Rabin-Karp algorithm

Robin Visser

What is Rabin-Karp?

What is Rabin-Karp?

- String searching algorithm

What is Rabin-Karp?

- String searching algorithm
- Uses hashing

What is Rabin-Karp?

- String searching algorithm
- Uses hashing

(*e.g.* `hash("answer") = 42`)

Algorithm

Algorithm

```
function RabinKarp(string s[1..n], string sub[1..m])
hsub := hash(sub[1..m]); hs := hash(s[1..m])
for i from 1 to n-m+1
    if hs = hsub
        if s[i..i+m-1] = sub
            return i
    hs := hash(s[i+1..i+m])
return not found
```

Algorithm

```
function RabinKarp(string s[1..n], string sub[1..m])
hsub := hash(sub[1..m]); hs := hash(s[1..m])
for i from 1 to n-m+1
    if hs = hsub
        if s[i..i+m-1] = sub
            return i
    hs := hash(s[i+1..i+m])
return not found
```

Naïve implementation: Runs in O(nm)

Hash function

Hash function

- Use rolling hash to compute the next hash value in constant time

Hash function

- Use rolling hash to compute the next hash value in constant time

Example: If we add the values of each character in the substring as our hash, we get:

```
hash(s[i+1..i+m]) = hash(s[i..i+m-1]) - hash(s[i])
+ hash(s[i+m])
```

Hash function

- A popular and effective hash function treats every substring as a number in some base, usually a large prime.

Hash function

- A popular and effective hash function treats every substring as a number in some base, usually a large prime.

  (e.g. if the substring is "IOI" and the base is 101, the hash value would be $73 \times 101^2 + 79 \times 101^1 + 73 \times 101^0 = 752725$)

Hash function

- A popular and effective hash function treats every substring as a number in some base, usually a large prime.

  (e.g. if the substring is "IOI" and the base is 101, the hash value would be $73 \times 101^2 + 79 \times 101^1 + 73 \times 101^0 = 752725$)

  Due to the limited size of the integer data type, modular arithmetic must be used to scale down the hash result.

How is this useful

How is this useful

- Inferior to KMP algorithm and Boyer-Moore algorithm for single pattern searching, however can be used effectively for multiple pattern searching.

How is this useful

- Inferior to KMP algorithm and Boyer-Moore algorithm for single pattern searching, however can be used effectively for multiple pattern searching.

- We can create a variant, using a Bloom filter or a set data structure to check whether the hash of a given string belongs to a set of hash values of patterns we are looking for.

How is this useful

Consider the following variant:

How is this useful

Consider the following variant:

```
function RabinKarpSet(string s[1..n], set of string
subs, m):
    set hsubs := emptySet
    for each sub in subs
        insert hash(sub[1..m]) into hsubs
    hs := hash(s[1..m])
    for i from 1 to n-m+1
        if hs ∈ hsubs and s[i..i+m-1] ∈ subs
            return i
        hs := hash(s[i+1..i+m])
    return not found
```

How is this useful

- Runs in $O(n + k)$ time, compared to $O(nk)$ time when searching each string individually.

How is this useful

- Runs in O(n + k) time, compared to O(nk) time when searching each string individually.

- Note that a hash table checks whether a substring hash equals any of the pattern hashes in O(1) time on average.

Questions?