

Hierarchikus bináris keresőfa

Hierarchical Binary Search Tree

Dr. VARGA Imre
egyetemi docens

Debreceni Egyetem, Informatikai Kar,
4028 Debrecen, Kassai út 26.
varga.imre@inf.unideb.hu

Abstract

The memory-efficient storage and lexicographical sorting of n -tuples (multi-dimensional data records) pose a significant challenge in modern data processing. This paper introduces a novel hybrid data structure, the Hierarchical Binary Search Tree (HBST), which combines the advantages of prefix trees and binary trees to eliminate the redundancy of repetitive data prefixes. Due to its structural design, the HBST radically optimizes space complexity in narrow value ranges while guaranteeing fast, logarithmic-time search and insertion.

Keywords: binary search tree, multilevel recursion, n -tuples, HBST

Kivonat

A nagyméretű, többdimenziós adatrekordok (rendezett n -esek) memóriahatékony tárolása és lexikografikus rendezése komoly kihívás a modern adatfeldolgozásban. Jelen tanulmány egy új hibrid adatszerkezetet, a hierarchikus bináris keresőfát (HBKF) mutatja be, amely a prefix-fák és a bináris fák előnyeit ötvözve küszöböli ki az ismétlődő adatelőtagok okozta redundanciát. A struktúra felépítéséből adódóan a HBKF szűk értéktartományok esetén radikálisan optimalizálja a tárkomplexitást, miközben mindvégig garantálja a gyors, logaritmikus idejű keresést és beszúrást.

Kulcsszavak: bináris keresőfa, többszintű rekurzió, rendezett n -es, HBKF

1. BEVEZETÉS

A nagyméretű, többdimenziós adathalmazok hatékony tárolása, keresése és rendezése az informatika egyik klasszikus, ám mind a mai napig aktuális kihívása. Legyen szó relációs adatbázisok indexeléséről, adatbányászati feladatokról, vagy az adatvédelem (data privacy) területén felmerülő k-anonimitás vizsgálatáról, gyakran találkozunk olyan adatrekordokkal, amelyek $(x_1, \dots, x_n)$ formátumú rendezett n -esekként írhatók le. Ezen adathalmazok feldolgozása során alapvető elvárás, hogy a többször ismétlődő rekordokat vagy a közös előtaggal (prefixszel) rendelkező adatsorokat memóriahatékonyan tudjuk tárolni, miközben a gyors kereshetőség és az elemek gyakoriságának megállapítása is biztosított marad [1].

Az n -esek lexikografikus rendezésére számos jól ismert algoritmus (például a Quick sort vagy a Merge sort módosított változatai) létezik [2]. Bár ezek az eljárások $O(N \times \log N)$ időkomplexitással képesek sorbarendezni az adatokat, önmagukban nem hoznak létre olyan perzisztens, kereshető adatszerkezetet, amely alkalmas lenne az ismétlődő prefixek tömörítésére, a gyakoriságok azonnali lekérdezésére, vagy az adathalmaz folyamatos, dinamikus bővítésére az adatok újrendezése nélkül.

A feladatra kézenfekvő megoldásnak tűnhetnek a különböző fafajta adatszerkezetek, azonban a többdimenziós rekordok esetében ezek gyakran kompromisszumokat követelnek. A trie (avagy prefix-fa vagy lexikográfiai fa) rendkívül hatékony karakterláncok tárolására, mivel a közös prefixek egyetlen útvonalon osztoznak [3]. Hátránya azonban, hogy ha az n -esek elemei nem egy szűk ábécéből, hanem egy szélesebb értéktartományból (pl. lebegőpontos számok vagy 32 bites egészek) származnak, a fa minden csomópontjában hatalmasra nőhet az elágazási tényező. Ez üres mutatók tömegét eredményezi, ami drasztikus memóriapazarláshoz vezet.

Ezt a problémát kiküszöbölhetjük egymásba ágyazott leképezések (nested maps) használatával, ahol a ritka ágakat dinamikus struktúrák (pl. hash táblák) kezelik. Ezzel memóriát takarítunk meg, ám a hash-alapú megközelítés miatt elveszítjük a beépített rendezettséget, így lehetetlenné válik az adatok hatékony lexiko-

grafikus kinyerése. A többdimenziós adatok térbeli lekérdezésére optimalizált tartományfák (range trees) kiválóak az összetett [4], dimenziókon átívelő intervallum-keresésekhez, ugyanakkor a struktúra felépítése és karbantartása indokolatlanul bonyolult és erőforrás-igényes olyan esetekben, amikor elsősorban csak pontos prefix-egyeztetésekre és gyakoriságszámlálásra van szükségünk.

A fentiek alapján látható, hogy szükség van egy olyan hibrid adatszerkezetre, amely ötvözi a prefix-fák memóriatakarékos útvonal-megosztását a bináris keresőfák dinamikus, rendezett és alacsony elágazási költségű struktúrájával, hasonlóan ahhoz, ahogyan a ternáris keresőfák (TST) teszik azt a karaktersorozatokat esetében [5]. A jelen cikkben bemutatásra kerülő hierarchikus bináris keresőfa (röviden HBKF) ezt az őr kívánja betölteni.

2. A HIERARCHIKUS BINÁRIS KERESŐFA DEFINÍCIÓJA

A hierarchikus bináris keresőfa egy $X=(x_1, \dots, x_n)$ formában megadható rendezett n -esek (n -tuple) struktúrált tárolására kialakított fa adatszerkezet. Ennek minden E eleme 4 komponensből áll, azaz $E=\{A, K, N, S\}$ alakban írható fel. Minden csomópont tartalmaz egy hasznos terhet, mint tárolandó adatot, amit az A jelöl. Ez tulajdonképpen egy eltárolt rendezett n -es egyik x_i elemét jelenti. A további 3 komponens az adott csomópontból kiinduló részfákra vonatkozó hivatkozások. Hasonlóan a tradicionális bináris keresőfákhoz, a HBKF is egy rendezett fa. Ez abban nyilvánul meg, hogy adott i esetén a fa minden csomópontjának A komponensében tárolt érték rendre kisebb, illetve nagyobb, mint a K , illetve N komponensek által hivatkozott részfákban tárolt valamennyi x_i érték. Egy csomópont S komponense hivatkozik arra a részfára, amely azokat az értékeket tárolja el, amelyek adott i esetén konkrét $(x_1, \dots, x_i)$ előtaggal (prefix) rendelkeznek a rendezett n -esek elején.

Fentiek alapján elmondható tehát, hogy a HBKF egy ternáris fa, melynek minden csomópontja maximum 3 rákövetkező (gyermek) elemmel bírhat és a gyökér elemet leszámítva pontosan egy megelőző (szülő) elemmel rendelkezik. A tárolt rendezett n -esek i . pozícióban elhelyezkedő x_i tagjai a fa különböző rétegeiben helyezkednek el az i értékétől függően. A HBKF felfogható egy n rétegben egymásra épülő keresőfákat tartalmazó keresőfa koncepciónak. Bár ez a szerkezet alapvetően egy ternáris fa, az adatokat bináris logika mentén rendezi. Az $n=1$ esetben természetesen visszakapjuk a hagyományos bináris keresőfát.

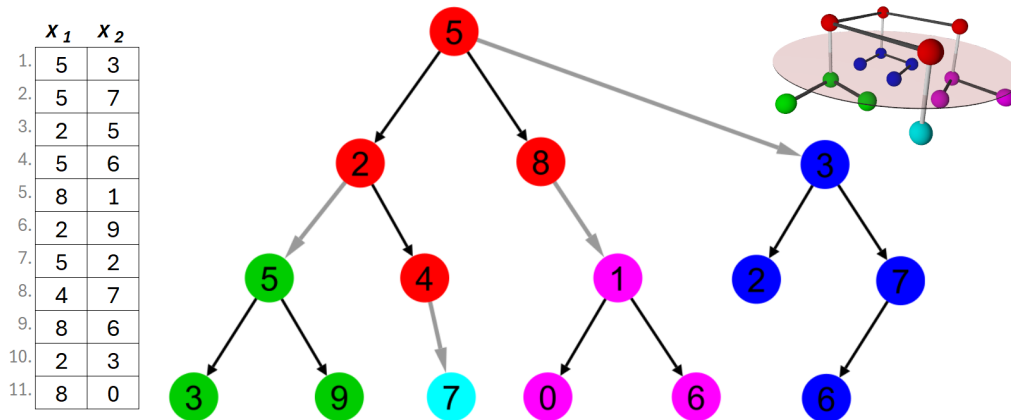
Az alábbi pszeudokóddal megadott rekurzív eljárás (lásd 1. ábra) pontosan leírja, hogyan történik egy X rendezett n -es beszúrása egy olyan (rész)fába, amelynek gyökerére a H segítségével hivatkozhatunk (az i pedig azt jelöli, hogy melyik rétegbe kerüljön beszúrásra az X adott komponense):

1: procedure BESZUR(X,H,i)	▷ Rekurzív eljárás
2: if $H = NULL$ then	▷ Üres (rész)fa
3: $H \leftarrow \text{new Csomopont}()$	▷ Új csomópont létrehozás
4: $H.A \leftarrow x_i$	▷ Adat tárolása
5: $H.K \leftarrow NULL$	▷ Nincs még kisebb
6: $H.N \leftarrow NULL$	▷ Nincs még nagyobb
7: $H.S \leftarrow NULL$	▷ Nincs még következő réteg
8: if $i < n$ then	▷ További réteg(ek)
9: BESZUR($X,H.S,i+1$)	▷ Rekurzív beszúrás a következő rétegbe
10: else	▷ Nem üres (rész)fa
11: if $H.A = x_i$ then	▷ Megtalált érték az i . rétegben
12: if $i < n$ then	▷ Further layer(s)
13: BESZUR($X,H.S,i+1$)	▷ Tovább lépés a következő rétegbe
14: else	▷ Nem talált érték az i . rétegben
15: if $H.A < x_i$ then	▷ Kisebb érték az i . rétegben
16: BESZUR($X,H.N,i$)	▷ Beszúrás a réteg nagy értékeihez
17: else	▷ Nagyobb érték az i . rétegben
18: BESZUR($X,H.K,i$)	▷ Beszúrás a réteg kis értékeihez

1. ábra. A hierarchikus bináris keresőfa adatszerkezetbe történő rekurzív beszúrás pszeudokódja

Mint látható, a rekurzió többrétű. Egyrészt arra szolgál, hogy megtalálja az adott x_i érték helyét a megfelelő rendezett részfában, másrészt arra, hogy minden i esetén a kellő rétegbe történjen beszúrás. Egy üres adatszerkezetből kiindulva rendezett n -esek sorozata beszúrható egy ciklikus eljáráshívással, amennyiben az i paraméter értéke kezdetben mindig 1.

A 2. ábra segítségével végigkövethető, hogy az egyes értékpárok adott sorrendben történő beszúrásával hogyan alakul ki a végleges szerkezet. A piros csomópontok jelentik az első rétegben felépített keresőfát, amely az összes előforduló x_1 értéket rendezetten tartalmazza. Mindegyik piros csomópontból elérhető egy-egy második rétegbe tartozó keresőfa, amely az adott x_1 értéket követő összes előforduló x_2 értéket tartja nyilván. A levélréteg kivételével, a hierarchia $i+1$. rétegben mindig annyi keresőfa található, amennyi csomópont van az i . rétegben.



2. ábra. A HBKF illusztrációja. A bal oldali rendezett 2-esek adott sorrendű beszúrásával ez a gráfszerkezet alakul ki. A csomópontokban lévő számok jelentik az A komponenszt, a rendre balra, illetve jobbra mutató fekete nyilak reprezentálják a K , illetve N komponenszt, a szürke nyilak pedig az S komponenszt, amelyek az új rétegbe való átlépést valósítják meg. A csomópontok színei az egyes keresőfákat szemléltetik. A jobb felső sarokban lévő 3D-kép szemlélteti a HBKF két rétegét.

Amennyiben a beszórando adathalmaz egy adott rendezett n -est többször is tartalmaz, akkor a HBKF-ban csak egyszer lesz eltárolva. Ha ez nem elegendő és ismerni szeretnénk minden n -es előfordulásának a számát, akkor a hierarchikus bináris keresőfa egy kibővített változatát kell alkalmaznunk. Ekkor a csomópontok egy új 5. komponenst is tartalmaznak, amely egy számláló szerepét tölti be. Valahányszor teljesül az 1. ábrán bemutatott algoritmus 11. sorában szereplő feltétel, akkor a számláló értékét növelni kell eggyel. (Természetesen egy új csomópont létrehozásánál a számlálót nullára kell állítani.) A levélelemekben lévő számláló így az adott rendezett n -esek előfordulási számát tartalmazza.

A fentebb bevezetett számláló nemcsak a levélelemekben hasznos. Egy i . rétegben található csomópont számlálóját megadja, hogy hány olyan rendezett n -es van az adathalmazban, amelynek az elején az adott $(x_1, \dots, x_i)$ adatsor található. Tehát gyakorlatilag minden megtalálható prefix előfordulásának száma ismert.

Ha egy adott probléma esetén nemcsak az fontos, hogy egy konkrét rendezett n -es hányszor fordul elő, hanem ezek az adatsomagok egyedileg azonosíthatók vagy az n darab kereshető érték mellett további leíró mezőket is tartalmaznak, akkor szükség lehet ezek beazonosítható tárolására. Ilyenkor a hierarchikus bináris keresőfához egy újabb réteget adhatunk (kibővített HBKF), amely a korábbi levél elemekhez kapcsolódó extra keresőfákat jelent. Ezekben tárolhatjuk rendezetten az egyes rendezett n -esek azonosítóit (és leíróit). Ezáltal a többszörösen előforduló érték n -esek is hatékonyan kereshetők lesznek, mialatt az adatszerkezet konzisztens marad.

3. TELJESÍTMÉNYELEMZÉS

Az adatszerkezetbe történő beszúrás algoritmusának tár- és időkomplexitás elemzése szempontjából két különböző eset kerül bemutatásra ebben a tanulmányban. Jelölje N a beszúrando rendezett n -esek darabszámát és K (az adott i esetén) az egyes x_i elemek által felvehető különböző értékek maximális számát, vagyis az egyes oszlopokra vonatkozó értéktartomány méretét! Tételezzük fel, hogy sok rendezett n -essel kell dolgoznunk, azaz $N \gg 1$, és az értékek véletlenszerűek, azaz függetlenek az adott rendezett n -esben szereplő más x_i értékektől és a további rendezett n -esektől is!

3.1. Széles intervallumból származó adatok

Tételezzük fel, hogy a HBKF adatszerkezetben tárolandó adatok olyanok, hogy a rendezett n -esek minden x_i eleme egy széles értéktartományból származik, azaz minden i esetén az x_i értékei nagyon sokfélék lehetnek ($K \gg N$).

Mivel a felvetésben teljesen véletlenszerű adatok szerepelnek, ezek biztosítják azt, hogy az adott szinten lévő bináris keresőfák nagyjából kiegyensúlyozottak maradjanak (tehát várhatóan nem fajulnak el aszimmetrikus láncolt listává). Amikor a rekurzív Beszur függvény elindul az első dimenzió (legfelső réteg) gyökerétől, egy maximum N elemű bináris fában kell megkeresnie az x_i helyét. Mivel a fa a véletlen adatok miatt nagyjából kiegyensúlyozott, a keresés és beszúrás ideje ezen a szinten $O(\log N)$. Ezután az algoritmus átlép a következő rétegbe. Ritka ütközések miatt, ha a legelső dimenzióban az érték teljesen egyedi, akkor a következő rétegben még nincs felépítve fa. Ekkor az algoritmus csak létrehozza a hátralévő dimenziókhoz a csomópontokat $O(n)$ lépésben. Az időkomplexitás ekkor tehát $O(\log N + n)$.

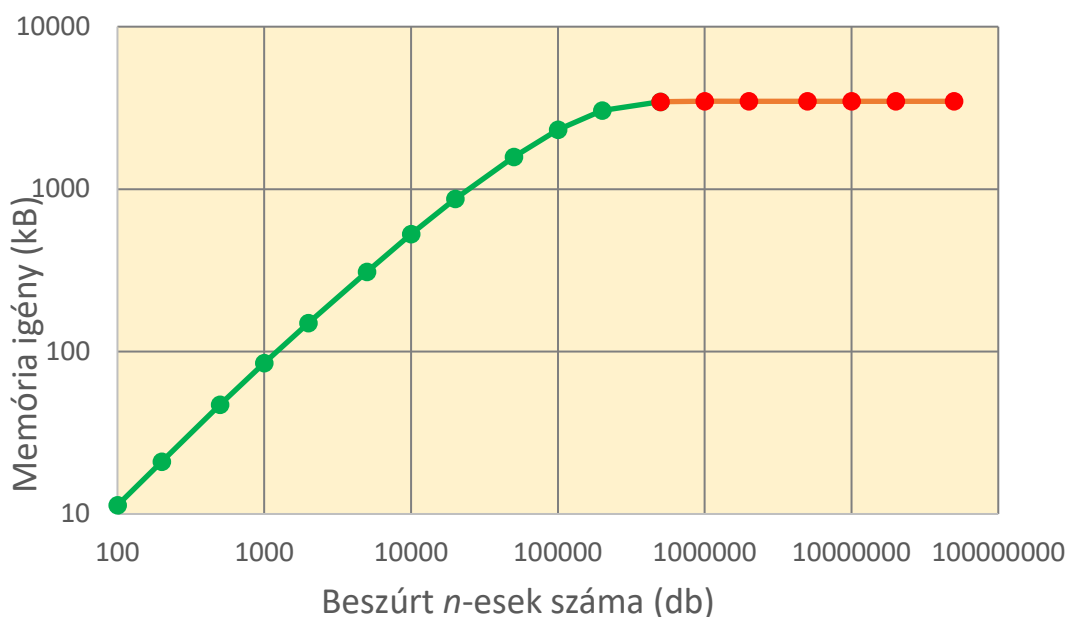
Egyetlen n -es beszúrásának tárkomplexitása $O(n)$, a teljes adathalmaz (N darab rendezett n -es) tárkomplexitása pedig $O(N \times n)$. Ez azzal magyarázható, hogy véletlenszerű adatok esetén rendkívül kicsi az esélye a hosszú prefix egyezéseknek. A rekordok szinte azonnal elválnak egymástól a fában, ami azt jelenti, hogy az algoritmus szinte minden egyes új rekordhoz n darab új csomópontot foglal le a memóriában.

Látható tehát, hogy a véletlenszerűség az időkomplexitás szempontjából kedvező (megelőzi a bináris fák degenerálódását és gyorsítja a keresést), a tárkomplexitás szempontjából ez a legrosszabb forgatókönyv, mert alig lesz memóriamegtakarítás az adatok ismétlődéséből, így a fa mérete brutálisan nagyra nőhet.

3.2. Keskeny intervallumból származó adatok

Ha a véletlenszerű x_i oszlopok értékkészlete véges és viszonylag szűk ($K < N$), akkor ez a hierarchikus adatszerkezet egy ponton túl telítődik. Egyetlen rendezett n -es beszúrásának várható időkomplexitása drasztikusan lecsökken: $O(n \times \log K)$. Mivel egy adott oszlop értéktartománya maximum K elemből áll, egyetlen rétegen belüli bináris fa mérete sem nőhet K fölé (hiszen nincsenek új, ismeretlen értékek), vagyis független lesz N értékétől. Mivel az elemek eloszlása továbbra is véletlenszerű, ezek a maximum K méretű részfák nagyjából kiegyensúlyozottak maradnak. Egy rétegen belül a keresés illetve beszúrás így legfeljebb $O(\log K)$ lépést vesz igénybe. Ezt a keresést a programnak az n dimenzió keresztlül kell elvégeznie. Bármilyen hatalmas is az N , egy új elem beszúrása konstans közeli időt vesz igénybe, mert a fa mélysége és az egyes szintek kiterjedése szigorúan korlátozott.

A véges értékkészlet miatt az n -esek elkezdnek megegyezni az első néhány oszlopban (közös prefixek alakulnak ki). A program memóriefoglalása optimalizálódik: a rétegek közötti kapcsolaton keresztül az adatsorok osztoznak a korábban létrehozott csomópontokon. A fa 1. rétegében maximum K csomópont lehet. A 2. rétegben összesen K^2 , míg az n -edik rétegben K^n csomópont lehetséges. A fa maximális elméleti mérete nagyságrendileg a lehetséges kombinációk számától (K^n) függ. Amint a feldolgozott sorok száma (N) elkezd jócskán meghaladni a lehetséges variációk számát, a program egyre ritkábban foglal memóriát új csomópontok számára, csak a számlálót növeli. Így a teljes adathalmaz tárkomplexitása a következő felső korlátot kapja: $O(\min(N \times n, K^n))$. Ahogyan azt a 3. ábra is alátámasztja, az N növelésével a memóriaigény konstanssá válik, ha $N \times n$ értéke megközelíti K^n értékét, kisebb N értékek esetén viszont a memóriaigény monoton nő.



3. ábra. A HBKF tárkomplexitása $n=5$ és $K=10$ esetén.

3.3. A mezők sorrendje

A rendezett n -esekben szereplő mezők (oszlopok) sorrendje kötött, de ennek a sorrendnek általában nincs jelentősége. Például egy személyt leíró 2-esben x_1 mező lehet az életkor, és az x_2 pedig az illető neme, de akár fordítva is megadható (először a nem, aztán a kor). A hierarchikus bináris keresőfában tárolt n -esek esetében viszont az oszlopok sorrendje befolyásolja a teljesítményt. Ha az i . mező K_i darab egyedi értéket tartalmazhat, akkor érdemes a mezők sorrendjét a K_i értékek növekvő sorrendje szerint megadni. Amennyiben $K_i < K_{i+1}$ viszony minden $i < n$ esetre fennáll, akkor a legelső rétegben kevés csomópont lesz, azaz belőlük kiindulva kevesebb második rétegbeli keresőfát kell kezelni, és így tovább. A szűkebb tartományú mezők előbbre kerülése révén gyakoribbak lesznek az azonos prefixek, amelyeket csak egyszer kell eltárolni. Ezt két véletlenszerű értékeket tartalmazó tesztadathalmaz segítségével szemléltethetjük. Az egyes adathalmazok esetén kétféleképpen kerülnek felépítésre a HBKF-k úgy, hogy a rendezett n -esekben csak a mezők sorrendje változik, de az értékek nem. Az alábbi táblázat egy 10 000 darab rendezett 5-ösből és egy szintén 10 000 rendezett 2-esből felépített adatszerkezet csomópontszámát foglalja össze. Látható, hogy a szükséges csomópontszám függ az egyes oszlopok K_i értékétől. (A * karakterrel jelölt csomópontszámok az egyes adathalmazok esetén minimális értékek.)

Példák a mezősorrend és a csomópontszám kapcsolatára

1. táblázat

n	K_1	K_2	K_3	K_4	K_5	csomópontszám
5	2	3	4	5	6	* 872
5	5	2	3	4	5	885
5	6	2	3	4	5	918
5	6	5	4	3	2	1236
2	10	10 000	-	-	-	* 9962
2	10 000	10	-	-	-	19468

4. KONKLÚZIÓ

A bemutatott hierarchikus bináris keresőfa [6], amely felfogható egy hagyományos keresőfákból álló keresőfának, kiválóan alkalmas lehet rendezett n -esek hatékony tárolására. Az azonos prefixeket csak egyszer kell tárolni, ezáltal a memóriakihasználtság nagyon jó lehet olyan esetekben, amikor a tárolandó adatok sok ismétlődő értéket tartalmaznak. Mivel az adatok keresőfákban vannak, a beszúrás és a keresés időigénye még óriási adathalmazok esetén is nagyon alacsony. A beszúrás és a keresés művelete is könnyen implementálható egy kétszintű rekurzív algoritmussal.

Számos területen érdemes lehet felhasználni egy ilyen adatszerkezetet. Álljon itt csak néhány példa.

- Ha a szokásos inorder bináris fabejárást kiterjesztjük a rétegeken átívelővé, akkor ennek segítségével könnyen tudunk nagy mennyiségű adatot (növekvő vagy csökkenő sorrendbe) rendezni.
- Ha a beszúrás előtt az adatok átesnek egy csoportosításon vagy kvantáláson, akkor egyfajta klaszterezést tudunk megvalósítani a kibővített HBKF segítségével. Ekkor a klaszterbe tartozó rekordok azonosítói az új extra rétegben kerülnek eltárolásra, tehát könnyen meg tudjuk mondani, mely rekordok tartoznak az adott értéktartományok által megadott klaszterhez.
- A számláló komponens kiválóan alkalmazható az előfordulási statisztikák azonnali előállításához. Például egy *ország-megye-város-utca* logikájú rendezett 4-es esetén a különböző rétegekben található számlálók azonnal megadják, hány rekord tartozik egy adott országhoz vagy akár egy konkrét *ország-megye-város* hármashoz.
- Az adatvédelem (data privacy) területén [7] fontos mérőszám a k -anonimitás [8], amely előírja, hogy egy közzétett adatkészletben található egyetlen rekordot se lehessen megkülönböztetni legalább $k-1$ másik rekordtól kváziazonosító adatok alapján. A HBKF levélcsoomópontjaiban tárolt számlálókon történő egyszerű minimumkereséssel meghatározható az adathalmaz k -anonimitás értéke.

KÖSZÖNETNYILVÁNÍTÁS

Ezt a munkát a Debreceni Egyetem TKP2021-NKTA projektje támogatta. A TKP2021-NKTA-34 számú projekt a Nemzeti Kutatási, Fejlesztési és Innovációs Alap (NKF) támogatásával valósult meg, a TKP2021-NKTA pályázati program keretében finanszírozva.

IRODALMI HIVATKOZÁSOK

- [1] Sedgewick R., Wayne K. *Algorithms*, (4. kiadás) Addison-Wesley Professional, 2011.
- [2] Charras C., Lecroq T. *Handbook of Exact String Matching Algorithms*, College Publications, 2004.
- [3] Fredkin E. *Trie memory*, Communications of the ACM, 1960, 3(9), 490-499.
- [4] Bentley J. L. *Multidimensional binary search trees used for associative searching*, Communications of the ACM, 1960, 18(9), 509-517.
- [5] Bentley J., Sedgewick R. *Fast algorithms for sorting and searching strings*, Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms, ACM, 1997, 360-369.
- [6] Varga I.: *Hierarchical Binary Search Tree*. GitHub repository, <https://github.com/VargaImre79/HBST> (Utolsó letöltés: 2026.08.26).
- [7] Dwork C., Roth A., *The Algorithmic Foundations of Differential Privacy*. Foundations and Trends in Theoretical Computer Science, 2014, 9 (3-4), 211-487.
- [8] Khanaa V., Thooyamani K.P., *Protecting Privacy When Disclosing Information: K Anonymity and its Enforcement through Suppression*. International Journal of Computing Algorithm, 2012, 1 (1), 19-22.