Übungen zur Algorithmischen Bioinformatik I

Blatt 11

Xiheng He

Juli 2021

4. Aufgabe: Algorithmenentwurf mit Keyword- und Suffix-Trees

(a) Word Statistics

Gegeben ist ein String S und ein $n \in N$. Entwerfen Sie einen möglichst effizienten Algorithmus, der alle Teilstrings von S bestimmt, die in S genau n mal vorkommen ($O(|S| + k)$ ist denkbar, mit k ist Anzahl der Lösungs-Teilstrings). Korrektheitsbeweis und Laufzeitanalyse nicht vergessen.

```
Word Statistics (String S, int n)
begin
    char[] t;
    for (i = 0; i < |S|; i++) do
        t[i] := S[i];
    // Build Suffix-Tree first
    BuildSuffixTree(t, |S|);
    Set{} strs;
    // v_i: a internal node
    // {u_i}: all childnodes of v_i
    foreach (v_i : {v|v ∈ T ∧ {u_i} ≠ ∅}) do
        if (|u_i| = n) then
            // strs: all n repeating substrings in S
            strs := strs ∪ v_i;
    return strs;
end
```

Laufzeitanalyse:

- Die Umwandlung in Char-Array braucht $O(|S|)$ Laufzeit.
- Die Laufzeitkomplexität liegt in $O(|S|)$ um ein Suffix-Baum mit Hilfe des Ukkonens Algorithmus zu konstruktieren.
- Foreach Schleife kann insgesamt maximal $O(|S|)$ durchlaufen weil nur innere Knoten besucht werden und die Anzahl der ineren Knoten geringer als die Anzahl der Blätter $O(|S|)$ ist.

Daraus folgt, dass die Laufzeit für Word Statistics Problem in $O(|S|)$ liegt.

Korrektheitanalyse:

Die Grundidee ist: jede Teilfolge in S ist ein Prefix der Suffix in S.

Kommt eine Teilfolge n mal in S vor, muss sie als Prefix in genau n Suffixen vorkommt. D.h. diese Teilfoge kann in n Suffixen gefunden werden und alle Suffixen haben gleiche Prefix nämlich die Teilfolge. Damit gibt es sicherlich ein innerer Knoten, der die Teilfolge darstellt und genau $n (n \geq 2)$ Kinder besitzt. Besitzt ein innerer Knoten kein Kind, kann die dargestellte Teilfolge auch nicht n mal in S vorkommen. In dem obigen Algorithmus werden alle innere Knoten besucht, die Kinder besitzen. Am Ende werden entweder solche Teilfolgen ausgegeben wenn innerer Knoten mit n Kinder vorhanden ist oder eine leere Menge ausgegeben wenn solche Knoten nicht existiert damit wird die Foreach Schleife enden und der Algorithmus terminieren.

(b) Repeats

Ein *Repeat* ist ein Substring S eines Textes T der mehrfach (mindestens zweimal) exakt gleich in T vorkommt. Ein Maximal Repeat ist ein Repeat, der weder links noch rechts erweitert werden kann (also sowohl links wie rechts unterschiedliche Zeichen in T vorkommen (links-divers, rechts-divers)). Ein $MaximalRepeatPair$ in T ist ein Tripel $(p_1; p_2; n')$, das die Positionen p_1 und p_2 eines Maximal Repeats in T und seine Länge n' angibt.

(i) Wie kann man Maximal Repeats effizient finden? Laufzeit?

Maximal Repeats ist ein Repeats, der weder links noch rechts erweitert werden kann. Analog zum n Repeats Teilstrings Problem in (a) können wir jeden inneren Knoten finden dann überprüfen, ob der linksten Rand erweitert werden können. Wenn es nicht möglich ist dann stellt der aktuellen Knoten ein maximal Repeats dar. Ein maximal Repeats in S kann in Laufzeit $O(|S|)$ mit Hilfe des Ukkonens Algorithmus finden.

(ii) Wie viele Maximal Repeats kann es maximal (höchstens) in einem String T der Länge n geben?

Theorem 1. *Es kann höchstens n maximal Repeats in einem String T der Länge n geben.*

Ein Sffuix-Tree kann maximal n innere Knoten besitzen damit ergibt sich höchstens n maximal Repeats für ein Suffix-Tree.

(iii) Sind alle Maximal Repeats von T gleich lang?

Nein. Betrachte String $T =$ GABCEABCFABCF, dann haben offensichtlich maximal Repeat ABC und $ABCF$ unterschiedliche Länge.

(iv) Skizzieren Sie einen Algorithmus, der alle Maximal Repeats in T findet. Analysieren Sie Korrektheit und Laufzeit.

Laufzeitanalyse:

Die Laufzeit liegt in $O(|T|)$, da alle Schleifen in Maximal Repeats können in $O(|T|)$ durchlaufen und links-divers kann auch in linearer Laufzeit bestimmt werden.

Korrektheitanalyse:

Ein maximal Repeats muss laut Definition überprüft werden, ob entweder links oder rechts erweitert werden kann. Rechte Seite muss aber nicht überprüft werden da das gleichen Zeichen in Suffix-Tree enthalten muss wenn sclche Zeichen existiert. Deshalb müssen wir zunächst zeigen, dass die linke Seite nicht erweitert werden kann. Durch Algorithm isLeftDiverse können wir bestimmen, ob ein maximal Repeats immer unterschiedliche Zeichen an linker Seite hat. Wenn ein Knoten ein Kind besitzt, das links-divers ist, dann ist dieser Knoten natürlich auch links-divers. Wenn es noch nicht festgestellt werden kann, können wir checken, ob alle Kinder gleiche Zeichen an der linke Seite haben. Wenn nicht dann ist dieser Knoten ebenfalls links-divers. Dieser algorithm terminiert, da er nach überprüfung entweder $true$ oder $false$ zurückgeben muss. Somit terminiert der Maximal Repeats auch da er entweder zutreffende maximal Repeats oder eine leere Menge zurückgibt.

Maximal Repeats (String T, int n)

---

begin
- `// ` n`: Länge des T`
- char[] t;
- **for** $(i = 0; i < n; i{+}{+})$ **do**
  - $t[i] := T[i]$;
- `// Build Suffix-Tree using Ukkonen's algorithm`
- BuildSuffixTree(t, n);
- `// recorde all left chrarcters of every leaf`
- Dict{} $leftChars$;
- `// ` v_i`: a internal node`
- `// ` $\{u_i\}$`: all childnodes of ` v_i
- **foreach** $(v_i : \{v|v \in T\} \wedge \{u_i\} = \varnothing)$ **do**
  - `// add the left character of all suffixes`
  - $leftChars := leftChars \cup (v_i, T[i-1])$;
- `// ` $maxRepeats$`: all maximal Repeats in ` T
- Set{} $maxRepeats$;
- **foreach** $(v_i : \{v|v \in T\} \wedge \{u_i\} \neq \varnothing)$ **do**
  - `// check if current node is left-divers`
  - **if** (isLeftDiverse($v_i, leftChars$)) **then**
    - $maxRepeats := v_i \cup maxRepeats$;
- **return** $maxRepeats$;

end

---

isLeftDiverse (String node, Dict{} leftChars)

```
begin
    foreach (child : node.u_i) do
        // child is left-divers then node is also
        if (isLeftDiverse(child, leftChars)) then
            return true;
        else
            // check if all left characters of children from current node
            if (leftChar[child] ≠ leftChar[u_i(0)]) then
                return true;
    return false;
end
```

(v) Wie viele Maximal Repeat **Pairs** kann es in T geben? Wie kann man sie alle enumerieren?

Da es höchstens $O(|T|)$ maximal Repeats in T existieren, kann es höchstens $O(|T|)$ Pairs in T geben. Zum enumerieren können alle maximal Repeats bestimmen und p_1 sei die Postion des Knoten und p_2 sei die Postion des Kindes an der die Teilfolge endet. Dann $n' := p_2 - p_1$.

Hinweis: Benutzen Sie kompakte Suffix-Trees.