

Laborator 00: Introducere și Relaxare

De ce Proiectarea Algoritmilor?

Răspunsul este simplu: complexitatea algoritmilor, plăcerea de a explora alternative, de a le compara și în final de a rezolva optim problemele. Totodată posibilitatea de a fi creativ și de ce nu chiar inovativ, capacitatea de a generaliza anumite aspecte și de soluționa probleme din viața de zi cu zi sunt doar câteva dintre aspectele pe care le vom explora împreună.

Nu ne rezumăm la calcul computațional; algoritmii pot fi aplicați în orice circumstanțe și fundamentează o serie de decizii care guvernează multiple aspecte din realitatea înconjurătoare. Absolut toți algoritmii analizați se regăsesc în viața de zi cu zi, într-adevăr cu frecvențe diferite, iar faptul că noi putem înțelege ce se întâmplă în profunzime ne poate oferi alfel "je ne sais quois" care să ne diferențieze.

Asfel, avem o plajă largă din care să alegem: în cazul *rețelisticii*: algoritmii de rutare, politicile de load-balancing și flux maxim; în *sisteme de operare*: scheduling, caching; *grafica* cu prelucrarea imaginilor, determinarea conturilor, iluminare și texturare; *baze de date* cu indecși, operatorul "select" în sine; *componente hardware* pornind de la banalul incrementor pe biți și până la procesoarele DSP pentru prelucrarea semnalelor; *inteligența artificială* cu tot ce înseamnă prelucrarea limbajului natural, analiza rețelelor sociale, analiză semantică, sisteme bazate pe reguli. Pe scurt, sky is the limit!

În altă ordine de idei, fără a ne auto-flata, dacă acum nu încercam să înțelegem și stăpânim algoritmii și posibilitățile oferite, atunci când?

Asfel accentul este pus pe partea formală, importanța și utilitățile algoritmilor, iar implementările propriu-zise pot fi considerate o „validare”.

Ultimul aspect cu adevărat important este că algoritmii stau la baza a cam tot ce înseamnă Computer Science, iar proiectarea eficientă a acestora face diferența.

Pentru cine?

În primul rând pentru tine și sperăm să conștientizezi acest lucru: totul se rezumă la a-ți construi o bază de cunoștințe utilă în viață, indiferent de specializarea pe care o vei urma. Oricum nu există subdomeniul din Computer Science în care să nu apelezi la algoritmi, oricât de simpli sau complecși. Pe de altă parte, numărul de persoane care pot spune că înțeleg algoritmi, știu să-i aplice și să determine soluția optimă este restrâns, creând astfel o comunitate activă și dinamică.

Ce vom face?

Vom aprofunda elementele fundamentale necesare rezolvării fiecărei clase de probleme analizate, prezentând mai multi algoritmi de rezolvare pentru fiecare problemă studiată și evidențiând algoritmi optimi. Pornind de la aceste elemente, vom accentua punctele de interes identificate, descoperind șabloane de rezolvare și modalități de construire a soluțiilor pentru o problemă.

Câteva din aspectele atinse includ:

- Divide et Impera, Greedy și programare dinamică;
- Parcurgeni pe grafuri (BFS, DFS) și sortare topologică;
- Căutări în spațiul stărilor (best-first, A*);
- Alte aplicații DFS (componente tare conexe, puncte de articulație, punți);
- Backtracking și optimizări (prospective și retroactive);
- Drumuri minime și arbori minimi de acoperire;
- Algoritmi Minimax;
- Fluxuri maxime;
- Algoritmi aleatori.

Ce nu vom face?

Copiat teme, laboratoare. Pentru a continua ideea simplă și pentru a accentua întrebarea pe care trebuie să ne-o punem cât mai des - de ce? -, formulăm un răspuns la fel de simplu și imediat: fiindcă este greșit. Analogia cea mai simplă pe care o putem face este cu viața la nivelul căreia întodeauna există un echilibru: ce faci întodeauna ți se întoarce, cu diverse ponderi, în unele cazuri putând fi chiar înzecit. Și dacă la o materie care îți stimulează creativitatea, inovația, te pune la încercare și care fundamentează abordările din Computer Science apelezi la astfel de metode, sincer trebuie să îți pui întrebarea: *eu ce caut la această facultate?*

Din punctul nostru de vedere nu copiezii din mândrie personală, la urma urmei este bine să fii deasupra celorla care apelează la astfel de metode. În plus să nu uităm de satisfacția personală ulterioară. Și ce dacă nu sunt prinși? Garantăm că tot ei au numai de pierdut, atât din prisma aspectelor învățate, cât și a implicațiilor ulterioare.

Alegerea acestei facultati a fost rezultatul unei documentări minuțioase care a dus la identificarea acestora cu idealurile personale ale fiecăruia dintre voi. Rolul pe care și-l asumăm un îndrumator este doar de a materializa acest vis și de a vă ajuta în atingerea țelurilor propuse la intrarea în facultate.

Metodele de copiat reprezintă mijloace care alterează atingerea obiectivelor proprii și a perspectivei despre nivelul și capacitățile personale.

Totodată acestea nu pot să influențeze perspectiva posibilitilor angajatori, în momentul trecerii în practică a cunoștințelor acumulate în perioada facultății. Astfel, singurul rezultat este pe termen scurt și reflectă zicala „ți-ai furat singur caciula”.

În plus, în concordanță cu principiul „carot and stick” va exista o verificare minuțioasă împotriva copierii.



Pe scurt, **la prima abatere – ~10p pe tema respectivă (punctajul maxim ce se poate obține pe temă), la a doua – restanță**. Regulile sunt foarte simple.

Evaluarea Temelor

Notarea temelor va avea la bază următoarele criterii și punctaje aferente:

- 5 puncte dacă se respectă cerințele temei, adică compilare și execuție fără erori astfel încât să se obțină rezultatele cerute;
- 3 puncte pentru o implementare eficientă;
- 1 punct pentru comentariile din fișierele sursă, menționarea bibliografiei în **README** (dacă este cazul), respectiv conținutul propriu-zis al fișierului **README**;
- 1 punct pentru Coding Style.

Pentru a asigura o evaluare uniformă a temelor, va exista o singură persoană care este responsabilă de corectarea unei teme la nivelul întregii serii.

Toate temele vor fi corectate folosind același set de teste. Pentru a veni în sprijinul vostru și pentru a testa corectitudinea și eficiența implementării temelor, fiecare enunț va avea un link către o pagină de pe  infoarena unde se pot testa rezolvările realizate în C/C++. Temele rezolvate în Java (sau alte limbaje de programare, dar cu acordul explicit al responsabilului de temă) nu vor putea fi testate folosind infoarena. Pentru aceste teme, aveți posibilitatea să folosiți direct  vmchecker, putând astfel să beneficiați de feedback înainte de trimiterea rezolvării temei.

Toate temele vor fi testate automat folosind vmchecker, dar și prin verificarea codului și citirea ReadMe-ului.

Coding Style

Stilul de „redactare” al programelor are impact major asupra celor care parcurg respectivul cod.

Dacă pentru compilator nu este nicio diferență, o indentare corespunzătoare ne ajută oferind:

- claritate;
- concizie;
- viteză mult mai mare de regăsire și reamintire a anumitor aspecte.

Totodată, stilul de codare ține și de gradul de profesionalism care se dorește a fi exprimat, comentariile marcând:

- coeziunea idelilor;
- structurarea acestora;
- claritatea;
- ordonarea acestora din perspectiva organizării personale.

De asemenea, comentariile ar trebui să exprime pe scurt detaliile de implementare într-un limbaj tehnic, ghidând astfel programatorul atunci când citește pentru prima oară respectivul cod.

Desigur există particularități specifice fiecărui limbaj de programare, precum și individuale din prisma preferințelor personale. Ulterior pot apărea constrângeri generate de specificul echipei sau companiei din care veți face parte, dar există un set general de reguli care merită urmat (exemplele au fost preluate de pe Wikipedia):

Prezentare generală

Formatare (indentare) generală

```
if (hours < 24 && minutes < 60 && seconds < 60) {
    return 1;
} else {
    return 0;
}
```

Nume adecvate și sugestive pentru variabile/funcții

```
int correctFormat (int hours, int minutes, int seconds) {
    if (hours < 24 && minutes < 60 && seconds < 60) {
        return 1;
    } else {
        return 0;
    }
}
```

În loc de:

```
int correctFormat (int a, int b, int c) {
    if (a < 24 && b < 60 && c < 60) {
        return 1;
    } else {
        return 0;
    }
}
```

Valori boolene în structuri de decizie

Doar pentru efect stilistic întrucât codul generat alternativ este oricum optimizat de compilator.

```
return (hours < 24) && (minutes < 60) && (seconds < 60);
```

Comparații de tip Left-hand:

```
if ( a = 42 ) { ... } // Eroare inadvertenta care poate aparea si e dificil de identificat
if ( 42 = a ) { ... } // Eroare de compilare
```

Cicluri și structuri de control

Utilizarea acoladelor și indentarea corespunzătoare nivelului de imbricare (evitarea problemelor care pot apărea de exemplu în Python datorate indetării greșite):

```
for (int i = 0; i < 5; i++) {
    printf("%d", i * 2);
}

printf("Ended loop");
```

Liste – elementele sunt plasate pe linii diferite:

```
const char *zile[] = {
    "luni",
    "marti",
    "miercuri",
    "joi",
    "vineri",
    "sambata",
    "duminica"
};
```

Caracteristici specifice fiecărui limbaj se regăsesc în secțiunea de referințe.

Alte aspecte care trebuie luate în vedere la scrierea elegantă a codului:

- **modularitatea programelor** (dimensiune recomandată este de 20 de linii per funcție/procedură), totodată asigurând creșterea lizibilității codului, a posibilităților de reutilizare; se dorește de asemenea obținerea unei cuplări strânse în cadrul aceluiași modul, precum o cuplare cât mai slabă între module diferite din perspectiva proiectării arhitecturale;
- **folosirea de constante/variabile** în loc de valori hard-coded;
- **reutilizarea codului**, evitând astfel situații de repetare de tip „copy paste” a unor secțiuni de cod, care în urma unor modificări ulterioare pot duce la anumite situații dificile de tip debugging;
- **documentarea codului**, comentariile reprezentând un aspect important al scrierii elegante a codului.

Accentul pe Algoritmi

După cum reiese și din denumirea materiei, accentul va fi pus pe partea formală și înțelegerea efectivă a algoritmilor.

Asfel, la nivelul implementării, limbajul ales este la latitudinea fiecăruia. Din perspectiva ușurinței de scriere, a structurilor de date disponibile și a sintaxei se recomandă Java.

Din perspectiva performanțelor obținute, recomandările înclină spre C/C++, cu amendamentul că pentru structuri de bază să fie integrat STL. Mai multe detalii puteți găsi în anexa acestui laborator.

Aplicații de Laborator

Socializare

Obiectiv pentru studenți: cunoașterea asistentului și angajarea într-o discuție liberă. Obiectiv pentru asistenți: familiarizarea cu studenții.






Sugestii:

1. Asistentul se prezintă în câteva minute, într-o manieră pe care dorește să o urmeze și studenții (liceu terminat, specializare la facultate, domenii de interes în Computer Science, hobby-uri, diverse realizări).
2. Prezentarea fiecărui student folosind o abordare similară. Eventual se poate lucra și în echipe de 2 persoane în cadrul cărora fiecare student îl prezintă pe celălalt după principiul – vorbesc cu tine și nu despre tine; 3 categorii clare trebuie să reiasă din fiecare prezentare: latura personală (afinități, caracteristici individuale), profesională (reușite personale) și un secret la latitudinea fiecăruia (principiul că nimeni din cameră nu trebuie să știe acel aspect);
3. Întrebări interesante pentru studenți: dacă au fost la concursuri în liceu, dacă folosesc Infoarena sau Topcoder, dacă au citit CLRS măcar în parte, ce așteptări au de la PA, ce (nu) le-a plăcut până acum în facultate, unde vor să ajungă după facultate, etc.
4. Opțional: Determinarea pentru fiecare persoană a răspunsului la întrebarea: De ce mă bucur eu astăzi? (3 motive), clasificarea în funcție de specific (eveniment, aspect social, personal) și temporal (trecut, prezent, viitor); întrebarea și mai puternică pe care trebuie să ne-o punem apoi – de ce se bucură ceilalți când ne văd?
5. Opțional: Reflectarea influenței personale, necesitatea unei abordări pozitive în tot ceea ce facem; orice gest, atitudine se reflectă în exterior.
6. Opțional: Viziunile determină comportamentul, importanța inovației și încurajarea creativității. Aici exercițiul pentru argumentare este simplu. Avem 2 puncte A și B pe care le stabilim aleator în cameră. Cât de greu este să parcurgem distanța? Acum avem o barmă la nivelul solului. Trebuie mers în echilibru, dar nimic nu ni se poate întâmpla. În final, A și B delimitează spațiul dintre 2 blocuri (o prăpastie), nu poate fi ocolită și nu există sanse de supraviețuire în cazul nefecirii în care te dezechilibrezi? Ce se schimbă între scenarii și ce te-ar face să traversezi? O problemă dragă, o sumă considerabilă de bani, nici băut nu ai trece? Totuși dacă te uiți exclusiv în față vezi exact același lucru, totul ține de percepția și viziunea fiecăruia. Acum pe lângă motiv care implică clar motivația ce te face totuși să traversezi? Peste 90%, în general răspund voința. Adevărat, te poate duce sus și fără ea nu poți face multe. Dar și mai important, pentru a ajunge în vârf ai nevoie de creativitate, inovație, iar domeniul algoritmicii este clar un domeniu cu un potențial nelimitat în care se aplică perfect sintagma „only the sky is the limit”.

Prezentare Generală

Prezentarea așteptărilor și a regulamentului, al modului de desfășurare al laboratoarelor.


Probleme

1. Descărcați scheletul de cod pentru laboratorul 00. Deschideți sursele în editorul sau IDE-ul favorit.
2. În schelet este definită clasa Complex. Sortați descrescător elementele unui vector de numere complexe folosind un maxheap. Criteriul principal de sortare este partea reală. La părți reale egale, criteriul este partea imaginară. Pentru maxheap puteți folosi clasa  PriorityQueue în Java sau  std::priority_queue în C++.
3. Folosind  std::map în C++, sau  HashMap în Java, creați o mapare între elementele din vector și poziția lor în șirul sortat.
4. Bonus: Rezolvați, la alegere, 1-2-3 probleme care vi se par interesante din următoarea prezentare:  importanța algoritmilor pentru interviuri

Anexa

Un scurt tutorial de C++ scris pentru studenții de la PA se găsește  aici. Vă recomandăm să parcurgeți mai ales capitolele introductive și cele referitoare la STL, pentru a dobândi rapiditate în implementare și ușurința de a manipula structurile de date de baza cerute de algoritmi și implementate în limbaj: **vectori**, **liste**, **stive**, **cozi**, **heap-uri**, **map-uri**, **set-uri**, etc.

Referințe

1.  Code Conventions for the Java Programming Language
2.  Google C++ Style Guide
3.  C Coding Standard
4.  C++ Layout And Comments
5.  Brad Abrams, Design Guidelines, Managed code and the .NET Framework
6.  Mozilla Coding Style Guide
7. PHP::PEAR Coding Standards
8. Infoarena
9. VMChecker