**Jméno:** Petr Valenta **UČO:** 487561



**3. [20 bodů]** Spreading Array je datový typ určený pro uložení posloupnosti položek, nad kterými se vykonávají operace

AddToLeft($x$) na *začátek* posloupnosti přidá položku $x$,

AddToRight($x$) na *konec* posloupnosti přidá položku $x$,

Report($k$) vrátí položku, která je v posloupnosti na pozici $k$, resp. hodnotu Null, jestliže posloupnost má méně než $k$ položek. Operace nemění posloupnost.

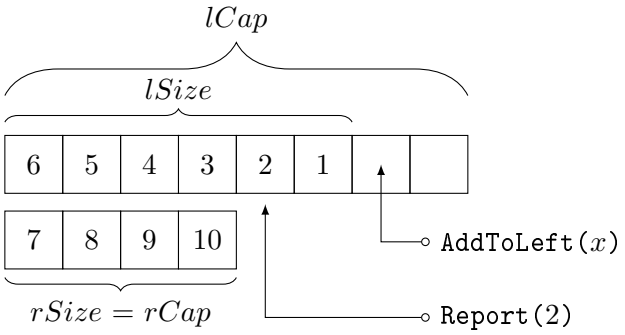
Navrhněte ***jednoduchou*** datovou strukturu, která implementuje datový typ Spreading Array. Požadujeme, aby *amortizovaná* složitost operací AddToLeft a AddToRight byla konstantní, zatímco u operace Report požadujeme konstantní složitost v nejhorším případě. Datová struktura může využívat pouze prostor $\mathcal{O}(n)$, kde $n$ je aktuální délka posloupnosti. Dokažte, že Vámi navržené operace mají požadovanou složitost.

---

**Datová struktura**

Můj návrh Spreading Array používá dvě dynamická pole. První pole obsahuje prvky vložené operací AddToLeft, druhé obsahuje prvky vložené operací AddToRight. Pro práci s těmito poli je nutné u každého pole znát velikost (počet prvků) a kapacitu (možný počet prvků).

Následuje pseudokód operací AddToLeft a Report. Operaci AddToRight neuvádím, její podoba je z operace AddToLeft zřejmá. Vzhledem k použití dvou polí je nutné v operaci Report převést pozici $k$ na index v některém z použitých polí. V pseudokódu uvažuji indexování od nuly.




```
1 function AddToLeft(x)
2     if lSize = lCap then
3         double the capacity of left
4     left[lSize] ← x
5     lSize ← lSize + 1
```

```
1 function Report(k)
2     if k > (lSize + rSize) then
3         return NULL
4     else if k > lSize then
5         return right[k − lSize − 1]
6     else
7         return left[lSize − k]
```

**Jméno:** Petr Valenta **UČO:** 487561

0007 *list* 2 *učo* 4875611 *body*

*Oblast strojově snímaných informací. Své učo a číslo listu vyplňte zleva dle vzoru číslic. Jinak do této oblasti nezasahujte.* 0123456789

**Analýza složitosti**

Operace Report nemění podobu ani jednoho pole a tudíž ji nemusíme zahrnout do analýzy složitosti zbylých dvou operací. Protože operace AddToLeft a AddToRight využívají každá pouze své vlastní pole, neovlivňují si navzájem svou amortizovanou složitost. Provedeme tedy analýzu složitosti pro každou operaci zvlášť.

Report($k$) v nejhorším případě provede dvě aritmetické operace, dvě porovnání a jeden přístup do pole. Neobsahuje žádné cykly ani nevolá žádné další operace. Složitost operace Report tedy nezávisí na velikosti vstupních dat - je konstatní.

AddToLeft($x$) Uvažujme potenciálovou funkci $\Phi(P) = 2 \times lSize - lCap$ a notaci: $P_i$ je stav pole po provedeni $i$-té operace AddToLeft. Platí tedy $\Phi(P_0) = 0$ a zároveň $\Phi(P_i) \geq \Phi(P_0)$, implementací AddToLeft je totiž zaručeno, že vždy platí $lSize \geq lCap/2$. Pro výpočet amortizované ceny pozorujeme u operace AddToLeft dva případy: Levný případ, kdy nedojde k rozšíření pole a drahý případ, kdy je pole zaplněno a je nutné zdvojnásobit jeho kapacitu.

Levný AddToLeft :

$lSize(P_{i-1}) < lCap(P_{i-1})$ Platí: $c_i = 1$ , $lCap(P_i) = lCap(P_{i-1})$ , $lSize(P_i) = lSize(P_{i-1})+1$

$\hat{c}_i = c_i + \Phi(P_i) - \Phi(P_{i-1})$

$\hat{c}_i = 1 + 2 \times lSize(P_i) - lCap(P_i) - (2 \times lSize(P_{i-1}) - lCap(P_{i-1}))$

$\hat{c}_i = 1 + 2 \times (lSize(P_{i-1}) + 1) - lCap(P_{i-1}) - 2 \times lSize(P_{i-1}) + lCap(P_{i-1})$

$\hat{c}_i = 3$

Drahý AddToLeft :

$lSize(P_{i-1}) = lCap(P_{i-1})$ Platí: $c_i = lSize(P_{i-1}) + 1$ , $lCap(P_i) \geq 2 \times lCap(P_{i-1})$ , $lSize(P_i) = lSize(P_{i-1}) + 1$

$\hat{c}_i = c_i + \Phi(P_i) - \Phi(P_{i-1})$

$\hat{c}_i = lSize(P_{i-1}) + 1 + 2 \times lSize(P_i) - lCap(P_i) - (2 \times lSize(P_{i-1}) - lCap(P_{i-1}))$

$\hat{c}_i \leq lSize(P_{i-1}) + 1 + 2 \times (lSize(P_{i-1}) + 1) - 2 \times lCap(P_{i-1}) - 2 \times lSize(P_{i-1}) + lCap(P_{i-1})$

$\hat{c}_i \leq 3 + lSize(P_{i-1}) - lCap(P_{i-1})$

$\hat{c}_i \leq 3$

Amortizovaná cena operace AddToLeft je tedy v obou případech shora omezena konstantou 3. Amortizovaná složitost operace je $\mathcal{O}(1)$.

AddToRight($x$) viz analýza složitosti AddToLeft.

**Prostorová složitost**

Prostorová složitost navržené struktury je nejhorší hned po rozšíření pole. V tomto případě je naalokováno téměř dvakrát více paměti, než je využito: $n = 2k - 1$. Dále je nutné pamatovat si velikost a kapacitu pole. Prostorová složitost je tedy lineární:

$\mathcal{O}(2k - 1 + 2) = \mathcal{O}(2k + 1) = \mathcal{O}(2k) = \mathcal{O}(k)$