Online Technical Interview Bootcamp at Stanford

Session 2

Yongwhan Lim

Sunday, April 16, 2023

Yongwhan Lim

MIT · Columbia University · icpc .foundation · Stanford

Education

Stanford University

MIT

Part-time Jobs

Cornell Tech Jacobs Institute

MIT EECS

Columbia University — In lumine tuo videbimus lumen

Full-time Job

Google Research

2σ TWO SIGMA

Workshops

Stanford ENGINEERING | Stanford Computer Forum

Carnegie Mellon University

Harvard — VE RI TAS

University of California

Seoul National University — VERI LUX TAS MEA

KAIST 1971

NUS National University of Singapore

Coach/Judge

icpc International Collegiate Programming Contest

icpc.foundation

advancing the art and sport of competitive programming

https://www.yongwhan.io

Yongwhan Lim

- Currently:
  - **CEO** (Co-Founder) in a Stealth Mode Startup;
  - **Co-Founder** in Christian and Grace Consulting;
  - ICPC **Internship Manager**;
  - ICPC **North America Leadership** Team;
  - Columbia ICPC **Head Coach**;
  - ICPC **Judge** for NAQ and Regionals;
  - **Lecturer** at MIT;
  - **Adjunct** (Associate in CS) at Columbia;

https://www.yongwhan.io

Session 2: Overview

- **Part I:**
  - Topic 1: String
- **Part II: Problem Walkthroughs**
  - LeetCode Weekly 341
  - LeetCode Biweekly 102
  - AtCoder Beginner Contest 298
  - CodeForces Round #866 (Div. 2)
- **Important Reminders**

Pre-bootcamp Survey

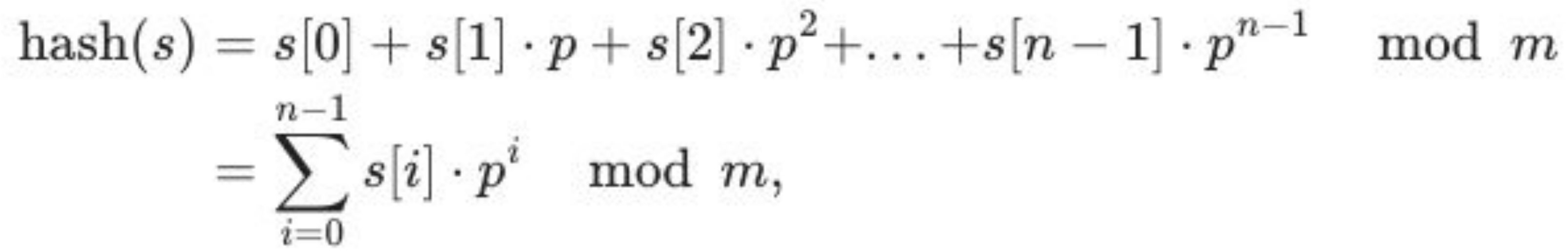
- Please complete the following Google form, as a pre-workshop survey:

https://forms.gle/u2wDzhBt3wxBk83J9

Topic 1

String

I. String Hashing: Main Idea



$$\begin{aligned} \text{hash}(s) &= s[0] + s[1] \cdot p + s[2] \cdot p^2 + \ldots + s[n-1] \cdot p^{n-1} \mod m \\ &= \sum_{i=0}^{n-1} s[i] \cdot p^i \mod m, \end{aligned}$$

I. String Hashing: Implementation

```
long long compute_hash(string const& s) {
    const int p = 31;
    const int m = 1e9 + 9;
    long long hash_value = 0;
    long long p_pow = 1;
    for (char c : s) {
        hash_value = (hash_value + (c-'a'+1)*p_pow)%m;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}
```

Example Problem

- Given a list of `n` strings s_i, each no longer than `m` characters, find all the duplicate strings and divide them into groups.

Solution

```
vector<vector<int>>
group_identical_strings(vector<string> const& s) {
    int n = s.size();
    vector<pair<long long, int>> hashes(n);
    for (int i = 0; i < n; i++)
        hashes[i] = {compute_hash(s[i]), i};
    sort(hashes.begin(), hashes.end());
```

Solution

```
    vector<vector<int>> groups;
    for (int i = 0; i < n; i++) {
        if (i == 0 ||
            hashes[i].first != hashes[i-1].first)
            groups.emplace_back();
        groups.back().push_back(hashes[i].second);
    }
    return groups;
}
```

Fast hash calculation of substrings of given string

- Given a string `s` and indices `i` and `j`, find the hash of the substring `s[i...j]`.

Solution

$$\text{hash}(s[i \dots j]) = \sum_{k=i}^{j} s[k] \cdot p^{k-i} \mod m$$

$$\begin{aligned} \text{hash}(s[i \dots j]) \cdot p^i &= \sum_{k=i}^{j} s[k] \cdot p^{k} \mod m \\ &= \text{hash}(s[0 \dots j]) - \text{hash}(s[0 \dots i-1]) \mod m \end{aligned}$$

Applications

- **Rabin-Karp** algorithm for pattern matching in a string in $O(n)$ time.
- Calculating the *number of different substrings* of a string in $O(n^2 \log n)$
- Calculating the *number of palindromic substrings* in a string.

Determine the number of different substrings in a string

- Given a string s of length n, consisting only of lowercase English letters, find the *number of different substrings* in this string.

Solution

```
int count_unique_substrings(string const& s) {
    int n = s.size();
    const int p = 31;
    const int m = 1e9 + 9;
    vector<long long> p_pow(n);
    p_pow[0] = 1;
    for (int i = 1; i < n; i++)
        p_pow[i] = (p_pow[i-1] * p) % m;
    vector<long long> h(n + 1, 0);
    for (int i = 0; i < n; i++)
        h[i+1] = (h[i] + (s[i]-'a'+1) * p_pow[i]) % m;
```

Solution (con't)

```
    int cnt = 0;
    for (int l = 1; l <= n; l++) {
        set<long long> hs;
        for (int i = 0; i <= n - l; i++) {
            long long cur_h = (h[i + l]+m-h[i]) % m;
            cur_h = (cur_h * p_pow[n-i-1]) % m;
            hs.insert(cur_h);
        }
        cnt += hs.size();
    }
    return cnt;
}
```

II. Rabin-Karp (1987): Problem

- Given two strings - a pattern `s` and a text `t`, determine if the pattern appears in the text and if it does, enumerate all its occurrences in `O(|s| + |t|)` time.

II. Rabin-Karp (1987): Main Idea

- Calculate the hash for the pattern s.
- Calculate hash values for all the prefixes of the text t.
- Now, we can compare a substring of length $|s|$ with s in constant time using the calculated hashes.
- So, compare each substring of length $|s|$ with the pattern.
- This will take a total of $O(|t|)$ time.
- Hence the final complexity of the algorithm is $O(|t|+|s|)$
    - $O(|s|)$ is required for calculating the hash of the pattern and;
    - $O(|t|)$ for comparing each substring of length $|s|$ with the pattern.

II. Rabin-Karp: Implementation

```cpp
vector<int> rabin_karp(string const& s,
                       string const& t) {
    const int p = 31;
    const int m = 1e9 + 9;
    int S = s.size(), T = t.size();
    vector<long long> p_pow(max(S, T));
    p_pow[0] = 1;
    for (int i = 1; i < (int)p_pow.size(); i++)
        p_pow[i] = (p_pow[i-1] * p) % m;
```

II. Rabin-Karp: Implementation

```
vector<long long> h(T + 1, 0);
for (int i = 0; i < T; i++)
    h[i+1] = (h[i] + (t[i]-'a'+1) * p_pow[i]) % m;
long long h_s = 0;
for (int i = 0; i < S; i++)
    h_s = (h_s + (s[i]-'a'+1) * p_pow[i]) % m;
```

II. Rabin-Karp: Implementation

```
  vector<int> occurences;
  for (int i = 0; i + S - 1 < T; i++) {
    long long cur_h = (h[i+S] + m - h[i]) % m;
    if (cur_h == h_s * p_pow[i] % m)
      occurences.push_back(i);
  }
  return occurences;
}
```

III. Knuth-Morris-Pratt (KMP): Prefix function

- You are given a string s of length n.
- The **prefix function** for this string is defined as an array **π** of length n, where **π**[i] is the length of the longest proper prefix of the substring s[0...i] which is also a suffix of this substring.
- A proper prefix of a string is a prefix that is not equal to the string itself. By definition, **π**[0]=0.

$$\pi[i] = \max_{k=0\ldots i} \{k : s[0\ldots k-1] = s[i-(k-1)\ldots i]\}$$

III. Knuth-Morris-Pratt (KMP): Prefix function Example

- prefix function of string `"abcabcd"` is `[0,0,0,1,2,3,0];`
- prefix function of string `"aabaaab"` is `[0,1,0,1,2,2,3];`

III. Knuth-Morris-Pratt (KMP): Main Idea

- We compute the prefix values `π[i]` in a loop by iterating from `i=1` to `i=n-1` (`π[0]` just gets assigned with `0`).
- To calculate the current value `π[i]` we set the variable `j` denoting the length of the best suffix for `i-1`. Initially `j = π[i-1]`.
- Test if the suffix of length `j+1` is also a prefix by comparing `s[j]` and `s[i]`. If they are equal then we assign `π[i]= j+1`, otherwise we reduce `j` to `π[j-1]` and repeat this step.
- If we have reached the length `j = 0` and still don't have a match, then we assign `π[i] = 0` and go to the next index `i+1`.

III. Knuth-Morris-Pratt (KMP): Implementation

```
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j]) j++;
        pi[i] = j;
    }
    return pi;
}
```

Example Problem

- Given a text `t` and a string `s`, we want to find and display the positions of all occurrences of the string s in the text `t`.

Solution

- We generate the string `s+"#"+t`, where `"#"` is a separator that appears neither in `s` nor in `t`.
- If at some position `i` we have `π[i]=n`, then at the position `i-(n+1)-n+1=i-2n` in the string `t` the string `s` appears.

IV. Z-function: Definition

- Suppose we are given a string `s` of length n. The **Z-function** for this string is an array of length `n` where the `i`-th element is equal to the greatest number of characters starting from the position `i` that coincide with the first characters of `s`.

IV. Z-function: Example

- `"aaaaa" - [0,4,3,2,1]`
- `"aaabaab" - [0,2,1,0,2,1,0]`
- `"abacaba" - [0,0,1,0,3,0,1]`

IV. Z-function: Implementation

```
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r) z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }
    return z;
}
```

Example Problem

- Find all occurrences of the pattern `p` inside the text `t`.

Solution

- To solve this problem, we create a new string `s=p+"$"+t`, that is, we apply string concatenation to `p` and `t` but we also put a separator character `"$"` in the middle.
- Compute the Z-function for `s`. Then, for any `i` in the interval `[0, len(t)-1]`, we will consider the corresponding value `k=z[i+len(p)+1]`.
- If `k` is equal to `len(p)` then we know there is one occurrence of `p` in the `i`-th position of `t`, otherwise there is no occurrence of `p` in the `i`-th position of `t`.

Example Problem

- Find all occurrences of the pattern `p` inside the text `t`.

Solution

- To solve this problem, we create a new string `s=p+"$"+t`, that is, we apply string concatenation to p and `t` but we also put a separator character `"$"` in the middle.
- Compute the Z-function for `s`. Then, for any `i` in the interval `[0, len(t)-1]`, we will consider the corresponding value `k=z[i+len(p)+1]`.
- If k is equal to `len(p)` then we know there is one occurrence of p in the `i`-th position of `t`, otherwise there is no occurrence of p in the `i`-th position of `t`.

BREAK

Problem Walkthroughs

- LeetCode Weekly 341
- LeetCode Biweekly 102
- AtCoder Beginner Contest 298
- CodeForces Round #866 (Div. 2)

Request 1:1 Meeting, through Calendly

- Use calendly.com/yongwhan/one-on-one to request 1:1 meeting:
    - Mock Interview
    - Career Planning
    - Resume Critique
    - Practice Strategy
    - Volunteering Opportunity
    - …
- I am always **inspired** by driven students like yourself!
- Since I'd feel honored/thrilled to talk to you, do not feel shy to sign up!!!

Terse Guide Google Drive

- Browse through Terse Guides, which include:
  - Behavioral interview preparation
  - System design interview preparation
  - ICPC preparation
  - Live contests
  - Useful resources

Discord Server Invitations

- Some discord server invitations:
    - **[Online Technical Interview Bootcamp at Stanford]** https://discord.gg/aJwHBccg3n
    - **[ICPC CodeForces Zealots]** https://discord.gg/QC9ss6WJPy

Where to go from here? (for training)

- Train, train, train, BUT only go so much to **NOT** burnout. **IT IS REAL!**
- Each and every one of you can do it, from what I observed last few days!!
- Register for **Universal Cup**: ask, if interested!
- **CSES**: https://cses.fi/problemset/
- **Kattis**: https://open.kattis.com/ with its companion: https://cpbook.net/methodstosolve?oj=kattis&topic=all&quality=all
- **USACO Guide:** https://usaco.guide/ (especially Platinum and Advanced)
- **CP Algorithm:** https://cp-algorithms.com/
  - String Processing; Graphs; Linear Algebra; Data Structures; ...

Contact Information

- **Email**: yongwhan@yongwhan.io

- **Personal Website**: https://www.yongwhan.io/

- **LinkedIn Profile**: https://www.linkedin.com/in/yongwhan/
  - Feel free to send me a connection request!
  - Always happy to make connections with promising students!

Q&A’s