FAKULTA ústav
STROJNÍHO automatizace
INŽENÝRSTVÍ a informatiky

Doubly linked list

projektová zpráva

12. 12. 2023

David Bystroň, 228676

David Bystroň, 228676

Úvod

V této práci bude využit kód oboustranného listu z počítačových cvičení předmětu VZI. Třída oboustranného listu bude rozšířena o metody pro připojení dalšího listu k původnímu, vkládání jednoho listu do druhého a o třídící algoritmy. Konkrétně o insertion sort a quick sort algoritmus.

Kopírování listu

Pro potřeby vkládání listů byla vytvořena metoda kopírování, která má jako vstup list, který chceme zkopírovat. V kódu je pak označena jako *copy(self, list_to_copy).*

Metoda slouží k tomu, aby při připojování a vkládaní jednotlivých listů nebyly nijak změněny jejich původní parametry a používal se pouze zkopírovaný list.

Kopírování je prováděno pomocí metody *push(self, data).*

Připojení listu

Metoda *add_list(self, list_to_add)*, bere jako vstup další oboustranný list, který chceme připojit na konce původního listu.

Nejprve je list zkopírován a poté jsou ošetřeny 3 různé stavy.

Pokud je původní list prázdný, nastavíme jeho *self.head* a *self.tail* podle listu, který chceme připojit.

Ve chvíli, kdy je list, který chceme připojit prázdný, nevykoná se nic.

V ostatních případech je třeba ošetřit správně jednotlivé ukazatele:

```
self.tail.set_next(new_list.head)

new_list.head.set_prev(self.tail)

self.tail = new_list.tail
```

Vkládání listu

Další metoda *insert_list_after(self, list_to_add, after)* slouží k vložení listu za námi určený znak v původním listu.

Pro fungování metody je třeba, aby žádný z těchto listů nebyl prázdný, jinak se zavolá výjimka.

Stejně jako v předešlé kapitole se nejprve zkopíruje list, který chceme vložit. Pokud znak za který vkládáme byl zároveň posledním znakem, použijeme již naprogramovanou metodu *add_list(self, list_to_add).*

V ostatních případech musíme nastavit správně ukazatele *prev* a *next* u jednotlivých uzlů.

---

insert_list.tail.set_next(current.get_next())

insert_list.head.set_prev(current)

current.get_next().set_prev(insert_list.tail)

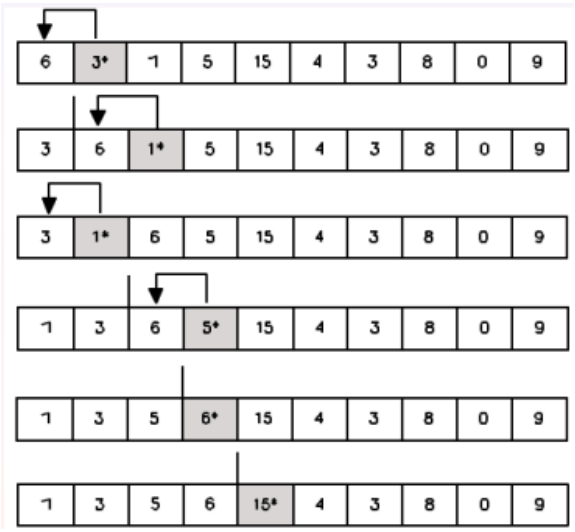
current.set_next(insert_list.head)

Insertion sort

Insertion sort můžeme považovat za jeden z lepších jednoduchých třídících algoritmů. Je vhodný pro částečně setříděné posloupnosti.

Jeho složitost je rovna $O(N^2)$ a je inspirován chováním karetního hráče při jejich třídění.

Algoritmus rozdělí pole na 2 podmnožiny, setříděnou a nesetříděnou. Postupně pak vybírá prvky a posouvá je do setříděné množiny, dokud není prvek na správném místě.



Obr.1) Ukázka insertion sort algoritmu

V kódu je tato metoda označena jako *insertion_sort(self)*. Algoritmus je poměrně jednoduchý, vybereme prvek a když je prvek před ním vetší, prohodíme je. Tak postupujeme, dokud není na správném místě a postupně projdeme celý list.

Nejpodstatnější část kódu je tedy tato smyčka.

```
while next_node is not None and next_node.get_prev() is not None \
        and next_node.get_prev().get_data() > next_node.get_data():
    temp = next_node.get_data()
    next_node.set_data(next_node.get_prev().get_data())
    next_node.get_prev().set_data(temp)

    next_node = next_node.get_prev()
```

Quick sort

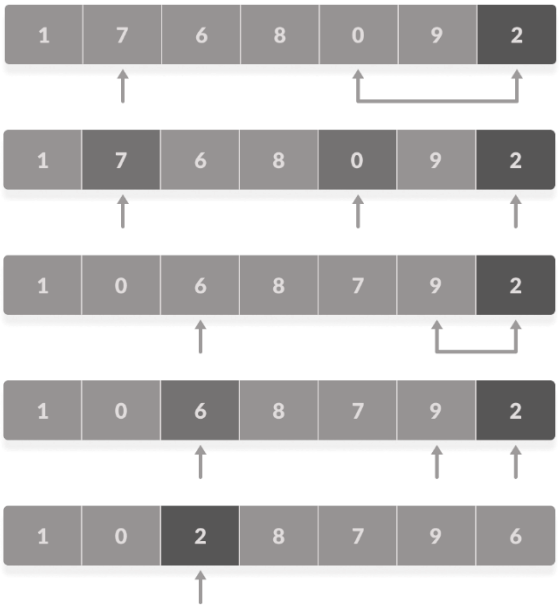
Velmi rozšířený a rychlý algoritmus, využívající princip rozděl a panuj. Lze jej naprogramovat jak pomocí rekurze, tak i nerekurzivně. V tomto projektu byla zvolena rekurzivní forma. V kódu jako *quick_sort(self, left = None, right = None)*.

Pro rychlost a zároveň i složitost algoritmu je důležitá volba tzv. pivota, který bude vysvětlen později. Nejlepší by jako pivot byl medián, ovšem jeho samotné nalezení zvyšuje složitost. Nejlépe se pak jeví volba náhodné prvku. V našem případě bude využita další možnost, a to sice volba prvního prvku jako pivota, respektive v kódu je to poslední prvek.

Princip quick sort algoritmu je pak takový, že po vybrání pivota se zavolá metoda na rozdělení celého pole na část, která je menší než pivot, ta je nalevo, a část kde jsou prvky větší než pivot, vpravo. Samotné části však nejsou setříděné. Na jednotlivé strany pak použijeme opět quick sort. To opakujeme, dokud nemáme pole celé setříděné.

Program pro samotný quick sort není moc složitý, důležitá je ovšem metoda na rozdělení pole na části menší a větší než samotný pivot. V kódu je tato metoda označena jako *partition(self, left, right)*.

Rozdělení probíhá tak, že ve chvíli, kdy máme pivota jako poslední prvek, je třeba si pamatovat pozici prvního většího prvku, než je samotný pivot, dále jen pozice. Poté postupně procházíme prvky a ve chvíli, kdy je prvek menší než pivot, prohodíme jej s prvkem, který je označen pozicí a následně pozici posuneme o jednu. Nakonec pak musíme prohodit pivot s prvkem na pozici, to nám rozdělí pole na dvě části.



Obr.2) Ukázka quick sort algoritmu

V samotném kódu je pak důležité si správně ukládat a posouvat pozici, při prohození dvou prvků.

```
pivot = right
index = left.get_prev()
current = left

while current != right:

    if current.get_data() <= pivot.get_data():
        if index is None:
            index = left
        else:
            index = index.get_next()

        temp = index.get_data()
        index.set_data(current.get_data())
        current.set_data(temp)

    current = current.get_next()
```

Finální prohození pivotu s prvkem na pozici je velmi podobné a není třeba jej zde uvádět.

Samotný quick sort pomocí rekurze pak vypadá následovně.

```
if right is not None and left != right and left != right.get_next():
    pivot = self.partition(left, right)
    self.quick_sort(left, pivot.get_prev())
    self.quick_sort(pivot.get_next(), right)
```

Závěr

V tomto dokumentu je shrnuto několik metod, které byly přidány do třídy oboustranného listu v rámci projektu do předmětu VZI. Tato zpráva obsahuje také důležité úryvky samotného kódu.

V rámci projektu tak byly přidány metody jako *copy*, *add_list*, *insert_list_after*, *insertion_sort*, *partition* a *quick_sort*.