Lecture 9, 21 Oct 2021

Lists, arrays, dictionaries: implementation details

Arrays

- Contiguous block of memory, size is declared in advance, all values uniform
- **Random access** -- accessing the value at `a[i]` does not depend on `i`
- Inserting or deleting a value is expensive
- Need to shift elements right or left, respectively, depending on the location of the modification

Lists

- Each location is a *cell*, consisiting of a value and a link to the next cell
- To reach cell `l[i]`, time proportional to `i`: traverse the links from `l[0]` to `l[i]`
- On the other hand, if we are already at `l[i]` modifying the list is easy
- Each insert/delete requires a fixed amount of local "plumbing", independent of where in the list it is performed

Dictionaries

- Values are stored in a fixed block of size m
- Keys are mapped by a hash function to $\{0, 1, \ldots, m-1\}$
- Lookup requires computing $h(k)$ which takes roughly the same time for any k
- Collisions are inevitable, different mechanisms to manage this, which we will not discuss now
- Effectively, a dictionary combines flexibility with random access
- **Why does Python insist that keys are immutable values?**

Lists in Python

- Flexible size, allow inserting/deleting elements in between
- However, implementation is an array, rather than a list
- Initially allocate a block of storage to the list
- When storage runs out, double the allocation
- `l.append(x)` is efficient, moves the right end of the list one position forward within the array
- `l.insert(0,x)` inserts a value at the start, expensive because it requires shifting all the elements by 1
- We will run experiments to validate these claims

```
1 import time
2
3 class TimerError(Exception):
```

```
    """A custom exception used to report errors in use of Timer clas

class Timer:
    def __init__(self):
        self._start_time = None
        self._elapsed_time = None

    def start(self):
        """Start a new timer"""
        if self._start_time is not None:
            raise TimerError("Timer is running. Use .stop()")
        self._start_time = time.perf_counter()

    def stop(self):
        """Save the elapsed time and re-initialize timer"""
        if self._start_time is None:
           raise TimerError("Timer is not running. Use .start()")
        self._elapsed_time = time.perf_counter() - self._start_time
        self._start_time = None

    def elapsed(self):
        """Report elapsed time"""
        if self._elapsed_time is None:
           raise TimerError("Timer has not been run yet. Use .start(
        return(self._elapsed_time)

    def __str__(self):
        """print() prints elapsed time"""
        return(str(self._elapsed_time))
```

```
t = Timer()
t.start()
l = []
for i in range(10000000):
    l.append(i)
t.stop()
print(t)
```

```
1.7382565920006527
```

```
t = Timer()
t.start()
l = []
for i in range(300000):
    l.insert(0,i)
t.stop()
print(t)
```

```
22.900079012999413
```

```
t = Timer()
t.start()
d = {}
for i in range(10000000,0,-1):
    d[i] = i
t.stop()
print(t)
```

```
1.9490711960006593
```

```
def createlist():  # Equivalent of l = [] is l = createlist()
  return({})

def listappend(l,x):
  if l == {}:
    l["value"] = x
    l["next"] = {}
    return

  node = l
  while node["next"] != {}:
    node = node["next"]

  node["next"]["value"] = x
  node["next"]["next"] = {}
  return

def listinsert(l,x):
  if l == {}:
    l["value"] = x
    l["next"] = {}
    return

  newnode = {}
  newnode["value"] = l["value"]
  newnode["next"] = l["next"]
  l["value"] = x
  l["next"] = newnode
  return


def printlist(l):
  print("{",end="")

  if l == {}:
    print("}")
    return
  node = l

  print(node["value"],end="")
```

```
41   while node["next"] != {}:
42     node = node["next"]
43     print(",",node["value"],end="")
44   print("}")
45   return
46
```

```
1 t = Timer()
2 t.start()
3 l = createlist()
4 for i in range(10000):
5     listappend(l,i)
6 t.stop()
7 print(t)
```

```
6.1525952339998184
```

```
1 t = Timer()
2 t.start()
3 l = createlist()
4 for i in range(1000000):
5     listinsert(l,i)
6 t.stop()
7 print(t)
```

```
1.630923595999775
```