Universitatea de Stat din Chisinau, Facultatea de Matematica si Informatica

LUCRARE DE LABORATOR NR.2

La disciplina: Matematica Discreta Algebra Geometrica

Tema: Determinarea multimii independente de varfuri maximala(care tinde spre maxima) intr-un graf neorientat

Efectuat de: Lupascu Iulii

Grupa: IA2402 (subgrupa I)

Coordonator: Niculita Angela

**Algoritm de Aproximare pentru Mulțime Independentă Maximă (algoritm greedy cu ordonarea vârfurilor) Deoarece găsirea unei mulțimi independente maxime (MIM) este o problemă NP\*-hard, algoritmii de aproximare oferă soluții rapide și aproape optime, mai ales pentru grafuri mari. Acest algoritm greedy selectează vârfurile pe baza unei ordini specifice (de obicei în funcție de gradul vârfului) pentru a construi eficient o mulțime independentă mare. \* Termenul informal hard, utilizat mai sus, se definește ca existența unui algoritm care rulează în timp polinomial. Clasa generală de întrebări pentru care un algoritm poate da răspuns în timp polinomial se numește „clasa de complexitate P”, sau simplu „P”. Pentru unele probleme însă, nu există o metodă de a găsi rapid un răspuns, dar dacă unui calculator i se prezintă un posibil răspuns, el îl poate verifica rapid. Clasa de astfel de probleme care pot fi verificate în timp polinomial se numește NP, care înseamnă „timp nedeterminist polinomial”. Algoritm Greedy de aproximare Ideea: 1. Prioritizăm vârfurile cu grad mic → Acestea au mai puține vârfuri adiacente (“vecini”), ceea ce le face mai potrivite pentru a face parte dintr-o mulțime independentă mare. 2. Selectăm iterativ un vârf, eliminând vecinii săi pentru a menține independența. Pașii algoritmului 1. Inițializăm o mulțime independentă S vidă. 2. Alegem vârful v de grad minim în graf (prioritizăm vârfurile mai puțin “conectate”) 3. Adăugăm vârful v în S. 4. Determinăm vecinătatea vârfului v. 5. Eliminăm vârful v și vecinătatea sa. 6. Returnăm S ca o aproximare a mulțimii independente maxime. Performanță și Compromisuri Execuție rapidă (nu este boptimal, dar eficient). Util pentru grafuri mari unde algoritmii exacți sunt impractici. Acest algoritm greedy de aproximare oferă o soluție rapidă și scalabilă pentru grafuri mari.**

**Codul**

```#include <iostream>

#include <vector>

using namespace std;

class Graf {

public:

    int nrnoduri;

    int matrice[50][50];

    Graf(int n) {

        nrnoduri = n;

        for (int i = 0; i < nrnoduri; i++) {

            for (int j = 0; j < nrnoduri; j++) {

                matrice[i][j] = 0;

            }

        }

    }

    void adaugaMuchie(int v, int u) {

        matrice[u][v] = 1;

        matrice[v][u] = 1;

    }

    void afiseazaMatricea() {

        cout << "Matricea de adiacenta este: \n";

        for (int i = 0; i < nrnoduri; i++) {

            for (int j = 0; j < nrnoduri; j++) {

                cout << matrice[i][j] << " ";

            }

            cout << endl;

        }

    }

    void gasesteS() {

        int M[50];  // Multimea M va contine nodurile

        bool exclus[50] = {false};  // Marcam nodurile excluse

        int k = 0;  // Contor pentru multimea S

        int S[50];  // Multimea finala a nodurilor excluse

        while (k < nrnoduri) {

            int gradMin = nrnoduri + 1;

            int varfMin = -1;

            // Calculam gradele fiecarui varf (luand in considerare doar vecinii nevizitati)

            for (int i = 0; i < nrnoduri; i++) {

                if (exclus[i] == false) {  // Verificam doar nodurile neexcluse

                    int grad = 0;

                    for (int j = 0; j < nrnoduri; j++) {

                        if (matrice[i][j] == 1 && exclus[j] == false) {

                            grad++;  // Contează vecinii nevizitați

                        }

                    }

                    if (grad < gradMin) {

                        gradMin = grad;

                        varfMin = i;

                    }

                }

            }

            if (varfMin == -1) {

                // Nu mai există noduri cu vecini nevizitați, deci terminăm

                break;

            }

            // Adaugam varful cu gradul minim in multimea S

            S[k++] = varfMin;

            exclus[varfMin] = true;  // Marcam varful ca fiind exclus

            cout << "Varful cu gradul minim: " << varfMin << endl;

            // Excludem vecinii acestui varf

            for (int i = 0; i < nrnoduri; i++) {

                if (matrice[varfMin][i] == 1 && exclus[i] == false) {

                    exclus[i] = true;  // Excludem vecinul

                    cout << "Vecin exclus: " << i << endl;

                }

            }

        }

        // Afisam multimea S (noduri excluse)

        cout << "Multimea S: ";

        for (int i = 0; i < k; i++) {

            cout << S[i] << " ";

        }

        cout << endl;

    }

};

int main() {

    int n, a, b, m;

    cout << "Introdu numarul de noduri: " << endl;

    cin >> n;

    cout << "Introdu numarul de muchii (mai mic decat combinari n a cate 2): " << endl;

    cin >> m;

    if (m > (n \* (n - 1)) / 2) {

        cout << "Ai depasit valoarea corecta" << endl;

    } else {

        Graf Graf(n);

        for (int i = 0; i < m; i++) {

            cout << "Pentru " << i + 1 << "-a muchie: ";

            cin >> a >> b;

            Graf.adaugaMuchie(a, b);

        }

        Graf.afiseazaMatricea();

        // Calculam si afisam multimea S

        Graf.gasesteS();

    }

    return 0;

}```

Изображение выглядит как линия

Контент, сгенерированный ИИ, может содержать ошибки. Pentru oricare graf, programul incepe cu cel mai mic varf din cele care au acelasi grad(cu indexul ‘0’ in cazul dat).

Primul pas se formeaza matricea de adiacenta goala prin 2 cicluri. Apoi se introduce o limita “m” a muchiilor, care sa nu permita sa fie numarul mai mare ca el. Mai departe matricea este completata prin adaugarea muchiilor din graf. Apoi deja are loc executarea algoritmului care consta in:

1. Formarea unei multimi, care initial este vida (adica nu are elemente);
2. Calcularea gradelor fiecarui varf din graf;
3. Selectarea varfului cu grad cel mai mic ( in acelasi timp si cu index cel mai mic);
4. Acest varf este adaugat in multimea formata la pasul 1;
5. Se selecteaza vecinatatea acestui varf;
6. Varfurile vecine se adauga in multimea de mai sus;
7. Daca mai raman varfuri nevizitate si neanalizate in graf, atunci ne reintoarcem la pasul 2 si continuam sa analizam pana nu mai raman varfuri neanalizate in graf. In caz contrar programul este oprit si se afiseaza sirul de varfuri;

In cod am folosit clase pentru a structura mai usor lucrul cu graful. Am format un constructor care formeaza matricea de adiacenta goala. Apoi este formata o functie pentru adaugarea muchiilor. La fel este formata o functie care afiseaza matricea de adiacenta deja dupa adaugarea muchiilor. Persista si functia care se ocupa de cautarea varfurilor vecine. Lucreaza in felul urmator: toate varfurile (neexcluse) primesc valoarea “neexclus” care, si este format un sir gol in care se vor adauga varfurile dupa ce le excludem. Parcurgem toate varfurile si se determina gradul lor. Apoi se verifica daca au fost vizitate sau nu, si calculam gradele lor (luand in considerare doar varfurile nevizitate de pana acum). Si daca este nevizitat si in matrice are valoarea egala cu 1 atunci i se calculeaza gradul. Daca gradul este mai mic ca cel initial, atunci valoarea se salveaza, si indexul varfului tot. In asa mod se verifica toate varfurile si se determina indexul si gradul a varfului cu cel mai mic grad. Daca varfuri nu mai raman atunci aceasta valoare va fi -1 ceea cce va insemna ca sau finisat varfurile. Varful gasit se adaoga in multimer, folosinf indexul. Si i se atribuie valoarea “exclus” = true. Apoi excludem si varfurile acestui varf si i se atribuie valoarea “exclus” = true. Si in fina; este afisata multimea cand indexul devine egal cu numarul de varfuri.

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.