DÚ 3

Všeobecné poznámky

- Ku úlohe je template. Píšte riešenie doňho. Meniť súbor Main.java je zakázané (až na výnimky, špecifikované nižšie v texte). Meniť štruktúru súborov (t.j. premenovávať, pridávať alebo odstraňovať súbory) je zakázané.
- Vaše riešenie by sa malo dať skompilovať príkazom `javac Main.java */*java` bez dodatočných flagov, tak je riešenie hodnotené počtom bodov nula (príkaz `java --version` vracia "openjdk 17.0.5 2022-10-18", alebo kompatibilný variant).
- Ak chcete, aby sa hodnotila príslušná podúloha, musíte odkomentovať príslušný riadok v metóde main triedy Main (v opačnom prípade je hodnotená počtom bodov 0.5) (obdoba pravidla "nechcem riešiť danú podúlohu" z písomiek z predmetu Formálne jazyky a automaty). Pre bonus tento bod **neplatí**.
- Deadline na odovzdanie je 3.12.2023, 23:59:00.
- Odovzdáva sa zdrojový kód v ZIP archíve do Teamsov.

Podúloha 1 - Konečný automat (5 bodov)

Pomocou návrhového vzoru State implementujte deterministický konečný automat nad abecedou {a, b} so štyrmi stavmi A, B, C, D, kde A je počiatočný stav, C je akceptačný stav, a prechodová funkcia f vyzerá nasledovne: f(A, 'a') = B, f(A, 'b') = A, f(B, 'a') = A, f(B, 'b') = C, f(C, 'a') = C, f(C, 'b') = D, f(D, 'a') = C, f(D, 'b') = D (t.j jazyk tohto automatu sa dá písať ako regulárny výraz nasledovným spôsobom: "b*a(ab*a)*b(a*bb*a)*a*").

Trieda FiniteAutomaton má podporovať dve verejné metódy:

- boolean isAcceptingState() - vráti true alebo false poďla toho, či sa práve automat nachádza v akceptačnom stave
- void transition(char c) - vykoná krok výpočtu daného automatu

Použite verziu návrhového vzoru State, pri ktorej za rozhodnutie o zmene stavu zodpovedá objekt stavu, nie objekt kontextu.

Podrobnosti o deterministických konečných automatoch a regulárnych výrazoch môžete nájsť v skriptách predmetu Formálne jazyky a automaty (1).

Podúloha 2 - Paralelné lineárne vyhľadávanie (5 bodov)

Implementujte generickú triedu MultiSearchArrayList, ktorá je potomkom generickej triedy ArrayList s preťaženou metódou contains, ktorá bude robiť to isté, čo pôvodná metóda

contains (t.j. lineárne prehľadávanie poľa), ale paralelné vo 4 vláknach (t.j. máte to pole rozdeliť na štyri rovnako veľké časti, jednú pre každé vlákno).

Implementácia má byť lock-free (t.j. bez použitia zámkov alebo kľúčového slova synchronized) a používať nanajvýš O(1) bajtov pomocnej pamäti. Triedu, reprezentujúcu Thread, implementujte ako vnútornú triedu (inner class) triedy MultiSearchArrayList.

Cielom je implementovať metódu, ktorá bude rýchlejšia než jednovláknová knižničná funkcia.

Zíde sa poznať metódu Thread.join() (dokumentáciu nájdete na Internete).

Bonus - Paralelný MergeSort (4 body)

Implementujte triedu MultiSorter s nasledovnými verejnými metódami:

- MultiSorter(int max_depth) - konštruktor, nastavujúci konštantu MAX_DEPTH
- void sort(ArrayList<Integer> array) - utriedi dané pole

Interne daný objekt implementuje paralelnú obdobu MergeSort: rozdelí pole na dve časti, paralelne rekurzívne utriedi obidve časti a následne spojí. Ak sa rekurziou dostanete do hĺbky viac ako MAX_DEPTH, tak pole utrieďte sekvenčnou metódou (napr. volaním knižničnej metódy Collections.sort).

Implementácia má byť lock-free (t.j. bez použitia zámkov a kľúčového slova synchronized). Triedu, reprezentujúcu Thread, implementujte ako vnútornú triedu (inner class) triedy MultiSorter.

Cielom je implementovať metódu, ktorá bude rýchlejšia než jednovláknová knižničná funkcia.

Oplatí sa poznať metódu Thread.join() (dokumentáciu nájdete na Internete).

**Komentár ku použitiu príkazu join(): príkaz join funguje tak, že po jeho zavolaní hlavné vlákno bude čakať na ukončenie príslušného vlákna. Čiže pre súčasný beh viacerých vlákien treba *najprv ich všetky spustiť,* a až potom zavolať na nich join.