ALGORITMUSOK ÉS ADATSZERKEZETEK

A tantárgy segédanyaga!

Elméleti tudnivalók, elméleti feladatlapok
Gyakorlati feladatlapok
Interaktív Videók
Tesztek

Osztián Erika
Előadás, gyakorlat
osztian@ms.sapientia.ro

Novák (Osztián) Pálma Rozália
Gyakorlat
osztian.palma@ms.sapientia.ro

ALGORITMUSOK ÉS ADATSZERKEZETEK

A szerszámosláda

A szerszámos táska: feladatok tárháza

TARTALOMJEGYZÉK

TARTALOMJEGYZÉK

TARTALOMJEGYZÉK

TARTALOMJEGYZÉK

Tartalomjegyzék ábrázolva

Tömb

```
+----+----+----+----+----+
| A0 | A1 | A2 | A3 | A4 |...
+----+----+----+----+----+
  0    1    2    3    4   <-- Indexek
```

Gyors elérés, **fix méret**

Lista

[Head] → [Node1] → [Node2] …→ NULL

Din. méret, **lassú keresés**

Verem

```
TOP
| A4 |
| A3 |
| A2 |
| A1 |
| A0 |
```

Egyszerű impl., **csak az utolsó elem**

Sor

[Front] → [A0] → [A1] → [A2] → [A3] → [A4]… → [Rear]

Valós alk., **lassú kersés**

Fa

```
        (8)
       /   \
    (3)    (10)
   /   \      \
 (1)   (6)   (14)
      /   \   /
    (4)  (7)(13)
```

Gyors keresés, **lassú beszúrás, törlés**

Hasító tábla

```
Index:  0    1    2    3    4    5
       ---  ---  ---  ---  ---  ---
Key:   K1   K2   K3    -   K4   K5
Value: V1   V2   V3        V4   V5
```

Gyors keresés, **ütközések**

Tartalomjegyzék ábrázolva

Rope (Kötél)
Szövegkezelés optimalizálása.

```
  [Root (50 karakter)]
       /              \
 [25 karakter]       [25 karakter]
  /       \         /        \
[12]      [13]    [10]       [15]
```

Skip List (Ugrólista) – Gyorsabb keresés láncolt listákban

```
Szint 3:  Head →---------------------→ 50 →---------------------→ NULL
Szint 2:  Head →------→ 10 →------→ 30 →------→ 50 →------→ 70 → NULL
Szint 1:  Head →  5 → 10 → 20 → 30 → 40 → 50 → 60 → 70 → 80 → NULL
```

Treap –
Bináris keresőfa +
Heap kombinációja

```
     (20, p=17)
        /      \
 (10, p=30)  (30, p=25)
     /
 (5, p=40)
```

…

…

palmacademy.education@gmail.com

01 BEVEZETŐ. ALGORITMUSOK MŰVELETIGÉNYE

TUDOD-E?

AZ ALGORITMUS...

Egy probléma megoldására alkalmas utasítássorozat.

Részletes útmutatás.

Recept.

AZ ADATSZERKEZET...

Adatok tárolására, szervezésére és manipulálására szolgáló strukturális, formai elrendezés.

ALGORITMUS + ADATSZERKEZET

Elégedett felhasználó...

Algorithms + Data Structures = Programs.

MIÉRT HASZNOS?

20%

Hely- és idő- komplexitás

20%

Hatékonyság

20%

Teljesítmény

20%

Kód olvashatóság

20%

Problémamegoldás

“

DATA STRUCTURES AND ALGORITHMS ARE AMONG THE MOST IMPORTANT INVENTIONS OF THE LAST 50 YEARS, AND THEY ARE FUNDAMENTAL TOOLS THAT EVERY PROGRAMMER SHOULD HAVE IN THEIR TOOLKIT.

— Eric S. Roberts

ADATSZERKEZETEK OSZTÁLYOZÁSA

Elrendezés szerint

- Lineáris és nem-lineáris

Működés szerint

- Rekurzív és nem-rekurzív

Hozzáférés szerint

- Elsődleges és másodlagos

Megvalósítás szerint

- Konkrét és absztrakt

Memória szerint

- Statikus és dinamikus

Adatok típusa szerint

- Homogén és heterogén

EGY ALGORITMUS IDŐBONYOLULTSÁGA

A futási idő hossza, amely szoros összefüggésben van a bemenet méretével.

IDŐBONYOLULTSÁG MÉRÉSE

Big-O

Felső korlát

(O)

Omega

Alsó korlát

(Ω)

Theta

Éles korlát

(θ)

IDŐ-BONYOLULTSÁG TÍPUSOK

```c
#include <stdio.h>


int main()
{
    int number;
    printf("Enter an integer number:");
    scanf("%i", &number);
    printf("The value of the number is: %i", number);
    return 0;
}
```

Konstans O(1)

```c
#include <stdio.h>

int main() {
    int array[7] = {1, 2, 3, 4, 5, 6, 7};
    printf("The elements of the array are:");
    for (int i = 0; i < 7; ++i) {
        printf("array[%i]: %i ", i, array[i]);
    }
    return 0;
}
```

Lineáris O(n)

```
int binarySearch(int array[], int x, int low, int high) {
  if (high >= low) {
    int mid = low + (high - low) / 2;
    // If found at mid, then return it
    if (array[mid] == x)
      return mid;
    // Search the left half
    if (array[mid] > x)
      return binarySearch(array, x, low, mid - 1);
    // Search the right half
    return binarySearch(array, x, mid + 1, high);
  }
  return -1;
}
```

Logaritmikus

O(logn)

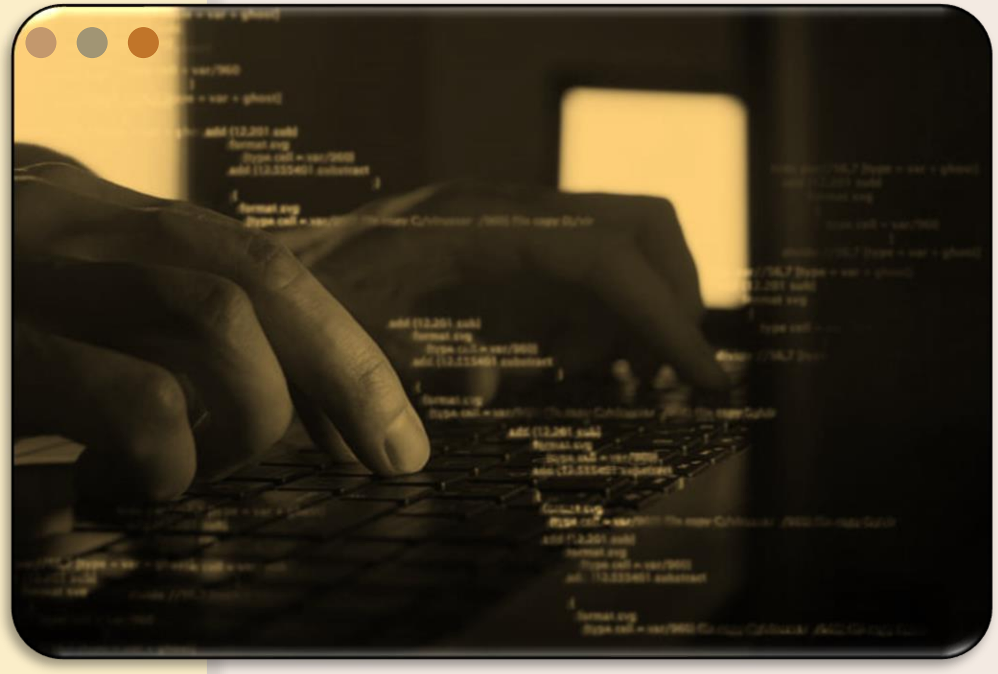
```
for(int i = 0; i < n; ++i)
{
    for(int j = 0; j < n; ++j)
    {
        printf("%i ", matrix[i][j]);
    }
    printf("\n");
}
```

Négyzetes

$O(n^2)$

ALGORITMUSOK MŰVELETIGÉNYÉNEK KISZÁMÍTÁSA

Pl: értékadások-, összehasonlítások-, összeadások száma



```
int main() {
   int N = 10, sum = 0;  // 2 assignment
   // i = 1 -> 1 assignment
   for (int i = 1; i <= N; i++) {
      // (i <= N) -> 1 comparison,
      // (i++)    -> 1 addition, 1 assignment
      sum = sum + i;  // 1 addition, 1 assignment
   }
   return 0;
}
```

Értékadások: 3 + 2*N

Összehasonlítások: N

Összeadások: 2*N

BIG O CHEATSHEET

Időbonyolultság

Adatszerkezetek és algoritmusok idő bonyolultsága

Hely igény

Az algoritmus futtatása során használt memóriaterület mennyiségének meghatározása

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Merge sort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Tim sort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cube sort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

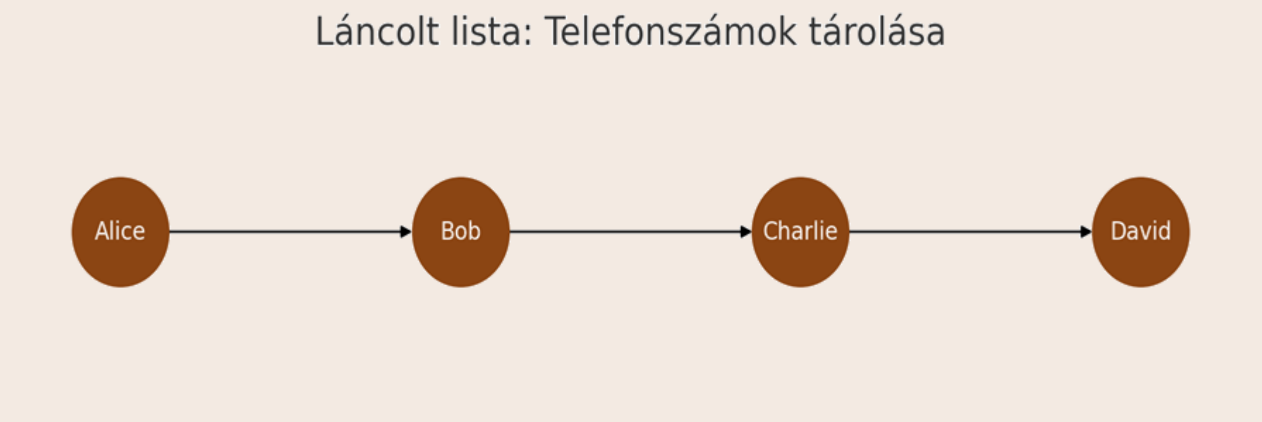
Példa-időhatékonyságra

Feladat: Egy alkalmazásban a felhasználók névjegyeket tárolnak, és gyakran szeretnék megkeresni egy adott személy telefonszámát. Az alkalmazás a következő műveleteket támogatja:

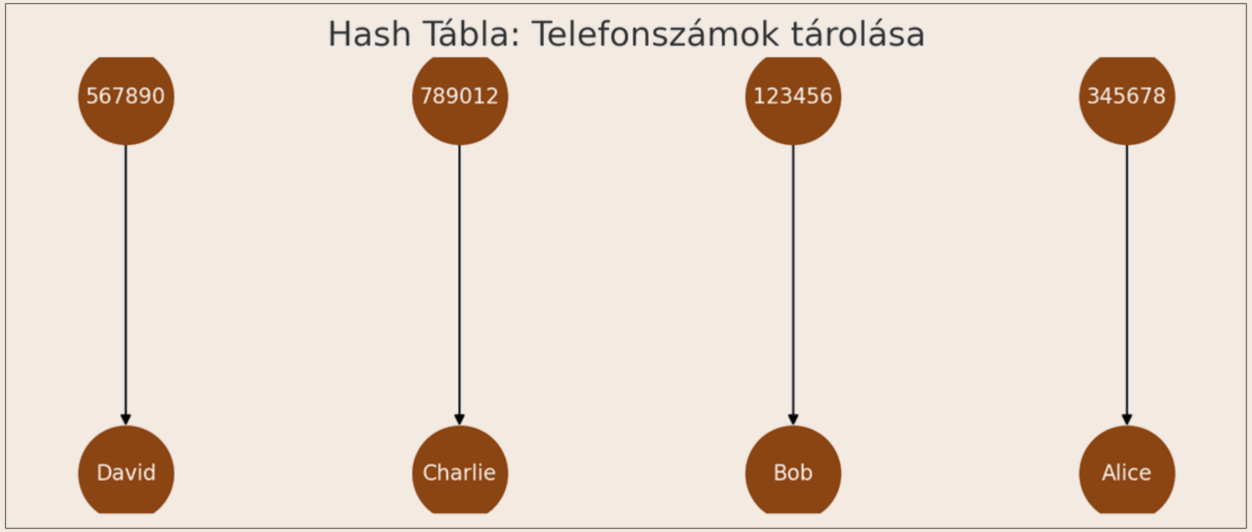
1. **Új névjegy hozzáadása**
2. **Névjegy törlése**
3. **Névjegy keresése név alapján**

Megoldás1: Egyszerű láncolt lista (Unsorted Linked List)

- **Beszúrás: O(1)** idő, mert az új elemet egyszerűen a lista elejére lehet tenni.
- **Törlés: O(n)** idő, mert végig kell **menni** a **listán**, hogy megtaláljuk az eltávolítandó elemet.
- **Keresés: O(n)** idő, mert végig kell nézni az összes bejegyzést, amíg megtaláljuk a keresettet.



A megoldás ábrázolva



Megoldás2: Hasító tábla (Hash Table) használata

- **Beszúrás: O(1)** idő, mert a kulcs alapján azonnal elhelyezzük a táblában.
- **Törlés: O(1)** idő, mert a kulcs alapján azonnal megtaláljuk és eltávolítjuk az elemet.
- **Keresés: O(1)** idő, mert a kulcs (név) alapján azonnal megkapjuk az eredményt.

A hash érték egy **polinomiális összegként** számítható ki:

$$\text{hash}(\text{"}Alice\text{"}) = (65 \times 31^4) + (108 \times 31^3) + (105 \times 31^2) + (99 \times 31^1) + (101 \times 31^0)$$

ALGORITHMS AND DATA STRUCTURES ARE THE POET'S TOOLS IN THE REALM OF PROGRAMMING, COMPOSING SYMPHONIES OF **EFFICIENCY** AND **ELEGANCE** WITHIN THE ORCHESTRA OF CODE.

— ChatGPT