STAT 534

Lecture 3

Data Structure

April 9th,2019

mmp@stat.washington.edu
Scribes: Dehai Liu

1 Python Object

non-object: simple types, such as integer, character, etc.
object: belongs to class(type), such as list, string and array.

When we use object, we also need to consider:

- How the data is stored
- What operation are performed on the data
- Independence to the programming language being used

2 Theorectical Computer Science

In this part, we will explore ways to organize data that make operations on it more efficiently.

Example Read n strings of length k and turn it into a long string.

Algorithm 1 Long String

```
1: Initialize long string ls
2: for each i ∈ [0, n − 1] do
3:     Read string s;
4:     ls ← ls + s
5: end for
```

- Runtime of concatenating strings: $ik/iter$
- Runtime of allocation of string s: $1/iter$
- Total runtime: $O(n^2) \quad write + O(n) \quad allocation$

Algorithm 2 Array

1: Initialize array la with nk elements
2: $l \leftarrow 0$
3: **for** each $i \in [0, n-1]$ **do**
4: Read string s;
5: $la[l : l + k] \leftarrow s$
6: $l \leftarrow l + k$
7: **end for**

- Runtime of assigning s to la: $k/iter$
- Runtime of allocation of string s: 1 allocation for the array la
- Total runtime: $O(n) \quad write + O(1) \quad allocation$

Algorithm 3 Python List - Append

1: Initialize python list pl
2: **for** each $i \in [0, n-1]$ **do**
3: Read string s;
4: Append s to pl (If out of space,allocate more space to pl)
5: **end for**

- Runtime of appending s to pl: $k/iter$
- Runtime of allocating space: $log_2 n$
- Total runtime: $O(n) \quad write + O(log_2 n) \quad allocation$

Algorithm 4 Python List - Insert

1: Initialize python list pl
2: **for** each $i \in [0, n-1]$ **do**
3: Read string s;
4: Insert s at front of pl (If out of space,allocate more space to pl)
5: **end for**

- Runtime of appending s to pl: $k(i+1)/iter$
- Runtime of allocating space: $log_2 n$
- Total runtime: $O(n^2) \quad write + O(log_2 n) \quad allocation$

3 (Abstract) Data Structure

Array Static

- Allocate the space at once
- Fixed size of n

(Double) Linked List Dynamic

Supposed we define runtime of "Easy" Operation to be $O(1)$ and "Hard"/"Slow" Operation to be $O(n)$. Then the runtime of the following operations are:

- Output elements in order: $O(n)$
- Append at the end: $O(1)$
- Prepend(insert at the beginning): $O(1)$
- Insert at location v_i: $O(1)$
- Delete at location v_i: $O(1)$
- Access the ith element when not knowing its location: $O(n)$ (traverse the linked list)

Stack A list or an array

Operations and their corresponding runtime of stack are:

- Push(append at the end): $O(1)$
- Pop(Output the last element and delete it): $O(1)$

Application of stack : Function calls

```
# main program
x = ···
y = f1(x,3)
# Functions
f1(x1,x2):
  f2(a,b)
f2(a,b):
  f3(···)
```

Main program, `f1`, `f2` and `f3` are pushed into the stack sequentially, then the return values of each function above pop out in reverse order.

4 List Algorithm

- Bubble sort ($O(n^2)$): Compare the value with its neighbor and switch them if they are not in order.
- Heap sort ($O(nlogn)$): Efficient algorithm for sorting problems.