Temă

1 Complexitatea comparării a două numere

- **Dimensiune uniformă.** Complexitate: $O(1)$
- **Dimensiune logaritmică.** Parcurgem simultan biții celor două numere, începând cu cei mai semnificativi (poziția $\lfloor \log_2 \max(a, b) \rfloor$), până când găsim doi biți diferiți. Dacă bitul din a este mai mic decât bitul din b, înseamnă că $a < b$. Dacă bitul din a este mai mare decât bitul din b, atunci $a > b$. Dacă am ajuns la poziția 0 fără să găsim vreo pereche de biți distincți, concluzionăm că $a = b$. Complexitate: $O(\log \max(a, b))$
- **Dimensiune liniară.** Inițializăm un contor cu 0, și îl incrementăm succesiv până când atinge valoarea uneia dintre variabilele date. Aceasta va fi $\min(a, b)$. Dacă ambele valori au fost atinse simultan, atunci $a = b$. Complexitate: $O(\min(a, b))$

2 Complexitatea înmulțirii a două numere

- **Dimensiune uniformă.** Complexitate: $O(1)$
- **Dimensiune logaritmică.** Un algoritm simplu pentru înmulțirea a două numere, ce se folosește de reprezentările acestora într-o bază arbitrară `BASE`, este următorul (descris în limbajul Alk):

```
prod(a, b, BASE) {
    c = [0 | k from [1..a.size() + b.size()]];
    for (i = 0; i < a.size(); i++)
        for (j = 0; j < b.size(); j++)
            c[i + j] += a[i] * b[j];
    for (k = 0; k < a.size() + b.size() - 1; k++)
        if (c[k] >= BASE) {
            c[k + 1] += c[k] / BASE;
            c[k] %= BASE;
        }
    while (c[c.size() - 1] == 0)
        c.popBack();
    return c;
}
```

  Practic, algoritmul constă în a lua fiecare pereche de cifre (i, j), cu i din a și j din b, și a aduna la c valoarea $a[i] \cdot b[j] \cdot \texttt{BASE}^{i+j}$. Complexitate: $O((\log a)(\log b))$

  Există însă și algoritmi mai avansați pentru înmulțirea a două numere întregi, cum ar fi Algoritmul lui Karatsuba, în $O(n^{\log_2 3})$, sau Fast Fourier Transform, în $O(n \log n)$, unde $n = \log \max(a, b)$.

- **Dimensiune liniară.** Inițializăm c cu 0. Iterăm un i de la 1 la a. Pentru fiecare i, mai iterăm un j de la 1 la b, iar la fiecare pas incrementăm rezultatul. Complexitate: $O(a \cdot b)$

3 Complexitatea operațiilor pe mulțimi

Notăm cu $\mathrm{time}(x)$ timpul pentru copierea valorii lui x într-o altă variabilă și respectiv cu $\mathrm{time}(x, y)$ timpul pentru compararea valorilor x și y, în funcție de metoda aleasă (uniformă, logaritmică sau liniară).

- **Reuniune.** Algoritmul pentru reuniune ($C \mapsto A \cup B$) este:

  1. Copiem pe A în C, element cu element.
  2. Pentru fiecare x din B:
  3. Îl inserăm pe x în C.

  Complexitate: Indiferent de modul în care sunt reprezentate mulțimile, primele două linii au complexitățile $O(\sum_{x \in A} \mathrm{time}(x))$ și respectiv $O(|B|)$. În cazul celei de-a treia linii, avem de analizat mai multe situații:

  a) **Liste.** Îl comparăm pe x cu fiecare element din C, iar dacă nu îl găsim, îl adăugăm la finalul listei: $O(\sum_{x \in B} \sum_{y \in A \cup B} \mathrm{time}(x, y))$.
     - **Uniform:** $O(\sum_{x \in B} \sum_{y \in A \cup B} 1) = O(|B| \cdot |A \cup B|) = O(|B|(|A| + |B|))$.
     - **Logaritmic:** $O(\sum_{x \in B} \sum_{y \in A \cup B} \log \max(x, y)) = O(\log \prod_{x \in B} \prod_{y \in A \cup B} \max(x, y)) = O(\log(\max(A \cup B))^{|B| \cdot |A \cup B|}) = O(|B|(|A| + |B|) \log \max(A \cup B))$.
     - **Liniar:** $O(\sum_{x \in B} \sum_{y \in A \cup B}(x + y)) = O(|A \cup B| \sum_{x \in B} x + |B| \sum_{y \in A \cup B} y) = O((|A| + |B|) \sum_{y \in B} y + |B|(\sum_{x \in A} x + \sum_{y \in B} y)) = O(|B| \sum_{x \in A} x + (|A| + |B|) \sum_{y \in B} y)$.

     În cazul listelor, linia 3 este cea mai costisitoare, celelalte două neinfluențând complexitatea finală.

  b) **Tabele de hashing.** Testăm în timp amortizat $O(\mathrm{time}(x))$ dacă x se află în tabela de hashing corespunzătoare lui C. În caz că nu, îl inserăm tot în $O(\mathrm{time}(x))$ amortizat. Complexitățile finale:
     - **Uniform:** $O(|A| + |B|)$ amortizat.
     - **Logaritmic:** $O(\log(\prod_{x \in A} x)(\prod_{y \in B} y))$ amortizat.
     - **Liniar:** $O(\sum_{x \in A} x + \sum_{y \in B} y)$ amortizat.

  c) **Arbori binari de căutare echilibrați.** În acest caz, inserarea lui x se produce în timp logaritmic (în raport cu înălțimea arborelui): $O(\sum_{x \in B} (\log |A \cup B|) \mathrm{time}(\max(A \cup B), x)) = O(\mathrm{time}(\max(A \cup B), \max B) \log(|A| + |B|)^{|B|} = O(\mathrm{time}(\max(A \cup B))|B| \log(|A| + |B|))$. Complexitățile finale:
     - **Uniform:** $O(|A| + |B| \log(|A| + |B|))$.
     - **Logaritmic:** $O((\log \prod_{x \in A} x) + (\log \max(A \cup B))|B| \log(|A| + |B|))$.
     - **Liniar:** $O((\sum_{x \in A} x) + \max(A \cup B)|B| \log(|A| + |B|))$.

- **Intersecție.** Algoritmul pentru intersecție ($C \mapsto A \cap B$) este:

  1. Pentru fiecare x din A:
  2. Îl căutăm pe x în B.
  3. Dacă l-am găsit, îl inserăm în C.

  Complexitate: Indiferent de modul în care sunt reprezentate mulțimile, prima linie are complexitatea $O(|A|)$. În cazul celorlalte două linii, avem de analizat mai multe situații:

  a) **Liste.** Pe linia 2, căutăm în $O(\sum_{y \in B} \mathrm{time}(y))$ numărul x în B. În caz că l-am găsit, linia 3 se va putea executa în $O(\mathrm{time}(x))$, căci pe x îl putem adăuga la finalul listei C. Complexitatea finală în cazul uniform este $O(|A| \cdot |B|)$.

  b) **Tabele de hashing.** Testăm în timp amortizat $O(\mathrm{time}(x))$ dacă x se află în tabela de hashing corespunzătoare lui B. În caz că da, îl inserăm tot în $O(\mathrm{time}(x))$ amortizat în C. Complexitatea finală în cazul uniform este $O(|A|)$ amortizat.

  c) **Arbori binari de căutare echilibrați.** În acest caz, căutarea lui x în B și respectiv eventuala sa inserare în C se produc în timp logaritmic (în raport cu înălțimea arborilor). Complexitatea finală în cazul uniform este $O(|A|(\log |B| + \log |A \cap B|)) = O(|A| \log |B|)$.

- **Diferență.** Algoritmul pentru diferență ($C \mapsto A \setminus B$) este:

  1. Pentru fiecare x din A:
  2. Îl căutăm pe x în B.
  3. Dacă **nu** l-am găsit, îl inserăm în C.

  Complexitate: Analog cu operația de intersecție.