Hashing (IV): Converting Non-integer Keys to Integers

CSCD 300 – Data Structures

Eastern Washington University

EASTERN WASHINGTON UNIVERSITY
start something big

Goal

So far we have been assuming the id of an element that we are hashing is always an integer, but in practice it's always possible that the id can be a non-integer. In this lecture, we introduce a few tricks/choices for people to use to convert a non-integer id to an integer, so we can then proceed to hash the elements using the hashing techniques that we have discussed in the previous lectures.

EASTERN WASHINGTON UNIVERSITY
start something big

Outline

EASTERN WASHINGTON UNIVERSITY
start something big

Guiding rule

In order to avoid future hashing collisions, we want the conversion

- spreads the non-integer keys as uniformly as possible.
- the same non-integer id must always be converted into the same integer.

Convert `byte`, `short`, `char`, `float` to `int`

Simply cast the `byte`, `short`, `char`, `float` data type to the `int` type.

- If the id is of `byte`, `short`, `char`: `new_id = (int)id`
- If the id is of `float` type: `new_id = Float.floatToIntBits(id)`

Convert `long, double` to int

`long` and `double` both have eight bytes. The conversion is simply to add the upper four bytes with the lower four bytes and use the summation as the new id.

For example:

```
static int long2int(long id){
   return (int)((id >> 32) + (int)id);
}
```

Arithmetic overflow is possible, but we don't care and just use whatever is left from the overflow.

Convert `String` to int: the polynomial method

View each byte (or character) in the string as one integer and view all the characters in the string as the coefficients of a polynomial.

For example, suppose the given id of string type is: $a_0 a_1 a_2 \ldots a_{n-1}$. This id will be converted into the following integer:

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{n-1} x^{n-1}$$

where x is a contant that you pick and it has been proved good from practice if $x \in \{33, 37, 39, 41\}$.

In practice, a must faster way to calculate the value of the above polynomial is to use the Horner's rule [a], so that the time cost of the computation is $O(n)$:

$$a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-2} + a_{n-1} x) \cdots))$$

[a] `http://en.wikipedia.org/wiki/Horner's_method`

(continue ...)

Horner's rule

$$\begin{aligned} & a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{n-1} x^{n-1} \\ = \quad & a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-2} + a_{n-1} x) \cdots)) \end{aligned}$$

Implementation

```
static int string2int_Horner(String s, int x){
   int n = s.length();
   int result = (int)s.charAt(n-1);
   for(int i = n - 2; i >= 0; i--)
      result = (int)s.charAt(i) + result * x;
   return result;
}
```

Arithmetic overflow is possible, but we don't care and just use whatever is left from the overflow.

Convert String to int: the bit cyclic shift method

We initialize an integer whose value is 0 (or any other value that you like). For each character in the string, we first cyclic shift the bits of the integer, then add the character's value into the integer. Repeat the same thing, until all characters are processed and then we output the integer as the integer id of the string.

For example, using 5-bit cyclic shift.

```
static int string2int_shift(String s){
   int h = 0;
   for(int i = 0; i < s.length(); i++){
      h = (h << 5) | (h >> 27); // 5-bit cyclic shift of the integer h
      h = h + (int)s.charAt(i); // add the character's value into h
   }
   return h;
}
```

(continue ...)

Arithmetic overflow is possible, but we don't care and just use whatever is left from the overflow.

Question for you

Understand why the first line of code inside the `for` loop is doing a 5-bit cyclic shift of the integer h ? Find the answer by tracing an example for that line of code.