Data Structures

Hashing Homework 2

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Tip

- Whenever doable, use STL unordered set & map

Problem #1: Number of Distinct Substring

- Given a string, count how many unique substring inside it
  - int count_unique_substrings(const string &str)
- For string aaab, substrings are:
  - a, aa, aaa, aaab, a, aa, aab, a, ab, b
  - Only 7 distinct substrings
- Input ⇒ Output
  - aaaaa ⇒ 5
  - aaaba ⇒ 11
  - abcdef ⇒ 21
- Find hash-based solution. What is your best complexity?
- Do you think we can use another data-structure for a more efficient solution?

Problem #2: Common substrings

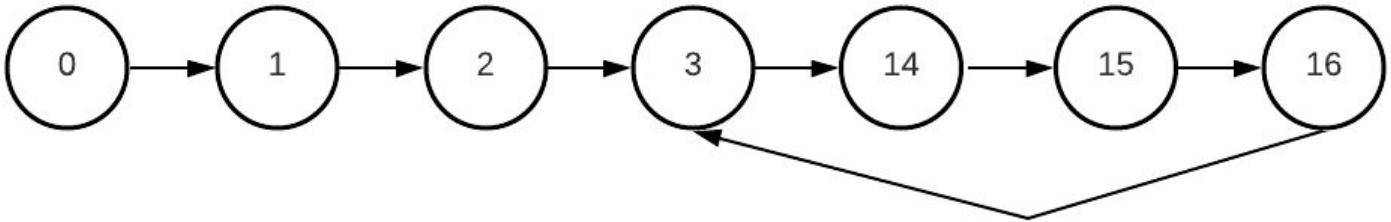
- int count_substrings_match(const string &str1, const string &str2)
  - Assume we have S1 for the unique substrings in str1
  - Assume we have S2 for the unique substrings in str2
  - How many common substrings between S1 and S2?
- Input ⇒ Output
  - aaab, aa ⇒ 2 [a, aa]
  - aaab, ab ⇒ 3 [a, b, aa]
  - aaaaa, xy = 0
  - aaaaa, aaaaa ⇒ 5

Problem #3: Unique Anagrams

- An anagram of a string of lower letters is another string that contains the **same** characters, only the **order** of characters can be different
  - AAB and BAA are anagrams. BBCDE and EDCBB are anagrams
  - BBCDE and EDCB are NOT anagrams (missing B)
- Given a string, find the number of unique anagrams among its substrings
  - int count_anagram_substrings(const string &str)
  - abba has 10 substrings: a, ab, abb, abba, b, bb, bba, b, ba, a
  - Only 6 are unique anagrams. **Duplicate groups** are (ab, ba), (a, a), (b, b), (abb, bba)
  - Find $O(L^3 \log L)$ or better $O(L^3)$
- Input ⇒ Output
  - aaaaa ⇒ 5, abcba ⇒ 9, aabade ⇒ 17

Problem #4: Cycle in linked-list

- Given a singly linked list where last node points to some node in the middle not null as expected. We need to correct the list by making the last edge actually points to null
- Develop a simple hash-based solution



Problem #4: Cycle in linked

- Use the linked list code we developed before
- Add this create cycle utility
- Implement remove_cycle function

```
LinkedList lst;
lst.create_cycle();
lst.remove_cycle();
lst.print();
// 0 1 2 3 14 15 16
```

```
class LinkedList {
private:
    Node *head { };
    Node *tail { };
    int length = 0;
public:
    void print() {..
    void insert_end(int value) {..
    void create_cycle() {
        for (int i = 0; i < 4; ++i)
            insert_end(i);
        Node* now = tail;
        for (int i = 0; i < 3; ++i)
            insert_end(14 + i);
        tail->next = now;// cycle
    }
    void remove_cycle() {..
};
```

Problem #5: Quadratic Probing

- Change the lecture code to quadratic probing
- void put(PhoneEntry phone)
  - The function this time must guarantee element is added
  - Important question: When to know we can’t add the element using current sequence? In other words, when to stop moving to the next some_idx + i * i
- Add rehashing function
  - void rehashing()
  - Assume no load_factor is requested. If called, just do rehash

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."