Mintaillesztések összehasonlítása dokumentáció

*Készítette: Gergácz Dániel*

*Neptun: FAZXPD*

*E-mail:* [*gergaczdaniel@gmail.com*](mailto:gergaczdaniel@gmail.com)

*Dátum: 2014. január 06.*

# Feladat

Mintaillesztések, elsősorban a Knuth-Morris-Pratt és a Boyer-Moore algoritmusok műveletigényének összehasonlítása, de más is szóba jöhet, pl. az utóbbi helyett a Boyer-Moore-Horspool féle egyszerűbb változat, amelyet „quick search” néven tanultunk a BSc-n. Az input lehet természetes, illetve speciális szakirányú (esetleg random) szöveg. Lehet-e valamit mondani arról, hogy milyen természetű szövegek a kedvezők az egyes mintaillesztések számára?

# Megvalósítás

A feladat implementálása során a Knuth-Morris-Pratt, a Boyer-Moore, és a Booyer-Moore-Horspool algoritmusokat hasonlítottuk össze. A forrás szöveg megadására két lehetőség van: kézzel való begépelés (vagy beillesztés), valamint szöveges fájl megadása (akár több fájlt is megadhatunk egyszerre). A keresendő mintát viszont hagyományos módon kell megadni, be kell gépelni.  
Az implementált algoritmusaink olyan szempontból speciálisakra terveztük, hogy az összes előforduló találatot megkeresi, viszont a program felkínálja a lehetőséget rá, ha csak az elsőt szeretnénk megtalálni.

## Knuth-Morris-Pratt algoritmus

Az algoritmus lényege, hogyha elromlik az illesztésünk, akkor a legkisebb olyan ugrást kell választanunk, ahol az eredeti minta egy prefixe illeszkedik az rontásig lévő minta suffixére. Erre az algoritmusban bevezettünk egy nextArray nevű tömböt, ami ezt eltárolja még az elő feldolgozási lépésben, így az algoritmus futása során ennek az elérése konstans idejű lesz. Ennek a kiszámítása lineáris idejű.

## Boyer-Moore algoritmus

A KMP algoritmustól annyiban tér el, hogy a rontás esetén más tényezőket vesz figyelembe az ugrás kiszámításánál. Erre két szabályt alkalmaz, egyik a „rossz karakter” szabály, másik a „jó suffix” szabály. Az ugrási szabályok természete miatt a Boyer-Moore algoritmus hátulról kezdi el a mintaillesztést, azaz egy n hosszúsági minta esetén a szöveg n-ik karakterét hasonlítja először a minta n-ik karakterével, és így halad visszafele amíg egyezik.

A „rossz karakter” szabály  
Minden egyes karakterhez eltároljuk, hogy a minta egy adott pozíciójától kezdve, hol található a következő ilyen karakter, amennyiben nincs ilyen akkor -1 lesz. Ez pontosan azért jó, mert már tudni, fogjuk, hogy az egész mintából 1 karakter már jó helyen van. A -1-el azt jelöljük, hogy az egész mintát el kell tolni az aktuálisan vizsgált elrontott karakteren túlra.

Az implementáció során ezt, úgy oldottuk meg, hogy csak a mintában szereplő karaktereket néztük, és ezekre számoltuk ki ezeket az indexeket, ugyanis ha olyan karaktert kérdezünk le ami nincs, akkor az hamissal fog visszatérni JavaScript-ben.

### A „jó suffix” szabály

Ez az ugrási szabály egy kicsit bonyolultabb, és lényegében emiatt kell hátulról ellenőrizni a mintát is.

## Boyer-Moore-Horspool algoritmus

Annyiban tér el az előzőtől, hogy az ugrási szabályok közül csak a „rossz karakter” szabályt alkalmazza, ezzel javítva az algoritmuson.

# Hatékonyság vizsgálata

Az tesztelés során a Project Gutenberg (http://www.gutenberg.org) forrásból vettünk mintaként pár szépirodalmi művet. Mivel még aránylag nagy fájlok esetén is rendkívül gyorsnak mondhatóak az algoritmusok (2,5 millió karakter hosszú forrás esetén lépi át az 0,1 másodperces határt), ezért szükség volt egy teljesítmény kiértékelő könyvtárra. JavaScript esetén erre az egyik legalkalmasabb a Benchmark.js (<http://benchmarkjs.com>), ami többszöri futtatás alapján kiátlagolja, a futási idejét a benne elhelyezed kódrésznek, éppen ezért magának a környezetben történő futtatása egy kicsit hosszabb ideig tart a vártnál. Az eredmények vizualizációjánál szintén egy JavaScript-es könyvtárat vettünk segítségül, a YUI könyvtárat (<http://yuilibrary.com>). Ez sok mindenre használható, de esetünkben a diagrammok elkészítésére szolgál.  
Futtatáskor két diagram rajzolódik ki, az egyik a tiszta futási időre vonatkozik, magát az algoritmusok elő feldolgozását nem számoljuk bele, ez lényegében elhanyagolható, ugyanis csak a keresendő kifejezéstől függ, és amennyiben csak a szöveg változik, ez nem generálódik újra. A másik diagramon a futás során történő karakter összehasonlításokat figyelhetjük meg, ami a ciklus menetét adja meg. Ez azért is lényeges, mert független az implementációtól.