[Python-Dev] More compact dictionaries with faster iteration

Raymond Hettinger raymond.hettinger at gmail.com
Mon Dec 10 02:44:30 CET 2012

- Previous message: [Python-Dev] hg annotate is broken on hg.python.org
- Next message: [Python-Dev] More compact dictionaries with faster iteration
- **Messages sorted by:** [date] [thread] [subject] [author]

```
The current memory layout for dictionaries is
unnecessarily inefficient.  It has a sparse table of
24-byte entries containing the hash value, key pointer,
and value pointer.

Instead, the 24-byte entries should be stored in a
dense table referenced by a sparse table of indices.

For example, the dictionary:

    d = {'timmy': 'red', 'barry': 'green', 'guido': 'blue'}

is currently stored as:

    entries = [['--', '--', '--'],
               [-8522787127447073495, 'barry', 'green'],
               ['--', '--', '--'],
               ['--', '--', '--'],
               ['--', '--', '--'],
               [-9092791511155847987, 'timmy', 'red'],
               ['--', '--', '--'],
               [-6480567542315338377, 'guido', 'blue']]

Instead, the data should be organized as follows:

    indices =  [None, 1, None, None, None, 0, None, 2]
    entries =  [[-9092791511155847987, 'timmy', 'red'],
                [-8522787127447073495, 'barry', 'green'],
                [-6480567542315338377, 'guido', 'blue']]

Only the data layout needs to change.  The hash table
algorithms would stay the same.  All of the current
optimizations would be kept, including key-sharing
dicts and custom lookup functions for string-only
dicts.  There is no change to the hash functions, the
table search order, or collision statistics.

The memory savings are significant (from 30% to 95%
compression depending on the how full the table is).
Small dicts (size 0, 1, or 2) get the most benefit.

For a sparse table of size t with n entries, the sizes are:

    curr_size = 24 * t
    new_size = 24 * n + sizeof(index) * t
```

In the above timmy/barry/guido example, the current
size is 192 bytes (eight 24-byte entries) and the new
size is 80 bytes (three 24-byte entries plus eight
1-byte indices). That gives 58% compression.

Note, the sizeof(index) can be as small as a single
byte for small dicts, two bytes for bigger dicts and
up to sizeof(Py_ssize_t) for huge dict.

In addition to space savings, the new memory layout
makes iteration faster. Currently, keys(), values, and
items() loop over the sparse table, skipping-over free
slots in the hash table. Now, keys/values/items can
loop directly over the dense table, using fewer memory
accesses.

Another benefit is that resizing is faster and
touches fewer pieces of memory. Currently, every
hash/key/value entry is moved or copied during a
resize. In the new layout, only the indices are
updated. For the most part, the hash/key/value entries
never move (except for an occasional swap to fill a
hole left by a deletion).

With the reduced memory footprint, we can also expect
better cache utilization.

For those wanting to experiment with the design,
there is a pure Python proof-of-concept here:

   http://code.activestate.com/recipes/578375

YMMV: Keep in mind that the above size statics assume a
build with 64-bit Py_ssize_t and 64-bit pointers. The
space savings percentages are a bit different on other
builds. Also, note that in many applications, the size
of the data dominates the size of the container (i.e.
the weight of a bucket of water is mostly the water,
not the bucket).

Raymond

- Previous message: [Python-Dev] hg annotate is broken on hg.python.org
- Next message: [Python-Dev] More compact dictionaries with faster iteration
- **Messages sorted by:** [date] [thread] [subject] [author]

More information about the Python-Dev mailing list