Proiectarea Algoritmilor 2009-2010

Laborator 2-3

Greedy si Programare Dinamica

1. Obiective laborator

a. Intelegerea notiunilor de baza legate de tehnicile greedy si programare dinamica.

b. Insusirea abilitatilor de analiza in vederea conceperii algoritmilor de tip greedy si programare dinamica pentru rezolvarea diferitelor probleme.

c. Insusirea abilitatilor de implementare a algoritmilor bazati pe greedy si programare dinamica.

2. Aplicatii practice

De obicei, atat Greedy, cat si Programarea Dimanica sunt tehnici folosite pentru rezolvarea problemeleor de optimizare. Acestea pot fi exercitii in sine sau pot fi subprobleme dintr-un algoritm mai mare.

Tehnica greedy este folosita adesea in interiorul unui algoritm mai mare. De exemplu, algoritmul Dijkstra pentru determinarea drumului minim pe un graf alege la fiecare pas un nod nou urmarind algoritmul greedy.
Exista insa probleme care ne pot pacali. Exista probleme in care urmarind criteriul Greedy nu ajungem la solutia optima. Este foarte important sa identificam cazurile cand se poate aplica Greedy si cazurile cand avem nevoie de altceva. Alteori aceasta solutie neoptima este o aproximare suficienta pentru ce avem nevoie. Problemele NP-complete necesita multa putere de calcul pentru a gasi optimul absolut. Pentru a optimiza aceste calcule multi algoritmi folosesc decizii Greedy si gasesc un optim foarte aproape de cel absolut.

Programarea dinamica are un camp larg de aplicatie, aici amintind genetica (sequence alignment), teoria grafurilor (algoritmul Floyd-Warshall), metode de antrenare a retelelor neurale (Adaptive Critic training strategy), limbaje formale si automate (algoritmul Cocke-Younger-Kasami, care analizeaza daca si in ce fel un sir poate fi generat de o gramatica independent de context), implementarea bazelor de date (algoritmul Selinger pentru optimizarea interogarii relationale) etc.

3. Descrierea problemei si a rezolvarilor

3.1. Greedy

In limba engleza cuvantul “greedy” inseamna “lacom”. Algoritmii de tip greedy vor sa construiasca intr-un mod cat mai rapid solutia unei probleme. Ei se caracterizeaza prin luarea unor decizii be baza unui criteriu optim simplu si usor de verificat, care duc la gasirea unei solutii a problemei. Nu intotdeauna asemenea decizii verificabile usor duc la o solutie optima, de aceea trebuie sa reusim identificarea acelor tipuri de probleme pentru care se pot obtine solutii optime.

Algorimii greedy se numara printre cei mai directi algoritmi posibili. Ideea de baza este simpla: avand o problema de optimizare, de calculare a unui cost minim sau maxim, se va alege la fiecare pas decizia cea mai favorabila, fara a evalua global eficienta solutiei. In general exista mai multe solutii posibile ale problemei. Dintre acestea se pot selecta doar anumite solutii **optime**, conform unor anumite criterii. Scopul este de a gasi una dintre acestea sau daca nu este posibil, atunci o solutie cat mai apropiata, conform criteriului optimal impus.

Trebuie sa intelegem ca rezultatul obtinut este optim doar daca un optim local conduce la un optim global. In cazul in care deciziile de la un pas influenteaza lista de decizii de la pasul urmator, este posibila obtinerea unei valori neoptimale. In astfel de cazuri, pentru gasirea unui optim absolut se ajunge si la solutii suprapolinomiale. De aceea, daca se opteaza pentru o astfel de solutie, algoritmul trebuie insotit de o demonstratie de corectitudine.

Descrierea formala a unui algoritm greedy este:

```
function greedy(C)
        // C este multimea candidatiilor
        // in S construim solutia
        S ← Ø
        while not solutie(C) and C≠Ø
                x ← un element din C care maximizeaza/ minimizeaza select(x)
                C ← C\{x}
                if fezabil(S□{x}) then S←S□{x}
        return S
```

Este de inteles acum de ce acest algoritm se numeste lacom: la fiecare pas se alege cel mai bun candidat de la momentul respectiv, fara a studia situatiile viitoare.
Procesul de construire al solutiei este irevocabil: solutiile partiale ce nu pot fi optimizate sunt abandonate fara posibilitatea reevaluarii la un pas anterior (x este eliminat din C chiar daca nu va face parte din S final), iar daca o subsolutie este gasita fezabila la pasul curent atunci el va face parte cu siguranta din solutia finala.

Exemplu de problema rezolvabila cu tehnica Greedy:
Fie un sir de n numere se cere determinarea unui subsir de numere cu suma maxima. Un subsir al unui sir este format din caractere (nu neaparat consecutive) ale sirului respectiv, in ordinea in care acestea apar in sir.

Pentru numerele 1 -5 6 2 -2 4 raspunsul este 1 6 2 4 (suma 13).

Se observa ca tot ce avem de facut este sa verificam fiecare numar daca este pozitiv sau nu. In caz ca da, il introducem in subsirul solutie.

Problema cuielor:

Fie N scanduri de lemn, descrise ca niste intervale inchise cu capete reale. Gasiti o multime minima de cuie astfel incat fiecare scandura sa fie batuta de cel putin un cui. Se cere pozitia cuielor. Formulat matematic: gasiti o multime de puncte de cardinal minim M astfel incat pentru orice interval [ai, bi] din cele N, sa existe un punct x din M care sa apartina intervalului [ai, bi]. Complexitate: O(NlogN)

Exemplu:

- intrare: N = 5, intervalele: [0, 2], [1, 7], [2, 6], [5, 14], [8, 16]
- iesire: M = {2, 14}
- explicatie: punctul 2 se afla in primele 3 intervale, iar punctul 14 in ultimele 2

Solutie: Se observa ca daca x este un punct din M care nu este capat drepata al nici unui interval, o translatie a lui x la dreapta care il duce in capatul dreapta cel mai apropiat nu va schimba intervalele care contin punctul. Prin urmare, exista o multime de cardinal minim M pentru care toate punctele x sunt capete dreapta.

Astfel, vom creea multimea M folosind numai capete dreapta in felul urmator:

- cat timp au mai ramas intervale nemarcate
    - selectam cel mai mic capat dreapta, Bmin; acesta trebuie sa fie in M, deorece este singurul punct care se afla in interiorul intervalului care se termina in Bmin
    - marcam toate intervalele nemarcate care contin Bmin
    - adaugam Bmin la M

Pentru a obtine o complexitate redusa, sortam initial toate cele 2N capate si le parcurgem de la stanga la dreapta. Pentru fiecare punct distingem cazurile:

- daca este capat stanga, introducem intervalul in lista de „intervale in procesare” si trecem mai departe
- daca este capat dreapta si intervalul respectiv nu contine nici un punct din M, atunci am gasit cel mai mic capat drepata al unui interval nemarcat, introcucem capatul in M, si marcam toate intervalele din lista_de_intervale_in_procesare.
- daca este capat dreapta si intervalul din care face parte este deja marcat, trecem mai departe.

Complexitate:

- sortare: O(NlogN)
- parcurgerea capetelor: O(N)
- adaugarea si stergerea unui interval din lista de intervale in procesare: O(1)
- total: O(NlogN)

3.2. Programare Dinamica

Programare dinamica presupune rezolvarea unei probleme prin descompunerea ei in subprobleme si rezolvarea acestora. Spre deosebire de divide-et-impera, subproblemele nu sunt disjuncte, ci se suprapun.

Pentru a evita recalcularea portiunilor care se suprapun, rezolvarea se face pornind de la cele mai mici subprobleme si folosindu-ne de rezultatul acestora calculam subproblema imediat mai mare. Cele mai mici subprobleme sunt numite subprobleme unitare. Acestea pot fi rezolvate intr-o complexitate constanta, ex: cea mai mare subsecventa dintr-o multime de un singur element.

Pentru a nu recalcula solutiile subproblemelor ce ar trebui rezolvate de mai multe ori, pe ramuri diferite, se retine solutia subproblemelor folosind o tabela (matrice uni, bi sau multi-dimensionala in functie de problema) cu rezultatul fiecarei subprobleme. Aceasta tehnica se numeste **memorizare**.

Aceasta tehnica afla „valoarea” solutiei optime pentru fiecare din subprobleme. Mergand de la subprobleme mici la subprobleme din ce in ce mai mari ajungem sa gasim „valoarea” problemei intregi. Motivul pentru care aceasta tehnica se numeste Programare Dinamica este datorat flexibilitatii ei, „valoarea” schimbandu-si intelesul logic de la o problema la alta. In probleme de minimizarea costului, „valoarea” este acest cost minim. In probleme de aflarea unei componente maxime, „valoarea” este dimensiunea componentei.

Dupa calcularea valorii pentru toate subproblemele se pot afla efectiv elementele ce alcatuiesc solutia. „Reconstructia” solutiei se face mergand din subproblema in subproblema incepand de la problema cu valoarea optima si ajungand in subprobleme unitare. Metoda admite nuante in cazuri particulare, o sa se inteleaga mai bine din exemple.

Aplicand aceasta tehnica determinam **una** din solutiile optime, problema putand avea mai multe solutii optime. In cazul in care se doreste determinarea tuturor solutiilor optime, algoritmul trebuie combinat cu unul de backtracking in reconstructia solutiilor.

Dupa cum probabil ati intuit deja, diferenta majora dintre cele doua metode prezentate este ca algoritmul greedy mentine doar solutiile partiale de la pasul curent pentru a le folosi la pasul urmator, in timp ce programarea dinamica poate utiliza la un pas subsolutii generate la oricare alt pas anterior.

Programarea Dinamica poate fi descompusa in urmatoarea secventa de pasi:

1. Descoperirea structurii si "masurii" pe care o are o solutie optima.
2. Determinarea unei metode de calcul recursive pentru a afla valoarea fiecarei subprobleme.
3. Calcularea "de jos in sus" a acestei valori (de la subproblemele cele mai mici la cele mai mari)
4. Reconstructia solutiei optime pornind de la rezultatele obtinute anterior.

Exemple de probleme:

Programarea Dinamica este cea mai flexibila tehnica din programare. Cel mai usor mod de a o intelege este parcurgerea cat mai multor exemple.

O problema clasica de Programare Dinamica este determinarea celui mai lung subsir strict crescator dintr-un sir de numere. Un subsir al unui sir este format din caractere (nu neaparat consecutive) ale sirului respectiv, in ordinea in care acestea apar in sir.
Exemplu: pentru sirul 24 12 15 15 8 19 raspunsul este sirul 12 15 19

Se observa ca daca incercam o abordare greedy nu putem stabili nici macar elementul de inceput intr-un mod corect.

Se poate rezolva si cu un algoritm care alege toate combinatiile de numere din sir, valideaza ca sirul obtinut este strict crescator si il retine pe cel de lungime maxima, dar aceasta abordare are complexitatea temporala $O(2^N)$. Cu optimizari este posibil sa se ajunga la O(N!).

O metoda de rezolvare mai eficienta foloseste Programarea Dinamica. Incepem prin a stabili pentru fiecare element lungimea celui mai lung subsir strict crescator care incepe cu primul element si se termina in elementul respectiv. Numim aceasta valoare $best_i$ si aplicam formula recursiva $best_i = 1 + \max(best_j)$, cu $j<i$ si $elem_j < elem_i$.
Aplicand acest algoritm obtinem:

```
elem 24 12 15 15  8 19
best  1  1  2  2  1  3
```

Pentru 24 sau 12 nu exista nici un alt element in stanga lor strict mai mici decat ele, de aceea au best egal cu 1. Pentru elementele 15 se poate gasi in stanga lor 12 strict mai mic decat ele. Pentru 19 se gaseste elementul 15 strict mai mic decat el. Cum 15 deja este capat pentru un subsir-solutie de 2 elemente, putem spune ca 19 este capatul pentru un subsir-solutie de 3 elemente.

Cum pentru fiecare element din multime trebuie sa gasim un element mai mic decat el si cu best maxim, avem o complexitate O(N) pentru fiecare element. In total rezulta o complexitate $O(N^2)$. Se pot obtine si rezolvari cu o complexitate mai mica folosind structuri de date avansate. Atat solutia in $O(N^2)$, cat si o solutie in O(NlogN) poate fi gasita la [5]. Tot acolo se poate gasi si o lista de probleme mai dificile ce folosesc tehnica Programarii Dinamice, adaptata in diferite forme.

Pentru a gasi care sunt elementele ce alcatuiesc subsirul strict crescator putem sa retinem si o „cale de intoarcere”. Reconstructia astfel are complexitatea O(N). Exemplu: subproblema care se termina in elementul 19 are subsirul de lungime maxima 3 si a fost calculata folosind subproblema care se termina cu elementul 15 (oricare din ele). Subsirul de lungime maxima care se termina in 15 a fost calculat folosindu-ne de elementul 12. 12 fiind cel mai mic element din subsir marcheaza sfarsitul reconstructiei.

O alta problema cu o aplicare clasica a Programarii Dinamice este si determinarea celui mai lung subsir comun a doua siruri de caractere. Descrierea problemei, indicatii de rezolvare si o modalitate de evaluare a solutiilor voastre poate fi gasita la [6].

O problema care admite o varietate mare de soltii este cea a subsecventei de sume maxime. Enuntul poate fi gasita la [7]. Daca se studiaza rezolvariile la aceasta problema se poate observa ca cea cu Programare Dinamica este printe cele mai usoare de implementat.

4. Concluzii si observatii

Aceste 2 tehnici invatate sunt flexibile si simpliste in idee, dar cu ajutorul lor se pot rezolva probleme foarte complexe. In viitor este posibil sa intalniti Programare Dinamica pe arbori sau Greedy pe stari, unde fiecare stare este o matrice. Conceptele raman aceleasi.

O problema avansata ce se rezolva cu Greedy aplicat pe o structura arborescenta a fost data si la selectia echipei ACM a politehnicii. Va invitam sa o rezolvati aici: [8]

Aveti grija doar ca solutiile gasite de voi ajung la rezultatul optim. Se pot confunda usor problemele care se rezolva cu Greedy cu cele care se rezolva cu Programare Dinamica.

5. Referinte

[1] http://en.wikipedia.org/wiki/Greedy_algorithm

[2] http://en.wikipedia.org/wiki/Dynamic_programming

[3] http://ww3.algorithmdesign.net/handouts/Greedy.pdf

[4] http://ww3.algorithmdesign.net/handouts/DynamicProgramming.pdf

[5] http://infoarena.ro/problema/scmax

[6] http://infoarena.ro/problema/cmlsc

[7] http://infoarena.ro/problema/ssm

[8] http://infoarena.ro/problema/bile7

[9] Capitolul IV din Introducere in Algoritmi de catre T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein