C-LABB

DD1320 Tillämpad datalogi

Jämförelse av OrderedDict och egen implementation av Hashtabell

Marwin Haddad

marwinh@kth.se
19980520-1051

November 2021

Sammanfattning

Syftet med denna rapport är att analysera och jämföra datatyperna OrderedDict och Hashtabell med avseende på styrkor och svagheter samt två aspekter tillhörande dessa. De aspekter som undersöks är spridingen av krockar och krockhantering samt körtiden hos några metoder datatyperna har gemensamt. Den första metoden är att lägga till element och den andra att returnera det värde den inmatade nyckeln innehar. Genom att testa körtiden för dessa metoder har vi sett att OrderedDict hittar ett element i snitt på 0.0016 sekunder och Hashtabell på 0.0373 sekunder. Användningsområdena för implementationerna av en hashtabell är snarlika men har samtidigt sina styrkor och svagheter och bör därmed användas där de hör bäst till.

Uppgiftsbeskrivning

Uppgiften är att göra en jämförelse mellan *två olika datastrukturer* med avseende på *två relevanta aspekter*. Denna rapport jämför *OrderedDict* med vår egna implementation av *Hashtabell* där jämförelsen innehåller:

- en beskrivning av ett konkret exempel där datastrukturerena skulle kunna användas.
- några rader kod som visar hur datastrukturerena fungerar.
- en jämförelse av datastrukturerna med avseende på två relevanta aspekter.
- ett resonemang om datastrukturernas styrkor respektive svageheter.

De aspekter som analyseras av rapporten är krockhantering samt körtid för gemensamma ekvivalenta metoder.

Metod

För att hitta relevant information om OrderedDict studerades källkoden samt officiell python-dokumentation för att få en större uppfattning om relevanta metoder och användningsområden till OrderedDict. Genom följande algoritm (se **Figur 5**) testades OrderedDict:s olika metoder för att få en idé om hur klassen fungerar. Därefter skrevs två ytterligare program som jämför datastrukturerna med avseende på körtid samt krockhantering och krockantal. Krockantalet jämfördes genom att skapa diagram över spridingen av element i datastrukturen (se **Figur 6**) och körtiden jämfördes med modulen *timeit* (se **Figur 7**) som i användes i tidigare laboration. På samma sätt som i tidigare laboration användes Pokemon-objekt för att testa datastrukturerna. Alla tester gjordes med hälften luft i Hashtabellen. Slutligen analyserades datastrukturernas svagheter och styrkor inom de områdena de är avsedda för.

Resultat

Output till **Figur 5** ges av följande:

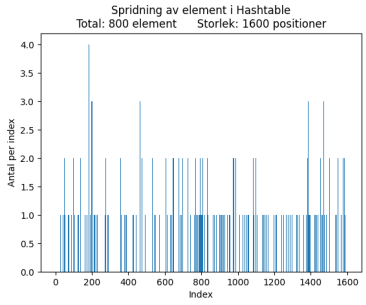
```
OrderedDict([(0, 'zero'), (1, 'one'), (2, 'two'), (3, 'three'), (4, 'four'), (5, 'five'), (6, 'six'), (7, 'seven'), (8, 'eight'), (9, 'nine')])
(0, 'zero')
OrderedDict([(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four'), (5, 'five'), (6, 'six'), (7, 'seven'), (8, 'eight'), (9, 'nine')])
OrderedDict([(3, 'three'), (1, 'one'), (2, 'two'), (4, 'four'), (5, 'five'), (6, 'six'), (7, 'seven'), (8, 'eight'), (9, 'nine')])

Hashtable([(0, 'zero'), ((1, 'one'), (9, 'nine')), ((2, 'two'), (8, 'eight')), (5, 'five'), ((3, 'three'), (7, 'seven')), ((4, 'four'), (6, 'six'))])
zero
Hashtable([(0, 'zero'), ((1, 'one'), (9, 'nine')), ((2, 'two'), (8, 'eight')), (5, 'five'), ((3, 'three'), (7, 'seven')), ((4, 'four'), (6, 'six'))])
```

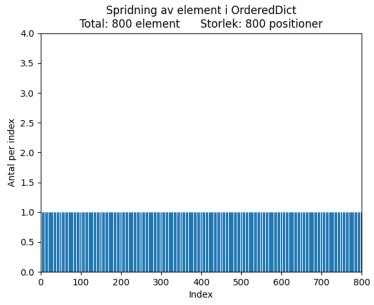
Figur 1:

Krockhantering

Fördelningen av element i respektive datastruktur skildras av följande diagram där varje steg i y-led representeras av antal element och varje steg i x-led av index.



Figur 2:



Figur 3:

Körtid

Tabellen visar körtiden att fylla datastrukturerna givet antalet element.

```
antal element       200      400      800
--------------  ------  -------  -------
Hashtabell       9.5124  19.057   42.0004
OrderedDict      0.1582   0.3138   0.6402

Bland 800 element tog det:
Hashtabell 0.0373s att hitta Pikachu
OrderedDict 0.0016s att hitta Pikachu
```

Figur 4:

Analys

I någon aspekt är en OrderedDict en *queue*-implementation av en dictionary. Sammanhang där det skulle vara relevant att behöva ha kännedom om insättningsordningen i en dictionary och använda vissa relevanta metoder som liknar de tillhörande *queue*-klassen (**popitem** mm.)[1] skulle exempelvis kunna vara för att hålla reda på vilken kund man ska skicka paket till härnäst om *key* skulle motsvara kundens profil och *value* orderinnehåll.
Dictionary-delen av en OrderedDict är i grunden en vanlig python-dictionary. När ett *key-value*-par läggs till sparas nyckeln i en cyklisk länkad lista där varje nod har pekare åt båda hållen. För att hålla koll på var listan börjar används en *dumnod* i början[2]. Eftersom vi endast manipulerar början eller slutet av listan genom att lägga till/ ta bort en nyckel är metoderna i stort sett $O(1)$[2]. Samtidigt som detta sker läggs *key-value*-paret i dictionary:n vilket också är $O(1)$. Styrkorna med OrderedDict är att vi får mer kontroll över dictionary-datatrukturen och kan därmed manipulera den med större frihet. Å andra sidan använder OrderedDict mer minne än en vanlig python-dictionary i och med att OrderedDict är två datastrukturer i en[2].

Ett användningsområde för Hashtabeller skulle kunna vara inom datahantering där Hashtabellen motsvarar databasen. Exempelvis skulle Hashtabellen kunna användas för att verifiera lösenord vid inloggningsprocessen till canvas där *key* motsvarar användarnamnet och *value* lösenordet användaren matar in i lösenordsfältet. Skulle det inmatade lösenordet överensstämma med värdet nyckeln motsvarar loggas användaren in.
Styrkan hos en Hashtabell är att det ger direkt tillgång till föremålet nyckeln besitter. Å andra sidan är nackdelen med vår implementation av Hashtabellen att tabellen inte är muterbar. Skulle det bli så att vi lagt till så många element att krockar är absolut oundvikliga behöver vi skapa en till Hashtabell som är större och sedan föra över alla *key-value*-paren från den övermättade tabellen. Detta innebär att hasha alla nycklar på nytt för att ta hänsyn till den nya tabellens storlek.

Av resultatet ser vi att OrderedDict har bättre spridning än Hashtabell. Anledningen till detta ges av att krockhanteringen är en form av probning beroende på tabellens storlek[3] medan vi i Hashtabell använder krocklistor där vi tillåter att flera element ligger på samma plats. Probningen fungerar som sådan att nästa index som undersöks fås genom $j_{n+1} = ((5 \cdot j_n) + 1) \mod 2^i$ där j motsvarar nuvarande index och 2^i storleken på tabellen för positiva heltal i. Skulle inte den första platsen vara tom säger man att $j_{n+1} = ((5 \cdot j_n) + \textit{förskjutninsvariabel}) \mod 2^i$ där just $\textit{förskjutninsvariabel} = 5$ används för att det visat sig ge bäst spridning genom experiment[3]. Vi ser även av resultatet att insättning av element måste vara $O(n)$ för båda datastrukturerna. Anledningen till att OrderedDict var snabbare kan ha att göra med en effektivare hashfunktion och krockhantering vilket också skulle förklara varför det går snabbare för OrderedDict att hitta och returnera det sökta elementet.

Appendix

```python
from collections import OrderedDict
from hashClassFile import Hashtable


def ordered(arr):
    o = OrderedDict()
    for i, x in enumerate(arr):
        o[i] = x
    print(o)
    print(o.popitem(last=False))
    print(o)
    o.move_to_end(3, last=False)
    print(o)
    print()


def table(arr):
    h = Hashtable(len(arr)*2)
    for i, x in enumerate(arr):
        h[i] = x
    print(h)
    print(h[0])
    print(h)


def main():
    arr = ['zero', 'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']
    ordered(arr)
    table(arr)


if __name__ == '__main__':
    main()
```

Figur 5:

```python
def get_chart(h, o):
    total = 0
    index_h = []
    count_h = []
    index = 0
    for current in h.table:
        index_h.append(index)
        count = 0
        while current:
            count += 1
            current = current.next
        index += 1
        total += count
        count_h.append(count)

    plt.figure(1)
    plt.bar(index_h, count_h)
    plt.title('Spridning av element i Hashtable\n'
              f'Total: {total} element      Storlek: {h.size} positioner')
    plt.xlabel('Index')
    plt.ylabel('Antal per index')

    plt.figure(2)
    plt.bar([i for i in range(len(o))], [1 for _ in range(len(o))])
    plt.title('Spridning av element i OrderedDict\n'
              f'Total: {total} element      Storlek: {len(o)} positioner')
    plt.xlabel('Index')
    plt.ylabel('Antal per index')
    plt.axis([0, len(o), 0, 4])
    plt.show()


def main():
    pokemon_list = read_cvs()
    h = Hashtable(len(pokemon_list)*2)
    o = OrderedDict()
    h = fill_dict(h, pokemon_list)
    o = fill_dict(o, pokemon_list)
    get_chart(h, o)
```

Figur 6:

```python
def fill_dict(d, arr):
    for x in arr:
        d[x.name] = x
    return d


def time_find(d, searchkey):
    return d[searchkey]


def main():
    pokemon_list = read_cvs()
    h = Hashtable(len(pokemon_list)*2)
    o = OrderedDict()
    hash_200 = timeit.timeit(stmt=lambda: fill_dict(h, pokemon_list[:200]), number=10000)
    ord_200 = timeit.timeit(stmt=lambda: fill_dict(o, pokemon_list[:200]), number=10000)
    hash_400 = timeit.timeit(stmt=lambda: fill_dict(h, pokemon_list[:400]), number=10000)
    ord_400 = timeit.timeit(stmt=lambda: fill_dict(o, pokemon_list[:400]), number=10000)
    hash_800 = timeit.timeit(stmt=lambda: fill_dict(h, pokemon_list), number=10000)
    ord_800 = timeit.timeit(stmt=lambda: fill_dict(o, pokemon_list), number=10000)
    h = fill_dict(h, pokemon_list)
    o = fill_dict(o, pokemon_list)
    hash_find = timeit.timeit(stmt=lambda: time_find(h, 'Pikachu'), number=10000)
    ord_find = timeit.timeit(stmt=lambda: time_find(o, 'Pikachu'), number=10000)
    table = [['antal element', '200', '400', '800'],
             ['Hashtabell', round(hash_200, 4), round(hash_400, 4), round(hash_800, 4)],
             ['OrderedDict', round(ord_200, 4), round(ord_400, 4), round(ord_800, 4)]]
    print(tabulate(table, headers='firstrow'))
    print('\nEfter 800 element tog det:')
    print(f'Hashtabell {round(hash_find, 4)}s att hitta Pikachu\n'
          f'OrderedDict {round(ord_find, 4)}s att hitta Pikachu')
```

Figur 7:

Referenser

[1] Python Software Foundation, "$class collections.OrderedDict([items])$", hämtad 19-11-2021,
"$https://docs.python.org/3/library/collections.html$"

[2] Public, hämtad 19-11-2021,
"$https://github.com/python/cpython/tree/main/Lib/collections$"

[3] Public, hämtad 20-11-2021,
"$https://github.com/python/cpython/blob/main/Objects/dictobject.c$"