Za domácí úkol máte příklady 2 a 3. Důkladně si přečtěte zadání! Můžete používat vyvážené vyhledávací stromy a využívat skutečnosti, že umí přidávat i mazat prvky a přitom si zachovávat logaritmickou výšku.

Definice 1 (Binární strom)**.** Strom nazveme binární, pokud je zakořeněný a každý vrchol má nejvýše dva syny, u nichž rozlišujeme, který je levý a který pravý. Vrcholy rozdělíme podle vzdálenosti od kořene do hladin: v nulté hladině leží kořen, v první jeho synové atd.

Pro vrchol v binárního stromu T značíme:
- $k(v)$ je klíč uložený ve vrcholu v.
- $T(v)$ je podstrom obsahující vrchol v a všechny jeho potomky.
- $l(v)$ a $r(v)$ jsou levý a pravý synové vrcholu v.
- $L(v)$ a $R(v)$ jsou podstromy pravého a levého vrcholu v.
- $h(v)$ je hloubka stromu $T(v)$, čili maximum z délek cest z v do listů.

Definice 2 (Binární vyhledávací strom)**.** Binární vyhledávací strom je binární strom, jehož každému vrcholu v přiřadíme unikátní klíč $k(v)$ z univerza. Přitom musí pro každý vrchol v platit:
- Kdykoliv $a \in L(v)$, pak $k(a) < k(v)$.
- Kdykoliv $a \in R(v)$, pak $k(a) > k(v)$.

Definice 3. Binární vyhledávací strom nazveme hloubkově vyvážený, pokud pro každý jeho vrchol v platí $|h(l(v)) - h(r(v))| \leq 1$.

Jinými slovy, hloubka levého a pravého podstromu se vždy liší nejvýše o jedna. Stromům s hloubkovým vyvážením se říká AVL stromy, neboť je vymysleli v roce 1962 ruští matematici Georgij Maximovič Aděľson-Veľskij a Jevgenij Michailovič Landis.

Definice 4. Obecný vyhledávací strom je zakořeněný strom s určeným pořadím synů každého vrcholu. Vrcholy dělíme na vnitřní a vnější, přičemž platí:
- Vnitřní (interní) vrcholy obsahují libovolný nenulový počet klíčů. Pokud ve vrcholu leží klíče $x_1 < \ldots < x_k$, pak má $k + 1$ synů, které označíme $s_0, \ldots, s_k$. Klíče slouží jako oddělovače hodnot v podstromech, čili platí: $T(s_0) < x_1 < T_(s_1) < x_2 < \cdots < x_k < T(s_k)$, kde $T(s_i)$ značí množinu všech klíčů z daného podstromu.
- Vnější (externí) vrcholy neobsahují žádná data a nemají žádné potomky. Jsou to tedy listy stromu. Na obrázku je značíme jako malé čtverečky, v programu je můžeme reprezentovat nulovými ukazateli (NULL v jazyku C, nil v Pascalu).

Definice 5. (a, b)-strom pro parametry $a \geq 2$, $b \geq 2a - 1$ je obecný vyhledávací strom, pro který navíc platí:
- Kořen má 2 až b synů, ostatní vnitřní vrcholy a až b synů.
- Všechny vnější vrcholy jsou ve stejné hloubce.

Definice 6. LLRB (červeno-černé, left-leaning red-black) strom je binární vyhledávací strom s vnějšími vrcholy, jehož hrany jsou obarveny červeně a černě. Přitom platí následující axiomy:
1. Neexistují dvě červené hrany bezprostředně nad sebou.
2. Jestliže z vrcholu vede dolů jediná červená hrana, pak vede doleva.
3. Hrany do listů jsou vždy obarveny černě. (To se hodí, jelikož listy jsou pouze virtuální, takže do nich neumíme barvu hrany uložit.)
4. Na všech cestách z kořene do listu leží stejný počet černých hran.

Příklad 1. Mějme posloupnost matic $A_1, \ldots, A_n$, kterou si chceme uložit tak, abychom rychle uměli odpovídat následující dotazy: Pro dané indexy i a j urči součin matic $A_i \cdots A_j$. Součin matic pomalá operace, takže chceme ukládat pomocné informace takové, že k vyhodnocení dotazu potřebujeme co nejmenší počet součinů matic. Mohli bychom si pamatovat součin matic $A_i \cdots A_j$ pro všechny indexy $i \leq j$, ale takových dvojic je $\binom{n+1}{2}$ a tolik paměti nemáme. Kolik paměti potřebujete, abyste dokázali dotaz vyhodnotit jen na jeden součin dvou matic? Kolik součinů potřebujete, pokud smíte použít jen $\mathcal{O}(n)$ paměti? Můžete předpokládat, že jednu matici lze uložit v $\mathcal{O}(1)$ paměti. Nezapomeňte, že násobení matic je sice asociativní, ale není komutativní ani nelze matice dělit.

Příklad 2. Na vstupu postupně přicházejí čísla. Kdykoliv přijde další, vypište medián z posledních k čísel. Jelikož velikost vstupu je obrovská, zajímá nás paměťová a časová složitost v k.

Příklad 3. Inverze v permutaci p čísel $1, 2, \ldots, n$ je dvojice indexů (i, j) taková, že $1 \leq i < j \leq n$ a $p_i > p_j$. Najděte co nejrychlejší algoritmus, který pro danou permutaci určí počet inverzí.