Seznamy

Seznam

Seznam představuje datovou strukturu, která umožňuje dynamicky přidávat, přistupovat a odebírat objekty stejného typu. Přidávání objektů do seznamu se obvykle provádí pomocí metody *add()*. Metoda *get(int i)* naopak umožňuje nedestruktivně načíst prvek na pozici i.

Seznam je datová struktura, která v sobě spojuje jak vlastnosti zásobníku, tak vlastnosti fronty.

Implementovat ho pomocí pole je možné, ale není výhodné. Proto si ukážeme jen implementaci pomocí referencí. Takový seznam se jmenuje „jednoduchý spojový seznam“:

- jednoduchý – v každém uzlu je jen jedna reference, takže propojení vede jen jedním směrem,
- spojový – propojení je pomocí spojů, referencí,
- seznam.

Přidat prvek na začátek seznamu

Na začátku máme hlavu seznamu *hlava* (obsahuje *null*) a pomocnou proměnnou pom (reference na instanci třídy Node, také *null*)

Pomocí operátoru *new* vytvoříme nový prvek, na který ukazuje *pom*.

do nového prvku naplníme data:

```
pom.data1 = …
pom.data2 = …
```

Původní obsah proměnné *hlava* překopírujeme do položky *dalsi* toho nového prvku (v našem příkladu tam byla hodnota *null*, proto se do *pom.dalsi* vloží *null*).

```
pom.dalsi = hlava;
```

Nakonec do pointeru hlava nakopírujeme obsah pom, čili adresu nového prvku.

Tím je přidání skončeno.

Poznámka: nemohli jsme provést rovnou *hlava = new*, protože by se nám tím ztratil původní obsah proměnné *hlava*.

Postupovali jsme takto, protože to je univerzální algoritmus, který lze aplikovat i na částečně už zaplněný seznam.

Na obrázku je stav po přidání dalšího prvku do seznamu.

Procházet seznamem

Seznamem se dá procházet jen jedním směrem.

Potřebujeme pomocnou proměnnou *pom*.

Začíná se vždy v *hlava*

```
pom = hlava;
```

a prochází se postupným posouváním proměnné *pom*:

```
pom = pom.dalsi;
```

Poznámka: Povšimněte si, že když procházíme seznam od *hlava* ke konci, pořadí prvků je od nejnovějšího po nejstarší, čili LIFO. Tedy zásobník lze realizovat jako speciální případ seznamu, u kterého přidáváme na začátek a čteme od *hlava*.

Přidat prvek na konec seznamu

Na konec seznamu se přidává špatně, protože bychom museli nejprve ten konec najít (což je zdlouhavé)

Proto vedle reference *hlava* zavedeme ještě další proměnnou *zaKonec*, která bude ukazovat za konec seznamu na pomocný prvek - sentinel.

Za koncem seznamu vytvoříme pomocný prvek, sentinel (nárazník).

Počáteční stav seznamu bude tedy vypadat takto. Ani nově vytvořený seznam tedy nebude prázdný, protože bude obsahovat *sentinel.*

Procházení tohoto seznamu i přidávání prvků na začátek je skoro stejné.

Jediný rozdíl je v tom, že procházení seznamu musíme ukončit ještě **předtím**, než dojdeme na sentinel.

Připojení prvku na konec seznamu se provede v těchto krocích:

- Do sentinelu se naplní data.
- Za původním sentinelem se vytvoří sentinel nový.
- Reference *zaKonec* se přesune na nový sentinel.
- Doplní se *null* v novém sentinelu.

```
zaKonec = zaKonec.dalsi;
```

```
zaKonec.dalsi = null;
```

Seznam x fronta

Povšimněte si, že když procházíme od *hlava* seznam se vkládáním „na konec“, pořadí prvků je od nejstaršího po nejnovější, tedy FIFO.

Tedy frontu lze realizovat jako speciální případ seznamu, u kterého přidáváme na konec a čteme od *hlava*.

Vložení prvku doprostřed

Pro vkládání „doprostřed“ i pro další operace potřebujeme nějak označit místo v seznamu, se kterým pracujeme.

Proto zavedeme další proměnnou *znacka.*

Proměnná *znacka* bude vždy ukazovat na ten prvek seznamu, se kterým zrovna pracujeme.

Při vkládání můžeme nový prvek vkládat **za značku**, nebo **na značku**.

Vkládání za značku

Využijeme pomocnou proměnnou *pom* k vytvoření a naplnění nového prvku.

Adresu následujícího prvku nakopírujeme do nově vytvořeného prvku

```
pom.dalsi = znacka.dalsi;
```

Adresu nového prvku vložíme do pole „dalsi“ prvku, na který ukazuje značka

```
znacka.dalsi = pom;
```

Vkládání před značku

Přestože se zdánlivě jedná o podobný případ, je to mnohem obtížnější.

Důvod: nevíme, odkud vychází modrý pointer, proto ho nemůžeme změnit, jak bychom potřebovali (na červený pointer)

Nyní máme dvě možnosti.

- Najít ten neznámý „předcházející“ prvek a tím to převést na řešení předchozí úlohy „odstranit prvek za značkou“. Je to snadné, ale pomalé.
- Nebo znát „fintu“ (viz dále).

Poznámka: u zkoušky nutno buď znát „fintu“, nebo umět napsat program, jímž se najde „předcházející“ prvek

Výchozí stav pro trik je tento.

Nejprve si prvek, na který ukazuje *značka*, okopírujeme do pomocné proměnné *pom*.

Jen pozor na to, že nemůžeme jednoduše napsat

```
pom = znacka;
```

protože tím by se přiřadily jen reference a ne hodnoty (*pom* i *znacka* by ukazovaly na tentýž prvek). Musíme vytvořit duplikát prvku pomocí metody *copy()*:

```
pom = znacka.copy();
```

Jako poslední krok, nový prvek zapojíme do seznamu a naplníme data

```
znacka.dalsi = pom;
```

Odstranit prvek uprostřed

Podobně jako při vkládání prvků do seznamu, i při odebírání si označíme značkou, který prvek máme odebrat.

Máme v zásadě dvě možnosti: buď odebrat prvek, na který ukazuje značka. Anebo odebrat prvek, který následuje za ním. Obě možnosti si nyní probereme.

Odstranit prvek za značkou

Podobně jako vložit, i odstranit prvek za značkou je snadné.

Do pomocné proměnné *pom* si zapamatujeme adresu rušeného prvku

```
pom = znacka.dalsi
```

Změníme referenci na následující prvek seznamu

znacka.dalsi = pom.dalsi

Nepotřebný prvek zrušíme

```
pom = null;
```

Odstranit prvek se značkou

Přestože se zdánlivě jedná o podobný případ, je to mnohem obtížnější.

Důvod: nevíme, odkud vychází modrý pointer, proto ho nemůžeme změnit, jak bychom potřebovali (na červený pointer)

Nyní máme dvě možnosti.

- Najít ten neznámý „předcházející“ prvek a tím to převést na řešení předchozí úlohy „odstranit prvek za značkou“. Je to snadné, ale pomalé.
- Nebo znát „fintu“ (viz dále).

Poznámka: u zkoušky nutno buď znát „fintu“, nebo umět napsat program, jímž se najde „předcházející“ prvek

Výchozí stav pro trik je tento.

Nejprve si na prvek, následující **za prvkem** označeným značkou, nastavíme pomocnou proměnnou

```
pom = znacka.dalsi;
```

Do prvku, na kterém je značka, nakopírovat **celý** prvek za ním následující.

Úloha se tím vlastně převedla na předcházející úlohu „odstranit prvek za značkou“, kterou už umíme řešit. Jen pozor na to, že nemůžeme jednoduše napsat

```
znacka = pom;
```

protože tím by se přiřadily jen reference a ne hodnoty (*pom* i *znacka* by ukazovaly na tentýž prvek). Musíme vytvořit duplikát prvku pomocí metody *copy()*:

```
znacka = pom.copy();
```

Načež stačí do *pom* vložit *null*, o zbytek se postará *garbage collector*.

```
pom = null;
```

Prvek tím je odstraněn.

Zvláštní druhy seznamů

Seznam je trojice

Trojice pointerů *hlava* + *zaKonec* + *znacka* je pro seznam tak častá a charakteristická, že někteří autoři pod pojmem „seznam“ chápou právě tuto trojici

Realizace seznamu pomocí pole

Byla zde ukázána realizace seznamu pomocí referencí; to ale neznamená, že seznam nelze realizovat například pomocí dynamického pole.

Uspořádaný seznam

Pro seznam je typické, že prvky se identifikují pomocí jejich pozice v seznamu.

Pro zvýšení rychlosti hledání můžeme seznam uspořádat, tzn. jeho prvky seřadit podle hodnot jejich klíčů, ale není to moc výhodné (spojové seznamy nelze snadno půlit, viz metoda půlení).

Struktury, ve kterých jsou prvky identifikovány hodnotou svého klíče, se nazývají tabulka nebo mapa.

Dvojitý seznam

Největší nevýhodou jednoduchého seznamu je, že neumíme rychle najít předcházející prvek (procházení je jednosměrné).

Kde to vadí, tam použijeme dvojitě zřetězený seznam.

Obsah