Review Merge Sort

Divide: 
\[ \frac{n}{2} \]

Sort:
\[ \text{sort} \]

Merge:
\[ \text{merge} \]

\[ T(n) = 2 \cdot T(\frac{n}{2}) + n \]

\[ T(n) = \Theta(n \log n) \]
Skew Algorithm For Suffix Sorting

- Let $S_0, S_1, S_2, \ldots, S_{n-1}$ be all the $n$ suffixes. $S_i$ starts at $i$-th position.

- Skew algorithm uses divide and conquer. But it divides the problem into unequally sized parts.

- Two sets $SA^0 = \{S_i: i = 0 \mod 3\}$ and $SA^{12} = \{S_i: i=1 \text{ or } 2 \mod 3\}$. 
Skew Algorithm Example

- Example: mississippi

\[
\begin{align*}
\text{SA}^0 & \quad \text{SA}^f & \quad \text{SA}^2 \\
\text{SA}^{12} & \quad \text{SA}^2 \\
0: & \text{ mississippi} & 1: & \text{ississippi} & 2: & \text{ssissippi} \\
3: & \text{ sissippi} & 4: & \text{issippi} & 5: & \text{ssippi} \\
6: & \text{ sippi} & 7: & \text{ippi} & 8: & \text{ppi} \\
9: & \text{ pi} & 10: & \text{i} & & \\
\end{align*}
\]

1. Sort \text{SA}^{12} recursively
2. Sort \text{SA}^0 in linear time
3. Merge in linear time

\[
\begin{align*}
T(n) &= T\left(\frac{2}{3} n\right) + n + n \\
&= O(n).
\end{align*}
\]
Skew Algorithm Example

• Example: mississippi

SA^0

mississippi
sissippi
sippi
pi

SA^{12}

ississippi
issippi
ippi

length \sim \frac{2n}{3}

3-mer

Renaming:
3-mer \rightarrow integer

Now we can do recursion, except 3-mer.
Skew Algorithm Example

- Example: *mississippi*

\[
\begin{align*}
S_{A0} & \quad S_{A1}^2 \\
0: & \text{mississippi} & 2: & \text{ssissippi} \\
3: & \text{sissippi} & 5: & \text{ssippi} \\
6: & \text{sippi} & 8: & \text{ppi} \\
9: & \text{pi} & 10: & \text{i}
\end{align*}
\]

One-pass in radix sorting

\[m \leq s < p\]

\[4, 7, 10\]
Skew Algorithm Example

• Example: mississippi

<table>
<thead>
<tr>
<th>SA^0</th>
<th>SA^1</th>
<th>SA^2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: mississippi</td>
<td>ississippi</td>
<td>ssissippi</td>
</tr>
<tr>
<td>3: sissippi</td>
<td>issippi</td>
<td>ssippi</td>
</tr>
<tr>
<td>6: sippi</td>
<td>ippi</td>
<td>ppi</td>
</tr>
<tr>
<td>9: pi</td>
<td>i</td>
<td></td>
</tr>
</tbody>
</table>

\[ A \rightarrow i \rightarrow \leftarrow 5 \]

<table>
<thead>
<tr>
<th>a ∈ SA^0 \text{ v.c. } be ∈ SA^1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a ∈ SA^0 \rightarrow b ∈ SA^1</td>
</tr>
<tr>
<td>a ∈ [0, 3) \rightarrow b ∈ SA^2</td>
</tr>
<tr>
<td>b ∈ [0, 3) \rightarrow a ∈ SA^1</td>
</tr>
</tbody>
</table>

\[ a \rightarrow b \leftrightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow h \rightarrow i \rightarrow j \rightarrow k \rightarrow l \rightarrow m \rightarrow n \rightarrow o \rightarrow p \rightarrow q \rightarrow r \rightarrow s \rightarrow t \rightarrow u \rightarrow v \rightarrow w \rightarrow x \rightarrow y \rightarrow z \rightarrow A \]
Skew Algorithm For Suffix Sorting

- 1. Sort $SA^{12}$ recursively.
- 2. Sort $SA^0$ in linear time.
- 3. Merge sort $SA^0$ and $SA^{12}$ in linear time.
How to sort $SA^{12}$ recursively

mississippi

$SA^{12}$

mod 1

ississippi
ississippi
ippi
i

mod 2

ssissippi
ssiissippi
ippi

- We need to know the order of these suffixes.
- In order to solve it recursively, we need to reduce the problem to a smaller suffix sorting problem.
Reduction to a smaller suffix sorting problem

\[
\text{SA}^{12} \quad \text{mod } 1 \\
\text{iSSiSSiPpi00} \quad \text{iSSiPpi00} \\
\text{iPPi00} \quad \text{i00}
\]

\[
\text{mod } 2 \\
\text{sSiSiSSippi} \\
\text{SSiSiSSippi} \\
\text{SSiPi} \\
\text{Pi}
\]

- Pad 0 to make their length multiple of 3. Then treat each string as a string of “triplets”. Each subset is the suffixes of the “triplet string”.
- We connect the two “triplet strings” together to make a longer string. We put the one with padding at the left.
Reduction

\[ \text{SA}^{12} \mod 1 \]
- issi
- sissi
- ppi
- i00

\[ \text{mod 2} \]
- ssissi
- sippi
- ppi

- Now check all the suffixes of the concatenated triplet string. Their relative order can be used to build the relative order of \( \text{SA}^{12} \) easily.
- We are almost there, except that keeping tripling the size (number of bytes) of the “character” is a problem.
Renaming

• We solve the unlimited expansion problem by a trick called renaming. It maps each unique triplet to a single unique integer.

• To rename, we first sort the triplets, and then assign integer values sequentially to unique triplets. Sorting triplets can be done in linear time by radix sort.

• This ensures
  • The max value is always bounded by the length of array.
  • The suffix order is unchanged.

\[
i00 \rightarrow 0 \quad \text{iss} \quad \text{issippi} \quad ppi \quad ssi
ipp \rightarrow 1
iss \rightarrow 2
ppi \rightarrow 3
ssi \rightarrow 4
\]

\[
\begin{array}{ccccccc}
i00 & ppi & iss & ssi & i00 & ppi & iss \\
2 & 2 & 1 & 0 & 4 & 4 & 3
\end{array}
\]
Renaming Example

i00 -> 0
ipp -> 1
iss -> 2
ppi -> 3
ssi -> 4

ississippi i00 ssissippi

2 2 1 0 4 4 3
Recursion

- After renaming, we just suffix sort the new integer string, which has length approximately $2n/3$. This can be done by recursion.
- The time complexity of renaming is dominated by sorting the triplets. This can be solved in linear time with radix sort.
Radix Sort

- **Radix Sort**: Multiple passes. Each pass stable sorts according to one digit. From the least to the most significant digit.

  - original: its, iss, ipp, abc, att
  - pass1: abc, ipp, its, iss, att
  - pass2: abc, ipp, iss, its, att
  - pass3: abc, att, ipp, iss, its

- Radix sorting requires $O(k)$ space, where $k$ is the size of the alphabet.
- Each pass takes linear time. And only 3 passes needed in our case. So it is linear time.
Recap Sort $S^{12}$ recursively

1. Padding and concatenation to get string of triplets.
2. Radix sort the triplets to get an ID (name) of each triple.
3. Recursion to get the suffix order on the string of IDs.
We assume $SA^{12}$ is sorted already, and learn the other two steps first.

1. Sort $SA^{12}$ recursively.
2. Sort $SA^0$ in linear time.
3. Merge sort $SA^0$ and $SA^{12}$ in linear time.
Sort $S^0$ in linear time

- $S_i = s[i] S_{i+1}$.
- For all $S_i$ in $SA^0$, $S_{i+1}$ has been sorted already. Use $s[i]$ to do another pass of radix sorting will give us the right order of $SA^0$. This takes linear time.

Sorted $SA_{12}$

<table>
<thead>
<tr>
<th>Sorted SA12</th>
<th>0 1 2 3 4 5 6 7 8 9 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ississippi</td>
<td>mississippi</td>
</tr>
</tbody>
</table>

10: i
4: ississippi
1: ississippi
7: ippi
8: ppi
5: ssippi
2: ssissippi

To sort $SA_0$

<table>
<thead>
<tr>
<th>0: mississippi</th>
</tr>
</thead>
<tbody>
<tr>
<td>3: sissippi</td>
</tr>
<tr>
<td>6: sippi</td>
</tr>
<tr>
<td>9: pi</td>
</tr>
</tbody>
</table>
Skew Algorithm For Suffix Sorting

- 1. Sort $SA^{12}$ recursively.
- 2. Sort $SA^{0}$ in linear time.
- 3. Merge sort $SA^{0}$ and $SA^{12}$ in linear time.
Would be a simple merge if comparison of two takes constant time.

Trouble is when two suffices share a long prefix, which takes more than constant time to compare.

E.g. what if S5 = aaaa… and S6=aaaa…
Merge $S^0$ and $S^{12}$

- Merging only requires to compare a suffix $S_j$ with $j \mod 3 = 0$ with a suffix $S_i$ with $i \mod 3 \neq 0$.

- Case 1: If $i \mod 3 = 1$, we write $S_i$ as $(s[i], S_{i+1})$ and $S_j$ as $(s[j], S_{j+1})$.
  - Since $(i +1) \mod 3 = 2$ and $(j + 1) \mod 3 = 1$, the relative order of $S_{j+1}$ and $S_{i+1}$ can be determined from their position in $SA^{12}$.

- Case 2: If $i \mod 3 = 2$, we compare the triples $(s[i], s[i + 1], S_{i+2})$ and $(s[j], s[j + 1], S_{j+2})$. 
Recap
C codes

• 50 lines of C++ codes were given in J.C.M. Baeten et al. (Eds.): ICALP 2003, LNCS 2719, pp. 943–955, 2003.

• http://www.mpi-inf.mpg.de/~sanders/programs/suffix/