Hashovací tabulka: Datová struktura implementující množinu/slovník s *průměrnou* časovou složitostí $\mathcal{O}(1)$ na `Insert` a `Find`.
Hashování s přihrádkami: Pro každý hash máme spojový seznam, v případě kolize přidáme hodnotu do seznamu.
Hashování s otevřenou adresací: Máme jedno velké pole indexované hashem, v případě kolize se posuneme na jiný index pole.

1. Mějme zadanou množinu celých čísel. Pro číslo x zjistěte, jestli existují v množině a, b t.ž. $a + b = x$. Porovnejte implementace s různými datovými strukturami.

2. Popište algoritmus pro `Delete` na hashovací tabulce s otevřenou adresací a lineárním přidáváním[1], kde naše operace `Delete` rovou dostane ukazatel na prvek v tabulce.
   Jak dlouho trvá jedna operace (v nejhorším případě i průměrně)?
   Porovnejte časovou složitost s mazáním v hashovací tabulce s přihrádkami.

3. Najděte datovou strukturu, která pomocí hashování bude reprezentovat „optimistickou množinu“ o maximálně n prvcích s následujícími operacemi (a oběma běžícími v čase $\mathcal{O}(1)$ v nejhorším případě):
   - Operace `Insert` vloží prvek do množiny.
   - Operace `Member` vrátí „ano“, pokud prvek v množině leží. Pro prvek mimo množinu může také vrátit „ano“, ale jen s pravděpodobností $\leq \frac{1}{k}$ přes všechny hodnoty.

4. Řekněme, že chceme mít na disku uloženou velkou hashovací tabulku. Kterou z variant bychom měli zvolit, abychom při hledání prvku potřebovali načíst co nejmenší počet bloků disku?

5. Mějme posloupnost matic různých rozměrů, jejichž součin chceme zjistit. Pomocí dynamického programování najděte algoritmus, který v čase $\mathcal{O}(n^3)$ najde takové uzávorkování posloupnosti, aby výpočet součinu v této podobě minimalizoval počet násobených čísel.

6. Posloupnost je *bitonická*, pokud se skládá z rostoucí a klesající části. Popište algoritmus, který v posloupnosti čísel najde nejdelší bitonickou podposloupnost.

[1] V případě kolize zkusíme další sousední políčko.

