Outline

Overview

Programs manipulating large data sets

When we start considering programs that manipulate large data sets, we need to use more efficient algorithms.

Often the key to an efficient algorithm lies in how the data is organized, the so-called **data structures**.

OO-programs often use **container classes** to manage collections of objects.

Examples: lists, dictionaries in Python

Overview

Programs manipulating large data sets

When we start considering programs that manipulate large data sets, we need to use more efficient algorithms.

Often the key to an efficient algorithm lies in how the data is organized, the so-called **data structures**.

OO-programs often use **container classes** to manage collections of objects.

Examples: lists, dictionaries in Python

Overview

Programs manipulating large data sets

When we start considering programs that manipulate large data sets, we need to use more efficient algorithms.

Often the key to an efficient algorithm lies in how the data is organized, the so-called **data structures**.

OO-programs often use **container classes** to manage collections of objects.

Examples: lists, dictionaries in Python

Interface for the list Class

Lists are Containers

A container class provides objects which contain collections of other objects.

Usually, containers are homogeneous—the data is all of one type. But a Python list can contain string, int, and float values at the same time.

Interface for the list Class

Python list Method Specifications

- Concatenation `list1 + list2`
- Repetition `list1 * int1` or `int1 * list1`
- Length `len(list1)`
- Index `list1[i]`
- Slice `list1[start:stop:step]`
- Membership `item in list1`
- Append `list1.append(obj1)`
- Insert `list1.insert(int1, obj1)`
- Delete index `list1.pop(i)`
- Remove object `list1.remove(obj1)`

Interface for the list Class

Python list Method Specifications

see more at
https://docs.python.org/3/tutorial/datastructures.html

Interface for the list Class

Slice `list1[start:stop:step]`

Parameter Values for slicing

- The `step` parameter in the slice operation can be negative (it means step backwards).
- If the `step` parameter is missing, its default value is assumed to be 1.

  (The book only shows start and stop. This will work to step through each value, since the default step is 1. But to skip the odd indices, say, you would use step = 2.)

Array-based Lists

Python's List Implementation

- Computer memory is a sequence of storage locations
- Each storage location has an address (i.e. a number) associated with it
- A single data item may be stored across a number of contiguous memory locations
- To retrieve an item from memory we need a way to either look up or compute the starting address of the object
- To store a collection of objects, we need to have a systematic method for figuring out where each object in the collection is located.

Array-based Lists

Arrays

- An array is a collection of adjacent memory objects all of the same size (a simple method for storing a collection is to allocate a single contiguous area of memory) .
- Usually, the objects in an array are all of the same class. (We then say that the array is **homogeneous**)
- Each object has an index giving its position in the array. The first item has index zero, the next item has index one, and so on.
- The values can be accessed efficiently by index.

Array-based Lists

Arrays

Example: Imagine having an array of integers (each having size four bytes) stored at address 1024.

The first item (with index 0) is stored at locations 1024-1027, the next item is stored at locations 1028-1031.

The item with index i is at address $1024 + 4 \times i$

- quick to compute whenever the value of item i must be accessed or changed (in constant time, or $\Theta(1)$).

Array-based Lists

Pros of Arrays

- arrays are very memory efficient
- arrays support quick random access
  (i.e. we can "jump" directly to the item we want)

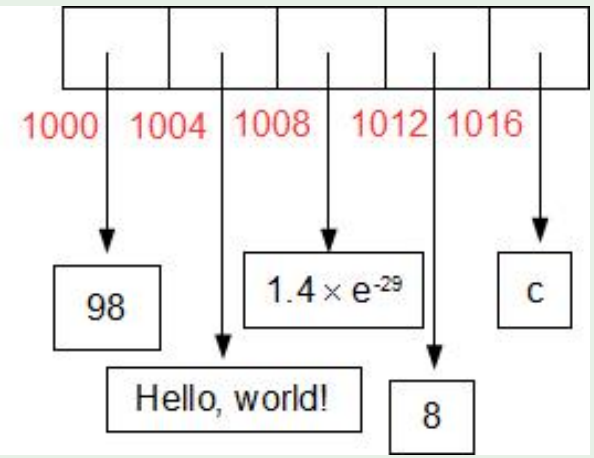
Difficulties with Arrays

- Heterogeneous data cannot be kept in an array, since items may have different sizes, with no easy formula to find items by index efficiently.
- Adding to a Full Array If the block allocated for an array has filled up, appending new items is not easy. The adjacent space after it might be unavailable, holding other information. (arrays are said to be **static**)

Array-based Lists

Python Overcomes These Difficulties

- In spite of these difficulties, Python overcomes these problems, and uses arrays to implement list objects.
- In Python, lists are arrays of references, which are memory addresses of the actual data objects in scattered locations in memory. Thus heterogeneous data can be handled, since the addresses themselves are all the same size (32 or 64 bits).



Array-based Lists

Python Overcomes These Difficulties

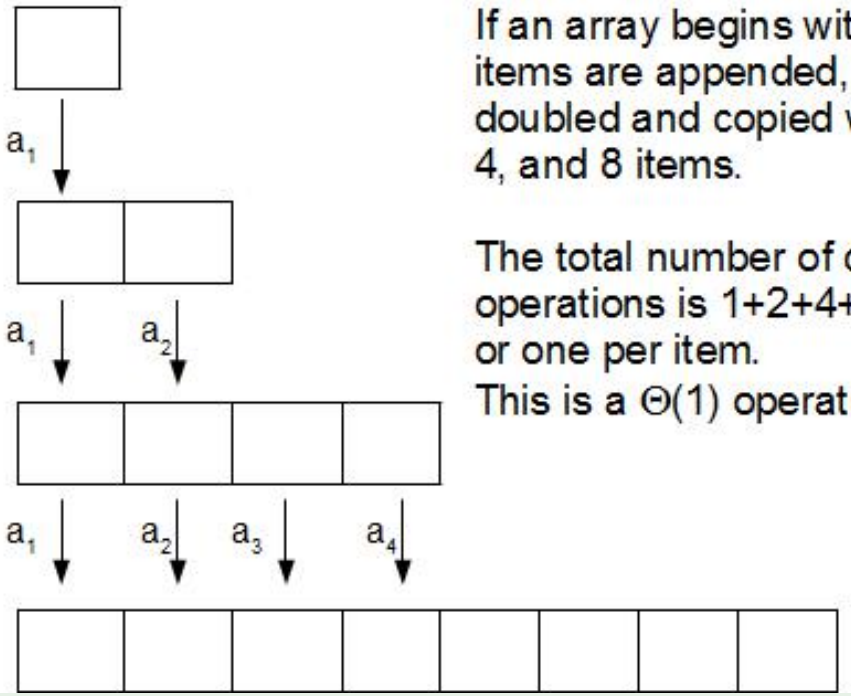
Adding to a Full Array:

In Python, if a list needs more space, a larger array is allocated, and the old array is copied into it with room to spare for appending new items.

Efficiency Analysis

List Operations

- append By doubling the size of an array each time it gets full, the average cost of appending a single item is a constant amount.



If an array begins with size 1, and 15 items are appended, the array gets doubled and copied when it has 1, 2, 4, and 8 items.

The total number of copying operations is 1+2+4+8 = 15, or one per item.
This is a $\Theta(1)$ operation.

Efficiency Analysis

List Operations

- insert To insert a single item into an array, all the items after it must be copied into the next higher locations.
  On average, this takes $\frac{1}{2}n$ operations, which is $\Theta(n)$.

- delete Similarly, to delete a single item from the middle of an array, all the items after it must be copied into the next lower locations.
  This is also $\Theta(n)$.

A Dictionary ADT

A Dictionary is Like a List

- You can access an item of a list by supplying its **index**: `l[10]`, `l[1]`, `l[4]` give the values at position `10`, `1`, and `4` respectively of list `l`.
  There is a function (mapping) from indexes to item values, where integers are the indexes.

- We think of each integer index as a key to its data.

- A Python Dictionary lets you use a value of *any immutable type* as a valid key.

A Dictionary ADT

Dictionary Operations

- `Create` Returns an empty dictionary.
- `put(key, value)` Associates the value `value` with `key` in the dictionary.
- `get(key)`
  pre: There is an `X` such that `(key, X)` is in the dictionary.
  post: Returns `X`.
- `delete(key)`
  pre: There is an `X` such that `(key, X)` is in the dictionary.
  post: `(key, X)` is removed from the dictionary.

Python Dictionaries

Example: A Suits Dictionary

```
>>> suits = {"c":"Clubs", "d":"Diamonds",
"h":"Hearts", "s":"Spades"}
>>> suits.get("c")
'Clubs'
>>> suits["c"]
'Clubs'
>>> suits["j"]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
KeyError:  'j'
```

Dictionary Implementation

Hash Tables

Hash tables are the structures used to implement Dictionaries.

They use a function (called a hash function - **the heart** of the hash table) which quickly ($\Theta(1)$) computes the index into an array from a key value (which may be a text string, for example).

The values of the dictionary are items referenced in that array, so they can be found in constant time from their keys, regardless of the size of the dictionary.

We will cover hash tables in detail later this semester.

Suggested self-development assignment

array and set in Python

Investigate two data types in Python: array and set in Python.

Note that in order to use arrays one needs to import the library, while set is a built-in type.

Work with them!

Try to create arrays and sets of integers. See what happens with duplicate values. Try to add/delete/replace elements.

Note that it is important to get information from *Python documentation*, not from *stack overflow*.

Suggested project, p. 104 / 9

A simple Solitare game

Write a program to play the following simple solitaire game. N cards are dealt face up onto the table. If two cards have a matching rank, new cards are dealt face up on top of them. Dealing continues until the deck is empty or no two stacks have matching ranks. The player wins if all the cards are dealt . Run simulations to find the probability of winning with various values of N.