Domáce zadanie 4 – analýza výkonu ADT tabuľka

Cieľom domáceho zadania je porovnanie dvoch implementácií ADT tabuľka. V rámci testovania je nutné porovnať výkonnosť štruktúr v konkrétnych scenároch a odhadnúť časovú zložitosť konkrétnych operácií v závislosti na počte prvkov v tabuľke.

Úloha 1 – implementácia

Implementujte ADT tabuľka, ktorý musí podporovať operácie definované v súbore **table.h**. Implementáciu realizujte ako nasledujúce AUŠ:

- ***usporiadaná sekvenčná tabuľka*** (**sorted_sequence_table.h**);
- ***binárny vyhľadávací strom*** (**binary_search_tree.h**).

V triedach definovaných v súboroch **sorted_sequence_table.h** a **binary_search_tree.h** je nutné implementovať všetky verejné metódy (a všetky verejné metódy predkov) a podľa potreby si môžete pridať ľubovoľné atribúty a ľubovoľné súkromné metódy (prípadne upraviť viditeľnosť atribútov u predkov z private na protected, ak využívate dedičnosť). **Môžete využiť vlastnú implementáciu z cvičení.**

Správnu implementáciu jednotlivých operácií je nutné otestovať prostredníctvom Vami implementovaných jednotkových testov (unit testov). Tieto implementujte v súboroch **table_test.h** a **table_test.cpp**.

Súbory table.h, sorted_sequence_table.h, binary_search_tree.h, table_test.h a table_test.cpp nájdete v projekte využívanom na cvičeniach.

Úloha 2 – overenie výkonu v scenári

Realizácie ADT tabuľka, ktoré ste implementovali v úlohe 1, otestujte v scenároch definovaných v Tab. 1. V každom scenári vykonajte spolu 100 000 operácií. Jednotlivé operácie sú v jednotlivých scenároch volané náhodne tak, aby na konci súhlasil podiel jednotlivých operácií (neplatí, že najskôr sa zavolajú operácie na vkladanie prvkov, potom operácie na mazanie prvkov a nakoniec operácie na vyhľadávanie prvkov). Parametre do operácií sú taktiež náhodné; kľúč je náhodné číslo z intervalu <0; 100 000>.

Tab. 1 Testovacie scenáre pre ADT tabuľka

Scenár	Podiel operácií		
	`insert`	`remove`	`tryFind`
A	20	20	60
B	40	40	20

V rámci analýzy výkonu v scenári je nutné merať len dĺžku trvania vybranej operácie. To znamená, že do merania **sa nesmie započítavať čas potrebný pre generovanie pomocných údajov**.

Úloha 3 – analýza časových zložitostí

V rámci analýzy časových zložitostí je nutné otestovať rýchlosť operácií `insert`, `remove` a `tryFind` v závislosti od počtu prvkov a implementácie tabuľky a na základe nameraných a spracovaných údajov **odhadnúť hornú asymptotickú zložitosť jednotlivých operácií**.

V rámci analýzy časových zložitostí vybraných operácií je nutné merať len dĺžku trvania vybranej operácie. To znamená, že do merania **sa nesmie započítavať čas potrebný pre generovanie pomocných údajov**.

Úloha 4 – bonus

Cieľom bonusovej úlohy je otestovať vplyv vyváženia binárneho vyhľadávacieho stromu na vyhľadávanie. Implementujte štruktúru ***treap*** (súbor nájdete v súbore treap.h) a porovnajte:

- výkon tejto štruktúry v scenári A a B z úlohy 2 voči už analyzovaným implementáciám tabuľky.
- porovnajte časové zložitosti operácií `insert`, `remove` a `tryFind` v závislosti od počtu prvkov medzi štruktúrami treap a binárny vyhľadávací strom.

Výstupy domáceho zadania

Domáce zadanie bude mať nasledujúce výstupy:

1. súbory (*odovzdávate všetky súbory bez ohľadu na to, ktoré časti domáceho zadania riešite*):
   - **sorted_sequence_table.h** – implementácia ADT tabuľka – utriedená sekvenčná tabuľka (úloha 1);
   - **binary_search_tree.h** – implementácia ADT tabuľka – binárny vyhľadávací strom (úloha 1);
   - **treap.h** - implementácia ADT tabuľka – treap (úloha 4);
   - **table_test.h**, **table_test.cpp** – implementácia jednotkových testov ADT tabuľka (úloha 1) vrátane výkonnostných testov definovaných v úlohách 2, 3 a 4;
   - **Po dosadení Vašich štruktúr sa projekt používaný na cvičeniach musí dať preložiť.** Ak využívate implementáciu tabuliek z cvičení, tak odovzdajte aj ostatné potrebné súbory:
     1. **pre sekvenčnú tabuľku:** sequence_table.h, array_list.h, array.h, vector.h, vector.cpp.
     2. **pre binárny vyhľadávací strom:** binary_tree.h, k_way_tree.h, array.h, vector.h, vector.cpp.

     Ak implementujete štruktúry vo vlastnej réžii, odovzdajte všetky potrebné súbory, ktoré bude projekt s Vašimi štruktúrami vyžadovať.
2. CSV súbory s údajmi zaznamenanými počas testov (*odovzdávate len v prípade riešenia úloh 2, 3 a 4*);

3. dokumentácia (*odovzdávate len v prípade riešenia úloh 2, 3 a 4*), ktorá obsahuje:
    - popis realizácie priebehu jednotlivých testov – v prípade úlohy 2 je nutné vysvetliť, ako budete realizovať jednotlivé scenáre, t. j. ako sa budete v danom scenári rozhodovať, ktorú operáciu vykonáte;
    - popis formátu údajov a CSV súborov v testoch – je nutné popísať údaje, zaznamenávané počas testov, a formát CSV súboru (teda, čo sa bude v CSV súbore nachádzať a kde), do ktorého budete tieto údaje ukladať;
    - metodiku spracovania a vyhodnotenia výsledkov testov (CSV súborov) – je nutné popísať, ako budete analyzovať výsledné CSV súbory;
    - prezentáciu výsledkov a záverov vyplývajúcich z testovania.

V kóde používajte namiesto číselných konštánt symbolické. Údajové štruktúry a testy musia byť naprogramované tak, aby plne dodržiavali rozhrania špecifikované v uvedených súboroch, čím ich bude možné bez problémov zakomponovať do projektu AUS používanom na cvičeniach. Údajové štruktúry a testy musia byť Vami naprogramované a **efektívne** implementované z pohľadu výpočtovej a pamäťovej zložitosti. Po ukončení behu projektu AUS s Vašimi súbormi musí byť pamäť **preukázateľne čistá** (nevznikli „memory leak-y“).

Testy vyprodukujú CSV súbor so zaznamenanými údajmi. Vyprodukované CSV súbory je potrebné spracovať (akokoľvek – ručne, pomocou tabuľkového editora, pomocou ďalšej alebo tej istej aplikácie). **Metodické popísanie spôsobu spracovania údajov je súčasťou dokumentácie.** Pomocou popísaného postupu musí byť možné nové údaje spracovať rovnako ako pôvodné. Výsledky spracovania (grafy, priemery, odhady, atď.) je potrebné zdokumentovať a interpretovať (vyvodiť závery). Identifikácia relevantných veličín je súčasťou domáceho zadania.