Temă Analiza Algoritmilor 2019

Dată publicare: 22.10.2019
Deadline Etapa 0: 03.11.2019
Deadline Etapa 1: 15.11.2019
Deadline Etapa 2: 29.11.2019
Deadline Etapa 3: 13.12.2019

Cuprins

1 Cerință

Realizați un studiu comparativ (orientativ, 6+ pagini) privind principalii algoritmi care rezolvă o problemă aleasă de voi, din lista de mai jos. Studiul va fi realizat pe seturi de date cât mai variate, propuse de voi sau folosite în alte studii similare, care să evidențieze principalele criterii de alegere (avantaje, dezavantaje) pentru algoritmii analizați.

1.1 Timeline

1.1.1 Etapa 0 - 0p

Înscrieți-vă opțiunea pentru problema pe care vreți să o rezolvați în sheetul corespunzător semigrupei voastre. Link înscriere

1.1.2 Etapa 1 - 1p

Pentru prima etapă veți redacta un document de 1-2 pagini, care va reprezenta punctul de plecare pentru etapa finala. Documentul va avea următoarea structură:

10% 1. **Descrierea problemei rezolvate**, respectiv menționarea unei aplicații practice.

20% 2. **Specificarea soluțiilor alese** și eventual detalii relevante legate de implementarea acestora.

60% 3. **Criteriile de evaluare pentru soluția propusă**. Descrieți modalitatea în care ați alege să întocmiți un set de teste de validare a corectitudinii, respectiv a eficienței.

10% 4. **O selecție de referințe** pe care le considerați relevante pentru tema aleasă.

Scopul principal al acestei etape este să primiți sugestii de îmbunătățire referitoare la modalitatea de rezolvare a problemei propuse sau legate de redactarea documentului.

1.1.3 Etapa 2 - 4p

Aceasta etapă va consta din:

60% **Implementarea efectivă a soluțiilor alese**, respectând API-ul minimal specificat în secțiunea problemei alese.

40% **Realizarea unui set de teste** pentru verificarea corectitudinii, respectiv eficienței soluțiilor propuse.

Precizări:

- Încercați să generați teste cât mai variate.
- Temele vor fi verificate automat, pe testele propuse de voi, de colegii voștri care au ales aceeași temă, respectiv pe un set de teste private.
- Cele mai eficiente implementări **realizate de voi** vor primi un **bonus** (0 - 2p).
- Tot pentru **bonus**, temele implementate în C++ vor fi comparate cu cele propuse de colegii voștri. Temele vor fi compilate folosind G++, versiunea $\geq$8.3.0, pe un sistem Linux pe 64 biți. Temele care nu sunt implementate in C++ nu vor beneficia de acest bonus.

- Dacă alegeți să implementați în alt limbaj, documentați in README compilatorul folosit. Submisia va fi compilată/executată tot pe un sistem Linux pe 64 biți.
  - Indiferent de limbaj adăugați un fișier Makefile în arhivă. (pentru un limbaj interpretat, este suficientă o regulă de **run**)
  - Opțiuni compilatoare pentru Java: OpenJDK [0], Oracle [1].
- Testele pot fi generate în orice limbaj.

1.1.4 Etapa 3 - 5p

Pentru această etapă veți finaliza studiul comparativ început la etapa 1. Acesta va fi structurat în felul următor:

1. **Introducere** [10%]
   (a) Descrierea problemei rezolvate. [1%]
   (b) Exemple de aplicații practice pentru problema aleasă. [4%]
   (c) Specificarea soluțiilor alese. [1%]
   (d) Specificarea criteriilor de evaluare alese pentru validarea soluțiilor. [4%]
2. **Prezentarea soluțiilor** [40%]
   (a) Descrierea modului în care funcționeaza algoritmii aleși. [10%]
   (b) Analiza complexitații soluțiilor. [15%]
   (c) Prezentarea principalelor avantaje și dezavantaje pentru soluțiile luate în considerare. [15%]
3. **Evaluare** [40%]
   (a) Descrierea modalitații de construire a setului de teste folosite pentru validare. [9%]
   (b) Menționați specificațiile sistemului de calcul pe care ați rulat testele (procesor, memorie disponibilă). [1%]
   (c) Ilustrarea, folosind grafice/tabele, a rezultatelor evaluării soluțiilor pe setul de teste. [20%]
   (d) Prezentarea, succintă, a valorilor obținute pe teste. Dacă apar valori neașteptate, încercați să oferiți o explicație. [10%]
4. **Concluzii** [6%]
   (a) Precizați, în urma analizei făcute, cum ați aborda problema în practică; în ce situații ați opta pentru una din soluțiile alese. [6%]
5. **Bibliografie** [2%]

Format document [2%]: Pentru realizarea documentului veți folosi formatul LNCS [Hof97]. Mai multe detalii, respectiv template .docx și .tex puteți găsi aici.

2 Lista de Probleme

2.1 Sortare

2.1.1 Descriere

Se dau **N** elemente comparabile (e.g. numere întregi), intr-o ordine oarecare. Se pune problema sortării lor in ordine crescătoare.
Cele mai populare limbaje de programare expun algoritmi diferiți pentru un apel generic de sortare a datelor, în bibliotecile standard. Astfel, putem intâlni în practică abordări distincte, precum algoritmi de sortare prin comparare (MergeSort, QuickSort, IntroSort, TimSort, ShellSort, CombSort etc.) sau care exploatează diferite proprietăți ale datelor de intrare (RadixSort, CountingSort etc.).
Mai multe detalii legate de formatul arhivei pentru etapa 2 găsiți aici.

2.1.2 Precizări

- Comparați **trei algoritmi**.
- Pentru orice algoritm trebuie prezentat cazul cel mai favorabil, mediu și cel mai defavorabil din punct de vedere temporal. Din punct de vedere spațial e suficient să prezentați cazul cel mai defavorabil. Aceste cerințe se aplică doar dacă nu se specifică altceva în particular pentru un algoritm.
- Testați cu date de dimensiuni variate, secvențe sortate sau parțial sortate, secvențe cu elemente identice, secvențe cu tipuri diferite de date (int, double, string, custom types), etc.
- Aveți în vedere și alte motive pentru care ați opta pentru un anumit algoritm în afară de complexitatea propriu-zisă (e.g. stabilitate).

2.1.3 Opțiuni posibile

- **QuickSort**
  - Complexitatea temporală în cazul mediu poate fi doar menționată.
  - Minim 3 strategii de alegere a pivotului (care să îmbunătățească performanța, nu doar să fie diferite între ele).
  - Puteți specifica și alte strategii de evitare a cazului defavorabil.
- **MergeSort**
  - Explicați de ce ar putea fi în practică mai lent decât QuickSort (chiar dacă nu ați ales QuickSort).
  - Precizați de ce complexitatea în cel mai defavorabil, mediu și cel mai favorabil caz e aceeași.
- **HeapSort**
  - Explicați ce avantaje/dezavantaje are în practică fața de QuickSort și/sau MergeSort.
- **ShellSort**
  - Este suficient să tratați cel mai defavorabil caz.

  - Menționați complexitatea pentru diverse liste de gap-uri (minim 3). Dați exemplu de o listă de gap-uri neadecvat aleasă și de ce e neadecvată.
  - Explicați de ce merge mai bine acest algoritm decât un BubbleSort. Hint: Gândiți-vă cum putem măsura gradul de sortare (cât de sortat este) al unei secvențe.
- **CountingSort & RadixSort**
  - Explicați pentru ce fel de inputuri sunt cei mai potriviți acești algoritmi.
  - Explicați cum putem adapta algoritmii să funcționeze și pentru numere cu virgulă (float sau double).
  - Puteți doar menționa complexitatea pentru cazul mediu.
- **Un algoritm hibrid** (IntroSort, TimSort, etc.)
  - Pentru a beneficia de bonus la temă, trebuie să abordați un algoritm hibrid existent sau unul propus de voi.
  - Analizați cazul cel mai favorabil și cel mai defavorabil.
- **Alt algoritm existent**
  - Puteți alege și alți algoritmi în afara celor menționați, cu condiția să sorteze prin comparare cu o complexitate temporală pe cazul mediu de $o(n^2)$.

2.2 Elementul minim dintr-un interval („Range Minimum Query")

2.2.1 Descriere

Dat fiind un vector **A** cu **N** elemente de tip întreg, răspundeți eficient la întrebări de tipul: "Care este elementul minim din intervalul care începe la poziția x și se termină la poziția y?". Se consideră că se dă vectorul și apoi se fac **M** interogări, fără a se modifica între timp vectorul.
Mai multe detalii legate de formatul arhivei pentru etapa 2 găsiți aici.

2.2.2 Precizări

- Comparați **trei variante** de rezolvare.
- Discutați cazurile când M $\ll$ N (M mult mai mic decât N), M proporțional cu N și M $\gg$ N (M mult mai mare decât N).
- Calculați complexitatea în funcție de aceste două variabile (N și M).
- Discutați cum se modifică problema dacă utilizatorul poate să solicite schimbarea valorii unui element de la o anumită poziție în timpul interogărilor pentru fiecare metodă aleasă. (varianta online)

2.2.3 Opțiuni posibile

- Nu se acceptă soluția banală sau variante echivalent de slabe cu O(n) per interogare.
- Una dintre variante trebuie să fie o variantă care se comportă bine pentru cazul când se pot și modifica elemente.
- Una dintre variante trebuie să fie o metodă care răspunde în O(1) la interogare pentru scenariul în care elementele vectorului nu se modifică.

2.3 Cel mai mic strămoș comun („Lowest Common Ancestor")

2.3.1 Descriere

Se dă un arbore **T** cu **N** noduri și apoi se fac **M** interogări astfel: pentru fiecare interogare se dau 2 noduri **u** și **v**. Trebuie să returnați nodul **w** cu cea mai mare adâncime din **T** care este strămoș atât pentru **u**, cât și pentru **v**.
Mai multe detalii legate de formatul arhivei pentru etapa 2 găsiți aici.

2.3.2 Precizări

- Comparați **două variante** de rezolvare.
- Discutați cazurile când M $\ll$ N (M mult mai mic decât N), M proporțional cu N și M $\gg$ N (M mult mai mare decât N).
- Calculați complexitatea în funcție de aceste două variabile (N și M).

2.3.3 Opțiuni posibile

- Nu se acceptă soluția banală sau variante mai slabe ca O(h) per interogare, unde h este înălțimea arborelui.

2.4 Algoritmi de aproximare

2.4.1 Descriere

Rezolvați **o problemă** NP-completă. [Kar72]
Mai multe detalii legate de formatul arhivei pentru etapa 2 găsiți aici.

2.4.2 Precizări

- Pentru fiecare problemă trebuie să comparați cel puțin două soluții:
  1. O soluție care obține întotdeauna rezultatul corect (e.g. backtracking).
  2. Un algoritm care obține un răspuns apropiat de cel corect și consumă o cantitate rezonabilă de resurse (spațiu/timp) pentru orice instanță a problemei. (e.g. o euristică nebanală a cărei complexitate și funcționalitate să puteți să o prezentați).
- Discutați care sunt compromisurile ce trebuie făcute pentru implementare și totodată explicați euristicile aplicate.
- **Este important să măsurați cât de apropiat este rezultatul obținut de soluția voastră de rezultatul corect.**
- Alegeți testele suficient de mici astfel încât să puteți rula un algoritm care garantează un rezultat corect într-un timp rezonabil. Puteți porni de la un set de teste existent.
- (Bonus): Discuție despre ce strategie/combinație de algoritmi ați aplica pentru a rezolva problema în practică. S-ar putea obține rezultate mai bune folosind algoritmul ales dacă am investi mai multe resurse?

2.4.3 Opțiuni posibile

- Problema **comis-voiajorului**.
- Problema **clicii**.
- Problema a**coperirii cu vârfuri**.
- Problema **colorării nodurilor unui graf**.
- **SAT**.
- Alte probleme **NP-complete**. [Kar72]

2.5 Drumuri minime in graf

2.5.1 Descriere

Se dă un graf ponderat (particularitățile grafului schimbă rezolvarea: poate fi orientat, neorientat, cu costuri negative pe muchii, aciclic etc.) se cere:

- Costul minim de la un nod la toate celelalte.
  - Format arhivă etapa 2.
- Costul minim între oricare 2 noduri din acest graf.
  - Format arhivă etapa 2.

2.5.2 Precizări

- Comparați cel puțin **trei algoritmi** pentru **una** dintre cele două probleme menționate mai jos.
- Specificați pentru fiecare algoritm pe ce fel de graf (cel mai general) poate fi aplicat și pe ce fel de graf nu poate fi aplicat și de ce (ex.: putem aplica Dijkstra pe grafuri cu costuri negative?, de ce?, dar Floyd-Warshall-Roy?).
- Pentru punctaj maxim este suficient să analizați pentru grafuri orientate.

2.5.3 Opțiuni posibile

- Costul minim de la un nod la toate celelalte (patru algoritmi):
  - **Dijkstra**: (specificați complexitatea cu/fără heapuri)
  - **Bellman-Ford**: (cu coadă, de ce e mai rapid cel cu coadă, deși complexitatea temporală e aceeași?)
  - **Dijkstra adaptat** pentru costuri foarte mici ale muchiilor/arcelor.
  - **Cea mai eficientă metodă pentru grafuri orientate aciclice** (explicați de ce nu putem aplica metoda asta pentru grafuri care sunt ciclice).
- Costul minim între oricare două noduri (patru algoritmi):
  - **Floyd-Warshall**: Explicați de ce nu putem schimba ordinea for-urilor fără a schimba și codul din interiorul blocului interior celor 3 for-uri.
  - **Dijkstra** sau **BellmanFord** (aplicate între oricare două noduri)
  - **Johnson**: Explicați de ce poate fi folosit pe grafuri cu muchii de cost negativ.

2.6 Structuri de date

2.6.1 Descriere

Fiecare structură de date are operațiile ei specifice. Sunt precizate pentru fiecare în parte mai jos. Mai multe detalii legate de formatul arhivei pentru etapa 2 găsiți aici.

2.6.2 Precizări

- Alegeți fie cele **două structuri** pentru mulțimi, fie cele **două structuri** pentru cozi de prioritate.

2.6.3 Opțiuni posibile

- **Mulțimi**
  - Operații de implementat: adăugarea unui element, ștergerea unui element, modificarea unui element.
  - Aici aveți de analizat **2 structuri**: **tabele de dispersie** (alegeți ce metode vreți pentru tratarea coliziunilor) și **arbori binari de căutare echilibrați** (AVL, Treap-uri etc. - e la alegerea voastră). Pentru fiecare structură trebuie implementate toate operațiile precizate mai sus.
  - Pentru tabele de dispersie trebuie să faceți o analiză a metodelor de evitare a coliziunilor. Este suficient să explicați în ce situații sunt mai potrivite unele decât altele (înlănțuire vs. adresare deschisă). Explicați cum puteți trata cazul când nu știți dinainte toate operațiile care vor fi aplicate structurii (nici măcar un număr aproximativ al lor, așa zisa variantă online). Explicați de ce nu există această problemă și în cazul arborilor. Explicați ce proprietăți ar trebui să aibă o funcție hash ca să fie considerată bună. Precizați doar complexitatea cazului mediu și a celui defavorabil, cu o scurtă argumentare pentru fiecare (referitor doar la complexitatea temporală).
  - Pentru arbori binari de căutare echilibrați trebuie argumentat doar pe scurt complexitatea temporală pentru cazul cel mai defavorabil.
  - În general, ar trebui să faceți o comparație între cele două structuri. Menționați ce avantaje are una față de cealaltă (complexitate, ce operații suportă în plus (în mod eficient), adaptabilitate).
- **Cozi de prioritate**:
  - Operații de implementat: adăugarea unui element, obținerea și ștergerea elementului minim/maxim.
  - Aici aveți de implementat **2 structuri**: **arbori binari echilibrați** (AVL, Treap-uri etc.) și **heap-uri** (puteți alege orice tip de heap).
  - Comparați cele două structuri de date între ele (avantaje/dezavantaje).

2.7 Algoritmi pentru procesarea șirurilor de caractere

2.7.1 Descriere

Nu este o problemă particulară, fiecare algoritm e asociat cu o problemă.
Mai multe detalii legate de formatul arhivei pentru etapa 2 găsiți aici.

2.7.2 Precizări

- Alegeți **doi** algoritmi din cei enumerați mai jos.

2.7.3 Opțiuni posibile

- Care este cea mai lungă secvența comună a două șiruri de caractere? (aici sunt mai mulți algoritmi posibili, nu trebuie implementați toți) Ce complexitate ar avea problema dacă am ști că cea mai lungă secvență comună conține maxim M caractere? Exprimați complexitatea în funcție de dimensiunile șirurilor și M și descrieți soluția pentru acest caz particular.
  - **Knuth-Morris-Path (KMP)** Găsiți două probleme care pot fi făcute cu acest algoritm eficient, dar nu și cu Rabin-Karp.
  - **Rabin-Karp** Precizați probabilitatea unei false potriviri în funcție de dimensiunea spațiului cheilor. Argumentați cum putem îmbunătăți această probabilitate (să fie cât mai mică). Discutați ce tipuri de funcții hash putem folosi în implementare.
  - **Boyer-Moore**
- **Aho-Corasick**: Argumentați de ce nu putem rezolva la fel de eficient problema asociată cu acest algoritm cu KMP (Knuth-Morris-Path). Discutați care este asemănarea între acest algoritm și KMP.
- **Trie**: Discutați când este bine să folosim acestă structură de date, când e mai eficient un simplu multiset.
- **Edit distance**: distanța Levenshtein sau alte distanțe. Analizați minim două astfel de distanțe la acest subiect.

2.8 Identificarea numerelor prime

2.8.1 Descriere

Fiind dat un set de date de intrare se dorește identificarea numerelor prime din acel set.
Mai multe detalii legate de formatul arhivei pentru etapa 2 găsiți aici.

2.8.2 Precizări

- Comparați **doi algoritmi**.

2.8.3 Opțiuni posibile

- **Fermat**: Argumentați de ce algoritmul poate întoarce true pentru numere compuse (care nu sunt prime) și cum influențează numărul de iterații acest lucru. Funcționează întotdeauna folosirea unui număr mare de iterații? Justificați.
- **Miller-Rabin**: Comparați algoritmul cu Fermat și evidențiați cazurile în care se preferă utilizarea Miller-Rabin.
- **Solovay-Strassen**: Argumentați de ce algoritmul poate întoarce true pentru numere compuse (care nu sunt prime).
- **Frobenius**: Evidențiați caracteristicile acestui algoritm prin comparație cu celălalt algoritm ales.

2.9 Compresia datelor

2.9.1 Descriere

Cerința este să comprimați, respectiv să decomprimați, **fără pierdere de informații**, un fișier.
Mai multe detalii legate de formatul arhivei pentru etapa 2 găsiți aici.

2.9.2 Precizări

- Trebuie să comparați **doi algoritmi** cunoscuți (e.g. Huffman coding, Lempel-Ziv-Welch (LSW), Arithmetic Coding, etc.) sau variante ale acestora.
- Pentru a evalua performanța algoritmilor trebuie să analizați cel puțin:
  - Resursele consumate în timpul procesului de comprimare/decomprimare.
  - Dimensiunea inițială a fișierului vs. dimensiunea fișierului comprimat.
- (Bonus) Puteți să propuneți propria soluție adaptivă, care ține cont de tipul de fișier primit la intrare (text, imagine, audio, etc.) (e.g. "Huffman coding" optimizat pentru texte scrise în limbaj natural) sau un algoritm cunoscut care combină mai multe tehnici.

3 Sugestii redactare

- **Evitați afirmațiile vagi, neargumentate**. (”după cum se spune”, ”cel mai folosit algoritm”, ”cel mai bun algoritm”)
- Dacă preluați o informație, tabel, poza, grafic, etc. **este obligatoriu** să specificați sursa, respectiv să folosiți ghilimele atunci când preluați un paragraf nemodificat.
- Ca să fiți siguri că performanțelor obținute nu sunt întâmplătoare, este util să rulați de mai multe ori pe același set de teste și să calculați o medie a valorilor obținute.
- Tot pentru o testare riguroasă, este important să măsurați separat rezolvarea problemei de partea de IO a datelor.
- Puteți fi penalizați pentru folosirea unor termeni fară explicarea semnificației lor. (e.g.: Complexitate O(n + r) - fără a specifica cine este ’r’)
- Evitați folosirea unor expresii repetitive (”Asadar.., asadar..”, ”Deci.., deci..”) în aceeasi frază/paragraf.
- La prezentarea algoritmului evitați să puneți prea mult cod direct in lucrare. Ar fi preferabil să prezintați (succint) în cuvinte comportamentul algoritmului (eventual folosind pseducod) pentru ca cineva care nu știe algoritmul să poată înțelege ușor despre ce este vorba fără să fie nevoit să urmărească secvențe lungi de cod.
- Pentru a fi ușor de înteles, graficele trebuie să aibă o legendă, respectiv să aibă valorile normalizate.
- Într-un tabel în care comparați performanța unor algoritmi marcați cu bold cel mai bun rezultat.
- **Specificați unitățile de măsură folosite in tabele/grafice**.
- Pentru editarea referințelor, puteți folosi funcția ”cite” din Google Scholar. Mai multe detalii aici.
- Dacă doriți să adăugați o referință la un website, menționați data ultimei accesări.
- Nu folosiți Wikipedia ca referință.

4 Trimiterea Temei

- **Etapa 0**

  Înscrieți-vă opțiunea în sheetul corespunzător semigrupei voastre. Link înscriere

- **Etapa 1**

  Veți încărca pe Moodle o arhivă care va conține documentul în format .pdf. Documentul se va numi ”**NumărSubiect.pdf** unde în loc de NumărSubiect veți trece numărul subiectului ales (1 pentru sortări, 2 pentru RMQ (ș.a.) și 10 pentru orice alt subiect care nu e în lista de probleme)”.

  Deoarece soluțiile vor fi repartizate pentru corectare automat, arhiva va fi denumită, **obligatoriu**, folosind ID-ul vostru de Moodle. De exemplu, daca ID-ul vostru este ”ion.popescu”, arhiva **trebuie** să se numească ”ion.popescu.zip”

- **Etapa 2**

  Veți încărca pe Moodle o arhivă care va conține:

  - **Sursele pentru algoritmii utilizați**, respectând structura sugerată la fiecare problemă.
  - **Setul de date de test**:
    1. Testele de intrare se vor afla într-un folder numit **"in"**.
       Fiecare test se va numi "testX.in", unde X reprezintă ID-ul testului. (ex: "test1.in")
    2. Rezultatele corecte pentru fiecare test se vor afla într-un folder numit **"out"**.
       Fiecare rezultat se va numi "testX.out", unde X reprezintă ID-ul testului.
       (ex: "test1.out")
  - **Programul folosit pentru generarea testelor**. (Opțional)
  - Fișierul **Makefile**. Acesta trebuie să conțină obligatoriu reguli de build și clean.
  - Fișierul **README**, în care să menționați ce reprezintă fiecare fișier din arhivă sau alte detalii relevante legate de evaluare. De asemeni, **specificați dacă ați preluat o secvența de cod dintr-o sursă externă**.

  Deoarece soluțiile vor fi repartizate pentru corectare automat, arhiva va fi denumită, **obligatoriu**, folosind ID-ul vostru de Moodle. De exemplu, daca ID-ul vostru este "ion.popescu", arhiva **trebuie** să se numească "ion.popescu.zip"

  Puteți sa verificați **un exemplu** de arhivă pentru etapa 2 aici.

- **Etapa 3**

  Veți încărca pe Moodle o arhivă care va conține documentul final în format .pdf. Documentul se va numi "**EtapaFinala.pdf**".

  Deoarece soluțiile vor fi repartizate pentru corectare automat, arhiva va fi denumită, **obligatoriu**, folosind ID-ul vostru de Moodle. De exemplu, daca ID-ul vostru este "ion.popescu", arhiva **trebuie** să se numească "ion.popescu.zip"

5 Deadlines

1. **Etapa 0**: **03.11.2019, 23:55**
2. **Etapa 1**: **15.11.2019, 23:55**

   Deadline-ul acestei etape este **hard**! Astfel, veți putea primi feedback la timp care să vă ajute pentru etapele următoare.
3. **Etapa 2**: **29.11.2019, 23:55**

   Acesta etapă va avea un deadline **soft**, cu penalizări de 5% din punctaj pentru fiecare zi de întârziere. Penalizările sunt limitate la cel mult 30% din punctajul total alocat etapei. Deadline-ul hard al acestei etape coincide cu cel pentru etapa 3.
4. **Etapa 3**: **13.12.2019, 23:55**

   Acesta etapă are un deadline **hard**!

6 Alte Menționi

1. În funcție de dificultatea conceptelor explorate și calitatea lucrării, puteți primi puncte bonus nespecificate (exemplu: pentru demonstrarea corectitudinii algoritmilor sau implementarea unui algoritm/unei structuri în plus față de numărul minim).
2. Recomandăm să folosiți tool-uri de profiling. (perf, valgrind, etc.).
3. Pentru clarificări suplimentare puteți să puneți întrebări pe forum.
4. Daca nu ați submis la o etapă puteți trimite în continuare pentru etapa următoare, respectiv puteți trimite oricând mai devreme soluțiile pentru o etapă. Deadline-urile pentru etapele 1 si 3 sunt **hard**.
5. La fiecare etapă puteți îmbunătăți soluția de la etapa precedentă (e.g. evaluare pe teste noi la etapa 3, modificare document dupa etapa 1, etc.)

Bibliografie

[Hof97] Alfred Hofmann, *Lecture notes in computer science: Authors' instructions for the preparation of camera-ready contributions to lncs/lnai/lnbi proceedings.*, `www.springer.com/?SGWID=4-102-45-72797-0`, 1997.

[Kar72] Richard M Karp, *Reducibility among combinatorial problems*, Complexity of computer computations, Springer, 1972, pp. 85–103.