Laboratorio di Motori di Ricerca

Il Pensiero Computazionale

Percorso Formativo per i Docenti della Scuola Secondaria di II Grado

Lezione 4 (Parte pratica)

Prof. Paolo Ferragina and Davide Neri

Cosa faremo oggi

- Costruzione SA (Suffix Array)
- Ricerca di sottostringhe (Ricerca binaria in SA)
- Costruzione LCP-array (Longest-Common-Prefix)

Costruzione SA (pseudo-codice)

Algorithm 10.3 COMPARISON_BASED_CONSTRUCTION(char *T, int n, char **SA)

```
1: for (i = 0; i < n; i ++) do
2:         SA[i] = T + i;
3: end for
4: QSORT(SA, n, sizeof(char *), Suffix_cmp);
```

`Suffix_cmp()` è la funzione in C-style usata per comparare due suffissi nella funzione `QSORT()`

```
Suffix_cmp(char **p, char **q){ return strcmp(*p, *q) };
```

Costruzione SA (codice v1)

```
def naiveBuildSA(text):
    # list of tuple (suffix,  index)
    satups = sorted([(text[i:], i) for i in range(len(text))])
    return list(map(lambda x: x[1], satups))
```

Complessità :

- Spazio: $O(n^2)$
- Tempo: $O(n^2 \ log \ n)$

Costruzione SA (codice v2)

la funzione `suffixCmp()` definisce la funzione per ordinare i suffissi in `sorted()`.

```
def buildSA(text):
    # inizializza il SA con gli indici da [0, n-1]
    SA = [x for x in range(len(text))]
    # funzione di comparazione
    def suffixCmp(suffixPos):
        return text[suffixPos:]
    return sorted(SA, key=suffixCmp)
```

Complessità:

- Spazio: $O(n)$
- Tempo: $O(n^2 \ log \ n)$

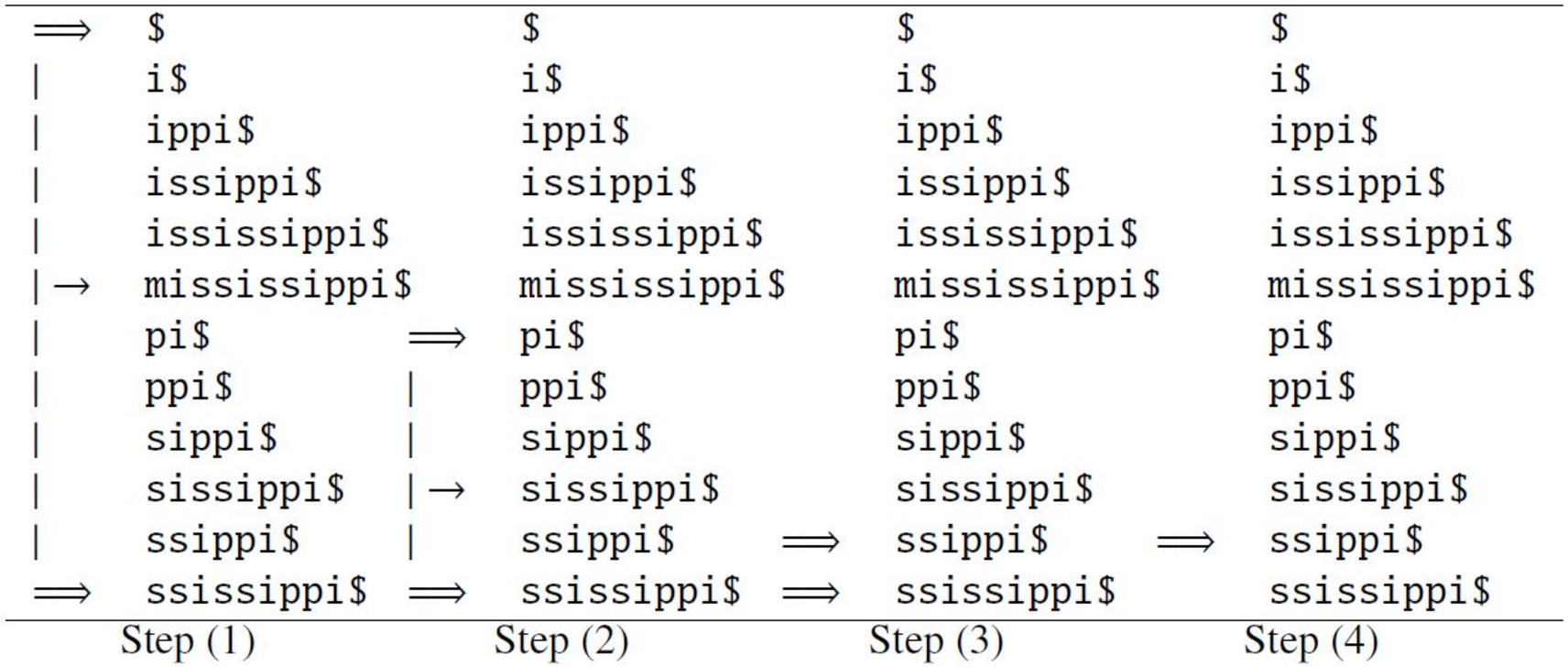
Costruzione SA (esempio)

```
T = 'mississippi$' # input("Inserisci testo")

print(naiveBuildSA(T))
print(buildSA(T))
```

Ricerca di sottostringhe

Esempio di ricerca binaria della posizione in SA del pattern `P = ssi` all'interno della stringa `mississippi$`.



Ricerca di sottostringhe con SA (pseudo-codice)

Algorithm 10.1 SUBSTRINGSEARCH(P, $SA(T)$)

```
1:  L = 1, R = n;
2:  while (L ≠ R) do
3:      M = ⌊(L + R)/2⌋;
4:      if (strncmp(P, suff_M, p) > 0) then
5:          L = M + 1;
6:      else
7:          R = M;
8:      end if
9:  end while
10: if (strncmp(P, suff_L, p) = 0) then
11:     return L;
12: else
13:     return −1;
14: end if
```

Ricerca di sottostringhe con SA (codice)

```
def substringSearch(t, sa, p):
    assert t[-1] == '$'       # se t ha il terminatore
    assert len(t) == len(sa) # sa è il sa di t
    if len(t) == 1: return 1 #

    n = len(sa)
    l, r = 0, n-1

    while l != r:
        m = int(math.floor((l + r) / 2))
        suffixM = t[sa[m]:]
        if p > suffixM:
            l = m + 1
        else:
            r = m
    suffixL = t[sa[l]:]
    if suffixL.startswith(p):
        return l
    else:
        return None
```

Ricerca di sottostringhe con SA (esempio)

```
T = 'mississippi$'
SA = buildSA(T)

P = "ssi"
isa = substringSearch(T, SA, P)
if isa is None:
    print("Substring ", P, " not found")
else:
    print("Substring ", P, " found in" , T[SA[isa]:])

P = "ti"
isa = substringSearch(T, SA, P)
if isa is None:
    print("Substring ", P, " not found")
else:
    print("Substring ", P, " found in" , T[SA[isa]:])
```

Costruzione LCP (codice)

```
# Calcola il lcp tra due stringhe
def lcp(x, y):
    for i in range(min(len(x), len(y))):
        if x[i] != y[i]: return i
    return min(len(x), len(y))

def naiveBuildLCP(t, sa):
    lcparray= []
    for i in range(len(sa)-1):
        nlcp = lcp(t[sa[i]:], t[sa[i+1]:])
        lcparray.append(nlcp)
    return lcparray
```

Costruzione LCP (esempio)

```
T = 'mississippi$'
SA = buildSA(T)

print(naiveBuildLCP(T, SA))
```