Suffix Arrays

Suffix Array Nedir?

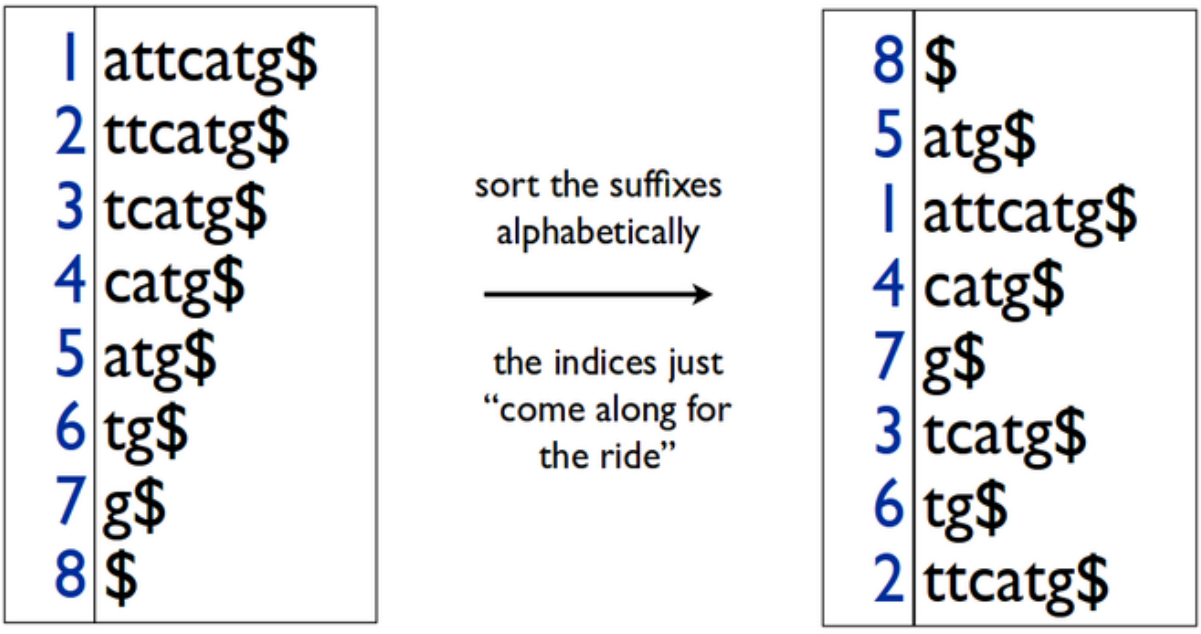
Basitçe ifade etmek gerekirse Suffix Array, verilen bir stringin tüm suffixlerinin yani soneklerinin sıralanmış halidir.

Bir veri yapısı olarak Suffix Array, veri sıkıştırma, biyoinformatik, daha genel söylemek gerekirse string ve string matching problemlerinde kullanılabilir ve bundan dolayı bilinmesi büyük bir önem arz eder.

Suffix Array, verilen stringin suffixlerinin sıralandıktan sonraki hallerinin stringteki başlangıç indislerini tutar.

Suffix Array'in bazı uygulamalarında stringin sonuna alfabede geçmeyen ve değeri daha düşük bazı özel karakterler ('$', '#', '@' vb.) konur.

Sağ tarafta asıl hali "attcatg" olan stringin suffixleri sort edildikten sonraki halini görmektesiniz.



Yukarıdaki resim ne yapmak istediğimizi anlatıyor ama asıl amacımız bunu yapmanın farklı yollarını keşfetmek.

Bazı tanıdık algoritmaları her ne kadar bazıları yavaş da olsa Suffix Array'i kodlarken kullanacağız.

Naive Algorithm

Problemimizi çözmeye saf algoritmayla başlayacağız ki bu kodlanması en kolay olanıdır.

Tüm suffixleri sıralamak için saf algoritmamızın anahtarı iyi bir karşılaştırma tabanlı sıralama algoritması kullanılmasıdır. Quick-Sort bunu iyi bir şekilde halleder.

Bize verilen stringin tüm suffixlerini bir Vector'e atacağız ve yine bu suffixlere karşılık gelen başlangıç indislerini de bir Map'te tutacağız. Vector'deki stringleri sıraladıktan sonra Map'te o suffixlere karşılık olan başlangıç indislerini yazdırabiliriz.

Normalde sort fonksiyonumuz O(N log N) iken işin içine string kattığımız ve compare ederken bize O(N) gibi maliyete mal olduğundan genel olarak time complexity $O(N^2 \log N)$ olacaktır ki bu da verimli bir çalışma maliyeti değil. Ayrıca memory complexity ise O(N^2).

```
string s;
map<string,int> m;
cin >> s;
vector<string> v;
for(int i = 0; i < s.size();i++)
{
    m[s.substr(i,s.size()-i)] = i;
    v.push_back(s.substr(i,s.size()-i));
}
sort(v.begin(),v.end());
for(int i = 0; i < v.size();i++)
{
    cout << m[v[i]] << endl;
}
```

Suffix Array'e daha etkili bir yaklaşım

Olaya biraz daha zekice yaklaşalım. İşimize verilen stringin sadece tek karakter içeren substringlerini sort etmek ile başlayalım ki hepsinin sıra indisi olacak ve bu sıralandığında kaçıncı sıraya karşılık geldiğini tutacak. Sonraki adımda 2 uzunluğunda olan substringleri önceki verileri kullanarak sıralayacağız. Bir sonraki adımımızda en son verileri kullanarak 4 uzunluğunda olan substringleri sıralayacağız. Bu işlemler biz tüm stringlerin sıralı olduğuna emin oluncaya kadar devam edecek. Ayrıca sıralı substringlerden aynı olanlara aynı sıra indisini vermeye dikkat etmemiz lazım.

Böylece algoritmamız $O(N \log^2 N)$ gibi güzel bir zamanda çalışabilecek.

“mississippi” kelimesini inceleyelim…

Öncesinde tüm 2 harfli substringleri sıralayalım.

Sort-Index Suffix-Index

```
  0      10: i
  1       7: ippi
  2       1: ississippi
  2       4: issippi
  3       0: mississippi
  4       9: pi
  5       8: ppi
  6       3: sissippi
  6       6: sippi
  7       2: ssissippi
  7       5: ssippi
```

Sonraki adımızda 4 uzunluğunda olanları sıralamamız gerekecek ki bunun için eski verileri kullanmamız lazım. Yani [i,i+3] aralığındaki substringin sırasını bulabilmemiz için [i,i+1] ve [i+2,i+3] aralıklarının indislerini pair halinde tutmamız yeterlidir.

```
  0      10: i             -1   (0, -1)
  1       7: ippi           9   (1, 4)
  2       1: ississippi     3   (2, 6)
  2       4: issippi        6   (2, 6)
  3       0: mississippi    2   (3, 7)
  4       9: pi            -1   (4, -1)
  5       8: ppi           10   (5, 0)
  6       3: sissippi       5   (6, 7)
  6       6: sippi          8   (6, 5)
  7       2: ssissippi      4   (7, 2)
  7       5: ssippi         7   (7, 1)
```

Bu işlemleri ardarda uygulayarak amacımıza ulaşabiliriz. Aşağıda $O(N \log^2 N)$ maliyetli C++ kodu görmektesiniz.

```
#define MAXN 65536
#define MAXLG 17

char A[MAXN];

struct entry
{
   int nr[2];
   int p;
} L[MAXN];

int P[MAXLG][MAXN];
int N,i;
int stp, cnt;

int cmp(struct entry a, struct entry b)
{
   return a.nr[0]==b.nr[0] ?(a.nr[1]<b.nr[1] ?1: 0): (a.nr[0]<b.nr[0] ?1: 0);
}

int main()
{
   gets(A);
   for(N=strlen(A), i = 0; i < N; i++)
      P[0][i] = A[i] - 'a';

   for(stp=1, cnt = 1; cnt < N; stp++, cnt *= 2)
   {
      for(i=0; i < N; i++)
      {
         L[i].nr[0]=P[stp- 1][i];
         L[i].nr[1]=i +cnt <N? P[stp -1][i+ cnt]:-1;
         L[i].p= i;
      }
      sort(L, L+N, cmp);
      for(i=0; i < N; i++)
         P[stp][L[i].p] =i> 0 && L[i].nr[0]==L[i-1].nr[0] && L[i].nr[1] == L[i- 1].nr[1] ? P[stp][L[i-1].p] : i;
   }
   return 0;
}
```

$O(N \log^2 N)$ maliyetli pseudo-code...

```
SortIndex[][] = { 0 }

for i = 0 to N-1
   SortIndex[0][i] = order index of the character at A[i]

doneTill = 1
step = 1

while doneTill < N
   L[] = { (0,0,0) } // Array of 3 tuples

   for i = 0 to N-1
      L[i] = ( SortIndex[step - 1][i],
               SortIndex[step - 1][i + doneTill],
               i
             )
   // We need to store the value of i to be able to retrieve the index

   sort L

   for i = 0 to N-1
      SortIndex[step][L[i].thirdValue] =
       SortIndex[step][L[i-1].thirdValue], if L[i] and L[i-1] have the same first and second
values
          i, otherwise

   ++step
   doneTill *= 2
```

LCP Array'inin oluşturulması ve açıklaması

Longest Common Prefix, Suffix Array'e yardımcı bir veri yapısıdır. Ardışık suffixlerin LCP'lerinin uzunluklarını tutmasını sağlar.

Suffix Array yapısını kurmuş biri için LCP'yi görmek nazaran daha kolaydır.

Yukarıda kullandığımız SortIndex dizisini 2 ardışık prefixin LCP'sini bulurken kullanabiliriz.

```
FindLCP (x, y)
   answer = 0

   for k = ceil(log N) to 0
      if SortIndex[k][x] = SortIndex[k][y]
         // sort-index is same if the first k characters are same
         answer += 2^k
         // now we wish to find the characters that are same in the remaining strings
         x += 2^k
         y += 2^k
```

LCP dizisi i.suffix ve (i-1).suffixin LCP'lerini tutan dizidir. Üstteki algoritma LCP Array'ini tutmak için N kere çağrılmalıdır ve sonuç olarak O(N log N) zaman alacaktır.

Ek

UVa Soru: http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=660

CodeChef Sorular: http://www.codechef.com/tags/problems/suffix-array

CodeForces Sorular: http://codeforces.com/problemset/tags/string%20suffix%20structures

CodeForces Öneri: http://codeforces.com/problemset/problem/19/C

Yardımcı Kaynak: http://www.cs.umd.edu/class/fall2011/cmsc858s/SuffixArrays.pdf