A Reimplementation of Pythons dict using Simple Tabulation Hashing and Linear Probing

Russell Cohen Thomas Georgiou Pedram Razavi

May 17, 2012

Abstract

This is the paper's abstract ...

1 Introduction

In this project we aim to analyze and improve the performance of Pythons dictionary or dict.

Outline

Motivation Python heavily utilizes dictionary data structure for several of its internal language features. This extensive internal use of dictionaries means that a running Python program has many dictionaries active simultaneously, even in cases where the the users program code does not explicitly use a dictionary. For instance, a dictionary can be instantiated to pass keyword arguments to a function. This means a dictionary could potentially be created and destroyed on every function call. Some other examples of internal usage of dictionaries in Python are as follows:

Class method lookup Instance attribute lookup and global variables lookup Python built-ins module Membership testing

Later in this paper, we compare the performance of our new dictionary implementation for some of these special cases. The aforementioned applications of the dictionaries further highlights the need for the fast instantiation

and deletion of dictionaries so that less memory is utilized once running a program. Aside from the issue of memory usage, to further enhance the runtime of a program we need fast key/value storage and retrievals which is the main focus of this paper. Therefore it becomes clear that a better dictionary structure will have significant effects on total Python performance.

Current Python Implementation In this section we give a brief overview of the key implementation details of dictionaries in CPython 2.7.3. We should note that Pythons dictionary supports several different data types as keys. However as we noted earlier, the dictionaries underlying class instances and modules have only strings as keys. Therefore if we can

but contains a specialized string-key-only version for performance in this common case. In this project we will focus more on the string data type as the key since most of the mentioned language features use only strings. The current Python dictionary implementation uses an iterative XORed polynomial terms for calculating the hash of the strings. The collision resolution strategy currently used is open addressing with a custom non-linear probing scheme. We will try to implement a hopefully more performant alternative to this strategy consisting of variants of simple tabulation hashing for variable length strings with linear probing.

The Theoretical Results and Way to improve implementation Recently, Patrascu et al. [2] showed that simple tabulation hashing provides unexpectedly strong guarantees and is constant competitive with a truly random hash function for hashing with linear probing. They also showed some experimental evaluations which proved that simple tabulation on fixed input length can be fast and robust when used with linear probing and cuckoo hashing. We plan to implement this scheme and analyze how it performs as a replacement for Pythons dict with the hopes of yielding better performance.

Technical details of implementation Main Results Benchmarks on real world apps Conclusion and future work

2 Previous work

A much longer LaTeX 2_ε example was written by Gil [**?**].

3 Results

In this section we describe the results.

4 Conclusions

We worked hard, and achieved very little.