Suffix Array

Chenqi Wang

2021.12.23

1 Definitions

string s: a sequence of symbols from a given alphabet. denoted as $s[0, ..., n-1]$. e.g. "aabaaaab"

prefix: $s[0, ..., i]$ $(0 \leq i \leq n-1)$. e.g. "aaba"

suffix: $s[i, ..., n-1]$ $(0 \leq i \leq n-1)$. e.g. "baaaab"

suffix array : there are n suffixes in string $s[0, ..., n-1]$, i.e. $s[i, ..., n-1]$ $(0 \leq i \leq n-1)$, and each suffix can be uniquely identified using i (the position of its first character).

If we sort these suffixes lexicographically, then we get the suffix array sa[0, ..., n - 1], where sa[i] is the i-th smallest suffix, represented as the position of its first character.

And we define the rank array rk[sa[i]] = i, i.e. rk is the inverse of sa, and vice versa.

e.g. the sorted suffixes of "aabaaaab" are

- $sa[0] = 3$: aaaab $rk[3] = 0$
- $sa[1] = 4$: aaab $rk[4] = 1$
- $sa[2] = 5$: aab $rk[5] = 2$
- $sa[3] = 0$: aabaaaab $rk[0] = 3$
- $sa[4] = 6$: ab $rk[6] = 4$
- $sa[5] = 1$: abaaaab $rk[1] = 5$

- $sa[6] = 7$: b $rk[7] = 6$
- $sa[7] = 2$: baaaab $rk[2] = 7$

the height array ht: $ht[i]$ = the length of longest common prefix of suffixes $sa[i]$ and $sa[i-1]$ $1 \leq i \leq n-1$. ht[0] := 0. In the above example, ht[0] = 0, ht[1] = 3 aaa, ht[2] = 2 aa, ht[3] = 3 aab, ht[4] = 1 a, ht[5] = 2 ab, ht[6] = 0, ht[7] = 1 b.

2 sa, rk, ht

How to efficiently calculate the suffix array sa, the rank array rk and the height array ht for string $s[0, ..., n-1]$?

Define $s(i, 2l) := s[i, ..., i + 2l - 1]$ (if $i + 2l - 1 \geq n$, simply leave out the part $\geq n$). We sort the suffixes by first sort $s[i](i = 0, ..., n-1)$, then $s(i, 2)(i = 0, ..., n-1)$, $s(i, 4)(i = 0, ..., n-1)$ ..., $s(i, n)(i = 0, ..., n-1)$.

Base case: sort $s[i](i = 0, ..., n-1)$. Simply do a counting sort.

Assume we already sorted $s(i, l)(i = 0, ..., n-1)$, and got the sa and rk for them, we can then sort $s(i, 2l)(i = 0, ..., n-1)$ using the sa and rk for $s(i, l)$: $s(i, 2l)$ corresponds to two keys $rk[i]$ and $rk[i + l]$, and $s(i, 2l) \leq s(j, 2l)$ iff $rk[i] < rk[j]$ or ($rk[i] == rk[j]$ and $rk[i+l] \leq rk[j+l]$). So we can do a radix sort based on those 2 keys, just like the radix sort for integers: we first do a counting sort by the second key, then do a (stable) counting sort by the first key. In this way, we calculate the sa for $s(i, 2l)(i = 0, ..., n-1)$.

During radix sort, for the second key, we can get the result directly from sa for $s(i, l)$: for $s(i, 2l)(i + l \geq n)$, they are the smallest (because empty string $\leq$ any string); for $s(i, 2l)(i + l < n)$, the order remains the same as the order of $sa[j](sa[j] \geq l)$, e.g. suppose $sa[j], sa[k] \geq l(j < k)$, then $s(sa[j] - l, 2l)$ should be before $s(sa[k] - l, 2l)$ in the sorted result by the second key.

For the radix sort for the first key, we make sure $s(i, 2l)$ are sorted by the second key at first. And the counting sort for the first key is stable, i.e. if the first keys are equal for $s(i, 2l)$ and $s(j, 2l)$, they are sorted by the second key.

Given sa, we calculate rk: $rk[sa[0]] = 0$; for $i > 0$, if $s(sa[i], 2l) == s(sa[i-1], 2l)$, $rk[sa[i]] = rk[sa[i-1]]+1$, otherwise $rk[sa[i]] = rk[sa[i-1]]$. If all the $rk[i](i = 0, ..., n-1)$ are distinct, we already finish sorting the suffixes.

After we got sa and rk, we calculate $ht[rk[i]](i = 0, ..., n-1)$ by brute force: each time count the length of longest common prefix between $s(i, n)$ and $s(sa[rk[i]-1]], n)$. Using the property that $ht[rk[i]] >= ht[rk[i-1]]-1$, we can start the counting at $ht[rk[i-1]] - 1$, so the time complexity is optimized to $O(n)$.

Time Complexity: $O(nlogn)$. we calculate sa and rk for $O(logn)$ groups of $s(i, 2l)(i = 0, ..., n-1)$ $(l = 1, 2, 4, ..., n)$, each group $O(n)$, so $O(nlogn)$, which dominates the others. (calculate ht is $O(n)$)

Space Complexity: $O(n)$.

3 The Number of Distinct Substrings

https://www.luogu.com.cn/problem/P2408

(suffix array template: https://www.luogu.com.cn/problem/P3809)

Given a string s of length n, after calculating the sa, rk, ht for it, we can easily calculate the number of distinct non-empty substrings of s as $\frac{n(n+1)}{2} - \sum_{i=0}^{n-1} ht[i]$.

proof. Each substring can be viewed as a prefix of a suffix of s. For $sa[0]$, the number of distinct substrings is $n - sa[0]$ (i.e. the length of suffix $s(sa[0], n)$). And for $sa[i]$ $i > 0$, the increased number of distinct substrings is $n - sa[i] - ht[i]$ ($ht[i] \leq n - sa[i]$, because ht[i] is the length of a prefix of $s(sa[i], n)$, $n - sa[i]$ is the length of $s(sa[i], n)$), the common prefix between $s(sa[i], n)$ and $s(sa[i-1], n)$ have already been counted, so we only count the prefix of $s(sa[i], n)$ that is not a prefix of $s(sa[i-1], n)$, such prefixes of $s(sa[i], n)$ cannot be a prefix of $s(sa[j], n)(j < i-1)$, because otherwise $s(sa[i], n)$ should be sorted adjacently to $s(sa[j], n)$ instead of $s(sa[i-1], n)$. So the total number of distinct substrings are $\sum_{i=0}^{n-1} n - sa[i] - ht[i] = \sum_{i=0}^{n-1}(n - sa[i]) - \sum_{i=0}^{n-1} ht[i] = \sum_{i=1}^{n} i - \sum_{i=0}^{n-1} ht[i] = \frac{n(n+1)}{2} - \sum_{i=0}^{n-1} ht[i]$

$T(n) = O(nlogn), S(n) = O(n)$

```cpp
#include <iostream>
#include <vector>
#include <string>
#define ll long long
using namespace std;

class SuffixArray{
public:
    vector<int> sa;
    vector<int> rk;
    vector<int> ht;

    // s[sa[i], ..., sa[i] + 2l - 1] == s[sa[j], ..., sa[j] + 2l - 1]
    inline bool equal(int i, int j, int l, int n) {
        if (rk[sa[i]] != rk[sa[j]]) return false;
        else {
            if (sa[i] + l >= n && sa[j] + l >= n) return true;
            else if (sa[i] + l >= n && sa[j] + l < n) return false;
            else if (sa[i] + l < n && sa[j] + l >= n) return false;
            else return rk[sa[i] + l] == rk[sa[j] + l];
        }
    }

    SuffixArray(const string& s) {
        int n = s.size();
        sa = vector<int>(n);
        rk = vector<int>(n);
        ht = vector<int>(n);

        int m = max(123, n);

        vector<int> f(m);

        for (int i = 0; i < n; ++i) f[s[i]]++;
        for (int i = 1; i < m; ++i) f[i] += f[i - 1];
        for (int i = n - 1; i >= 0; --i) sa[--f[s[i]]] = i;
        int p = 0;
        rk[sa[0]] = 0;
        for (int i = 1; i < n; ++i) {
            if (s[sa[i]] != s[sa[i - 1]]) p++;
            rk[sa[i]] = p;
        }

        vector<int> y(n);
        vector<int> w(n);

```

```
47          for (int l = 1; p < n - 1; l <<= 1) {
48              p = 0;
49              for (int i = n - l; i < n; ++i) y[p++] = i;
50              for (int i = 0; i < n; ++i) if (sa[i] >= l) y[p++] = sa[i] -
                    l;
51              for (int i = 0; i < n; ++i) w[i] = rk[y[i]];
52              for (int i = 0; i < n; ++i) f[i] = 0;
53              for (int i = 0; i < n; ++i) f[w[i]]++;
54              for (int i = 1; i < m; ++i) f[i] += f[i - 1];
55              for (int i = n - 1; i >= 0; --i) sa[--f[w[i]]] = y[i];
56              p = 0;
57              vector<int> tmp(n);
58              tmp[sa[0]] = 0;
59              for (int i = 1; i < n; ++i) {
60                  if (!equal(i, i - 1, l, n)) p++;
61                  tmp[sa[i]] = p;
62              }
63              rk = tmp;
64          }
65
66          p = 0;
67          for (int i = 0; i < n; ++i) {
68              if (p) p--; // ht[rk[i]] >= ht[rk[i - 1]] - 1
69              if (rk[i] == 0) continue;
70              while (i + p < n && sa[rk[i] - 1] + p < n && s[i + p] == s[sa
                    [rk[i] - 1] + p]) p++;
71              ht[rk[i]] = p;
72          }
73
74      }
75
76      inline ll numberOfDifferentSubstrings() {
77          ll n = sa.size();
78          ll sum = 0;
79          for (int i = 1; i < n; ++i) sum += ht[i];
80          return n * (n + 1) / 2 - sum;
81      }
82  };
83
84
85  int main() {
86      int n;
87      cin >> n;
88      string s;
89      cin >> s;
90      SuffixArray SA(s);
```

```
91      cout << SA.numberOfDifferentSubstrings() << endl;
92      return 0;
93  }
```

source: https://oi-wiki.org/string/sa/