de 2 noduri adiacente atunci afisez “x1-1&~x1-1”, adica o expresie nesatisfiabila deoarece nu poate exista un ciclu care sa viziteze toate nodurile grafului o singura data daca din fiecare nod nu pleaca doua muchii(daca nodul nu are doi vecini), altfel voi continua verificarile.

Cu metoda conditieA fac verificarea ca din fiecare nod al grafului sa se poata ajunge la nodul cu numarul 1, adica parcurg fiecare nod, incepand cu al doilea si pun conditia ca cel putin un ai-“nrNod” sa fie adevarat pentru fiecare nod, unde i ia toate valorile posibile(de la 1 la N/2+1), facand “sau” intre toate ai-“nrNod” pentru fiecare nod in parte.

Cu metoda conditieCiclu fac verificarea ca din fiecare nod al grafului sa plece exact doua muchii care sa apartina drumului ales(ciclu), pentru ca fiecare nod sa fie vizitat o singura data. Astfel, iau fiecare nod al grafului(fiecare linie a matricii) si pun intr-un ArrayList toate nodurile adiacente ale nodului curent(parcurgand fiecare coloana a matricii si verificand daca elementul de la linia si coloana respective e “1”) . Apoi fac combinari de noduri adiacente luate cate doua pentru a genera toate variantele posibile de perechi de doua muchii care sa plece din nodul curent, pe restul muchiilor care fac legatura dintre nodul curent si un nod vecin negandu-le(pentru ca trebuie sa existe exact doua muchii). Pentru a genera aceste perechi de cate 2 muchii, parcurg lista nodurilor adiacente plecand de la primul nod adiacent si pentru fiecare astfel de nod parcurg in continuare lista de noduri adiacente incepand de la cel ce urmeaza dupa nodul curent, combinand astfel cate doua noduri adiacente. Apoi, pentru fiecare nod trebuie sa pun conditia ca una dintre aceste combinari sa fie adevarata(din toate variantele de a alege exact doua muchii care sa plece din nod una trebuie sa fie valida/adevarata).

Cu metoda conditieOrientare fac verificarea ca daca exista o muchie ce pleaca de la nodul “i” la nodul “j”, atunci trebuie sa existe si muchia care pleaca de la nodul “j” la nodul “i”(ciclul sa existe oricare ar fi sensul de parcurgere), astfel: parcurg partea matricii de deasupra diagonalei principale si pentru fiecare nod(=linie a matricii) si pentru fiecare vecin al nodului verific conditia ca daca exista o muchie in drum care pleaca de la nod la vecin, atunci trebuie sa exist esi o muchie care pleaca de la vecin la nod (adica fac “si” intre “existenta muchie de la nod la vecin sau negare existenta muchie de la vecin la nod” si “existenta muchie de la vecin la nod sau negare existenta muchie de la nod la vecin”).

Cu metoda conditieAPrimNod fac verificarea ca daca nu exista in drum o muchie care sa lege nodul numarul 1 cu un alt nod, atunci cel mai scurt drum de la nodul respectiv la nodul cu numarul 1 sa nu poata fi egal cu 1 muchie si daca exista muchia, atunci cel mai scurt drum sa fie egal cu 1 muchie, astfel: parcurg fiecare nod al grafului si daca acel nod nu este vecin cu nodul numarul 1 verific conditia ca a1-“nrNod” sa nu fie adevarata(o neg) si daca nodul e vecin cu nodul numarul 1, atunci verific ca daca drumul continue muchia dintre aceste noduri, atunci calea cea mai scurta sa fie de 1 muchie.

Cu metoda conditieAuri fac verificarea ca daca de la un nod se poate ajunge la nodul cu numarul 1 printr-un numar de “i” muchii, atunci de la unul din vecinii acestui nod sa se poata ajunge la nodul cu numarul 1 printr-un numar de “i-1” muchii si sa nu existe niciun drum mai scurt de “i” muchii de la acest nod la nodul cu numarul 1, astfel: pentru toate lungimile posibile pe care le poate avea un drum de la un anumit nod la nodul cu nuarul 1(excluzand cazul in car aceasta lungimea este 1), pentru fiecare nod al grafului(pornind de la nodul cu numarul 2) iau toate nodurile adiacente cu acesta si verific conditia ca dintr-unul dintre acestea, cu care nodul e legat in drumul ales, sa se poata ajunge la nodul cu numarul 1 prin “i-1” muchii, apoi verific si conditia ca din nodul curent sa nu se poata ajunge la nodul cu numarul 1 printr-un numar mai mic decat “i” muchii, luand toate posibilitatile de lungimea de la 1 la i-1 muchii.

Deci pentru ca intr-un graf sa existe un ciclu Hamiltonian trebuie sa fie indeplinite simultan toate conditiile enuntate mai sus.

Demonstratie ca transformarea aleasa este polinomiala:

Complexitate metoda conditieA:

-primul for se executa in (N-1) pasi, iar for-ul din interiorul lui se executa in (N/2+1) pasi =>

(N-1)\*(N/2+1) = N^2/2 + N/2 -1 => complexitatea metodei este O(N^2)

Complexitate metoda conditieCiclu:

-primul for se executa in N pasi si contine un for care se executa in N pasi si un for care se executa in <=(N-2) pasi si care contine la randul sau un for care se executa in <=(N-1-k) pasi, unde k=0 : (N-2) si care la randul sau contine 3 for-uri, dintre care cel mai mare numar de pasi poate fi (N-2) =>

N \*(N + suma din (N-1-k)\*(N-2), unde k pleaca de la 0 la (N-2)) => complexitatea metodei este O(N^4)

Complexitate metoda conditieOrientare:

-primul for se executa in N pasi si continue un for care se executa in (N-i) pasi, unde i ia valori de la 1 la N =>suma din (N-i), unde i ia valori de la 1 la N => complexitatea metodei este O(N^2)

Complexitate metoda conditieAPrimNod:

-metoda contine un singur for care se executa in N pasi => complexitatea metodei este O(N)

Complexitate metoda conditieAuri:

-primul for se executa in N/2 pasi si contine un for care se executa in (N-1) pasi care la randul lui contine doua for uri care se executa in (N-1) pasi si doua care se executa in i pasi, unde i poate avea valori de la 2 la N/2+1 => N/2\*(N-1)\*(N-1) => complexitatea metodei este O(N^3)

Cum transformarea presupune executarea metodelor definite mai sus si cuprinde si un for care se executa in N pasi care contine un for care se executa in N pasi => O(N^2) =>complexitatea programului este O(N^4) =>deci transformarea este polinomiala