Universidad Católica San Pablo

Comprometidos con el futuro

Estructuras de Datos Avanzadas
CCOMP 6-1
Suffix Tree

1. Medina Vilca, Yeroen
2. Nuñez Montes, Sebastián
3. Roque Quispe, David
4. Yato Tintaya, Jeisson

MG. Le Guen, Marc-Antoine

SUFFIX TREE

Es una estructura estática que preprocesa una cadena S y responde a una consulta Q(subcadena).

El tiempo de respuesta no depende de la longitud n de la cadena S, sino, de la longitud de la cadena de consulta Q.

Propiedades

Query Time: O(longitud(Q)), aunque a esto debería sumarsele el tamaño del alfabeto.

Cantidad de nodos en el segundo nivel = cantidad de carácteres diferentes en la cadena S.

CAGTCAGG/0 ---- 5 carácteres diferentes

Los nodos internos siempre tienen más de una arista saliente.

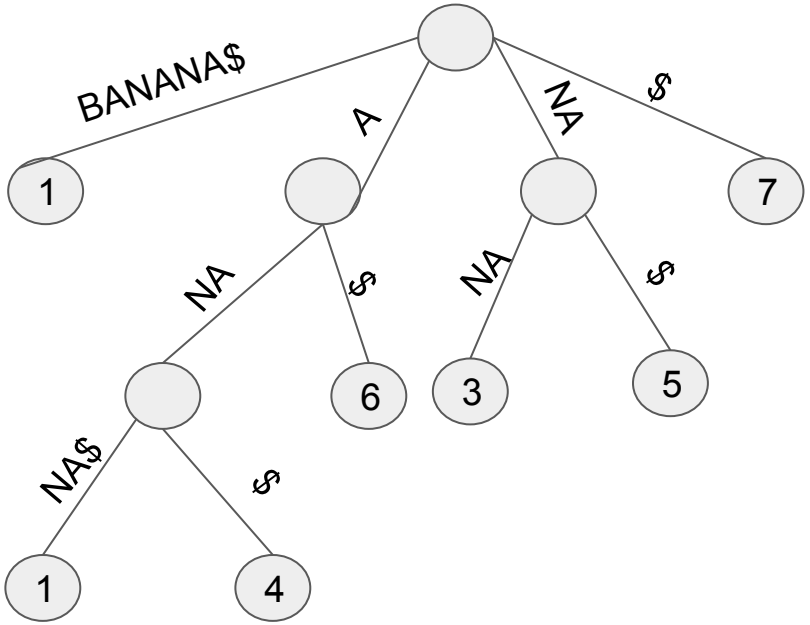
Consulta

Entrada de una consulta: Una cadena Q.

La salida a la consulta debe ser si la subcadena se encuentra en S, y en donde empieza.

Ejemplo

BANANA$



Diferentes propuestas:

- McCreight (1976),
  - Construye la estructura agregando sufijos de tamaño decreciente.
- Ukkonen (1995),
  - Construye la estructura de forma incremental desde el frente mientras mantiene un árbol de sufijos del prefijo ya procesado.

Método Ukkonen

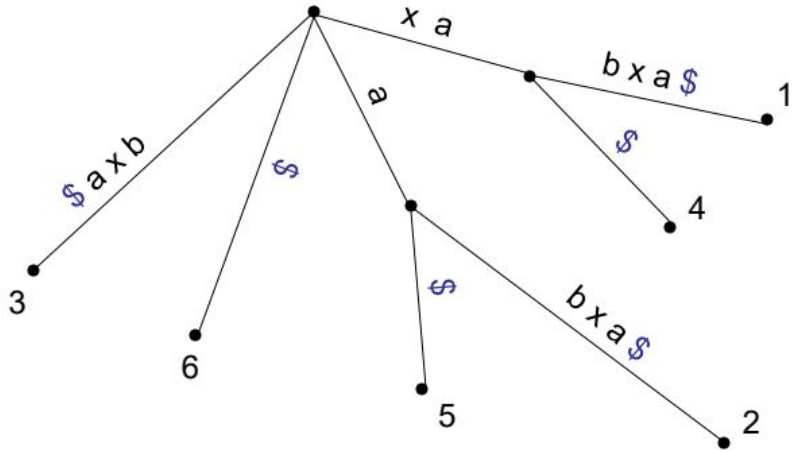
- Implicit Suffix Tree es un árbol obtenido de un Suffix Tree.
-

(1) deleting every copy of $ from the edge labels,

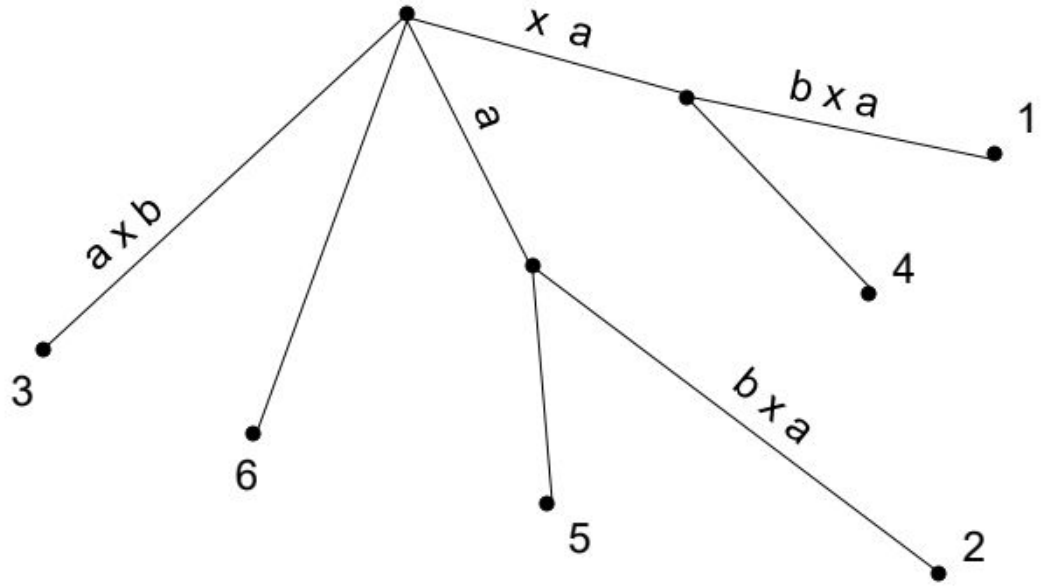
(2) deleting edges that have no label,

(3) deleting unary nodes.

t = x a b x a $
1 2 3 4 5 6

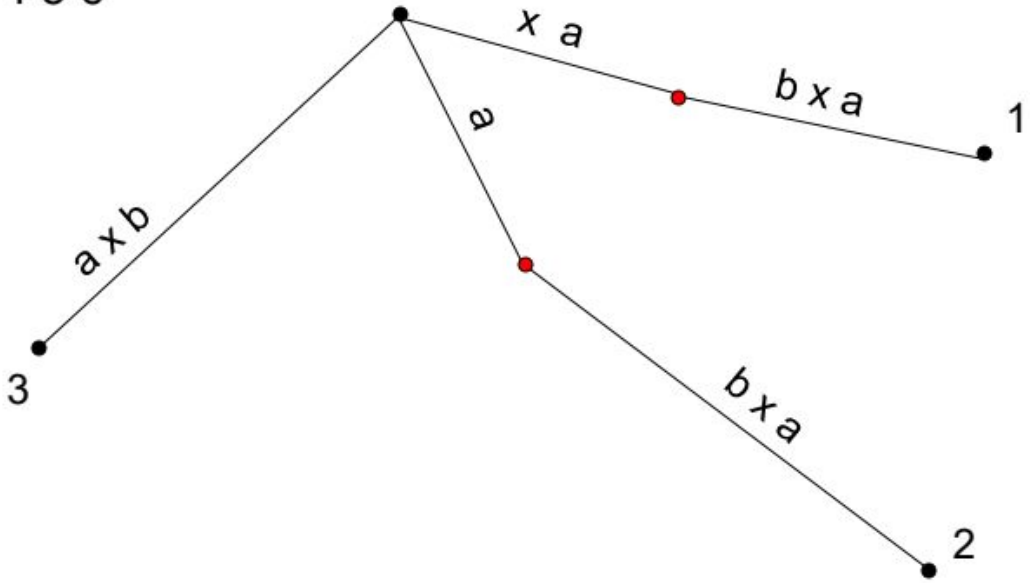


(1) deleting $ from the edge labels



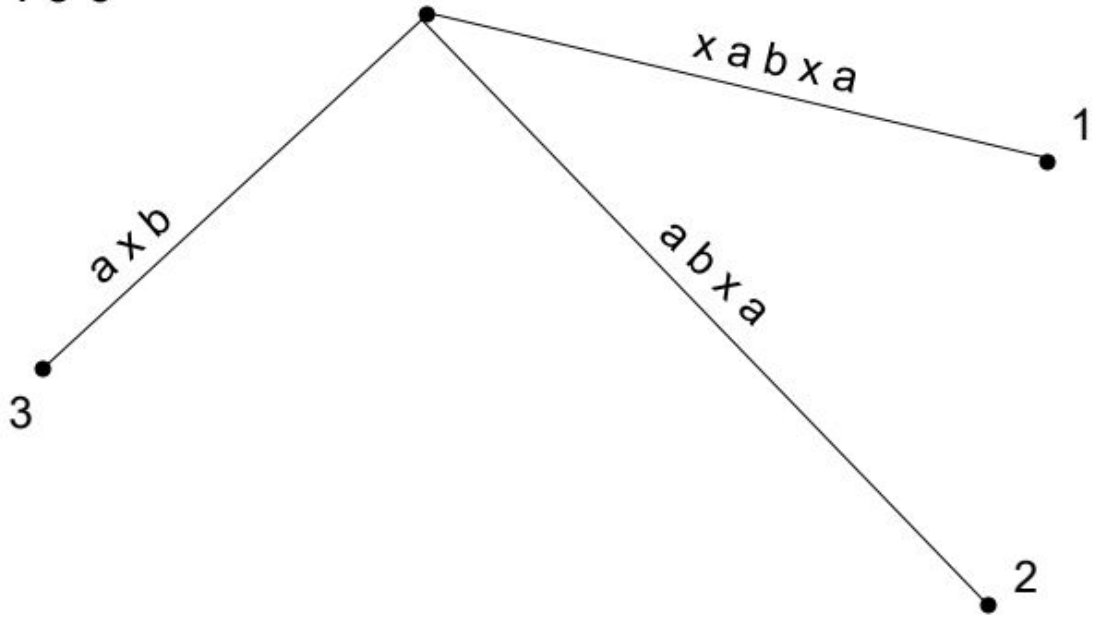
(2) deleting edges that have no label

t = x a b x a $
1 2 3 4 5 6



(3) deleting unary nodes

t = x a b x a $
1 2 3 4 5 6

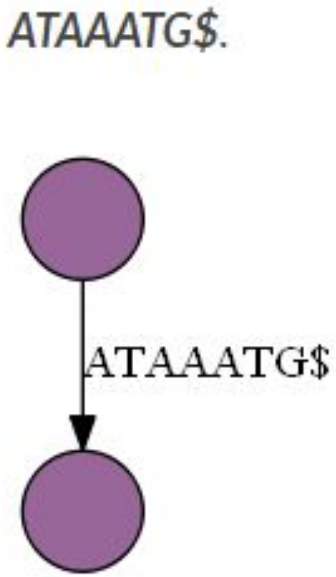


Idea General del algoritmo

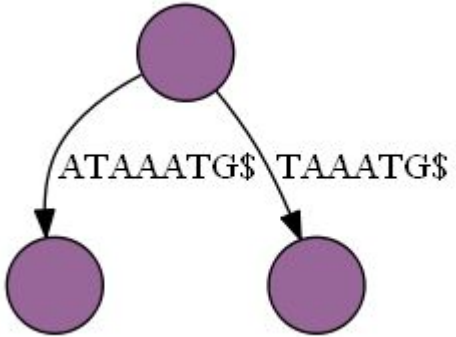
La idea de este algoritmo para construir el árbol de sufijos (Suffix Tree), de una cadena dada es:

```
para i = 1 hasta n – 1 do
    para j = 1 hasta i +1 do
        En el árbol actual encuentre el final del camino desde la raíz
        etiquetado t [j ... i]. Si es necesario, extienda esa ruta agregando
        carácter t [i + 1], asegurando así que la cadena t [j ... i + 1] está
        en el árbol.
    fin;
fin;
```

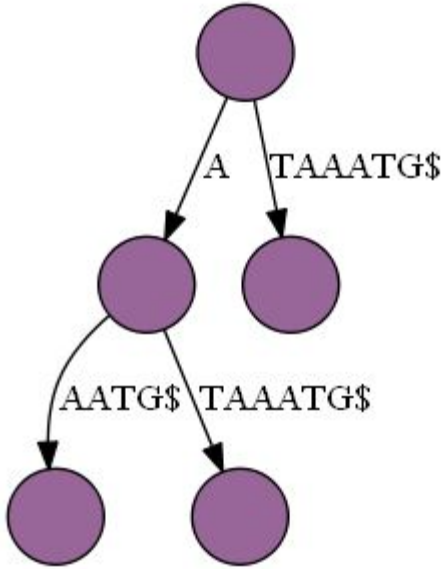
Ejemplo



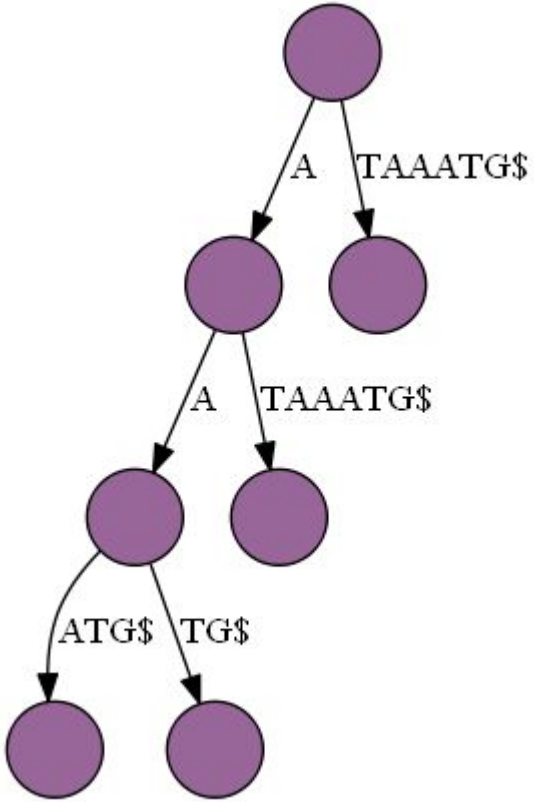
ATAAATG$



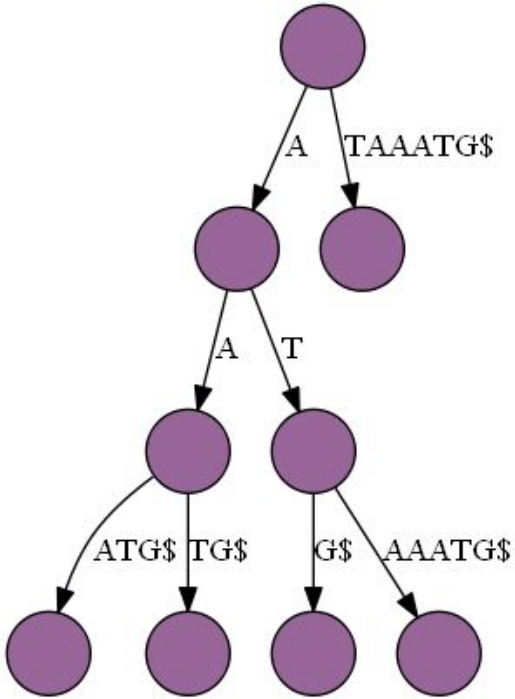
ATAAATG$



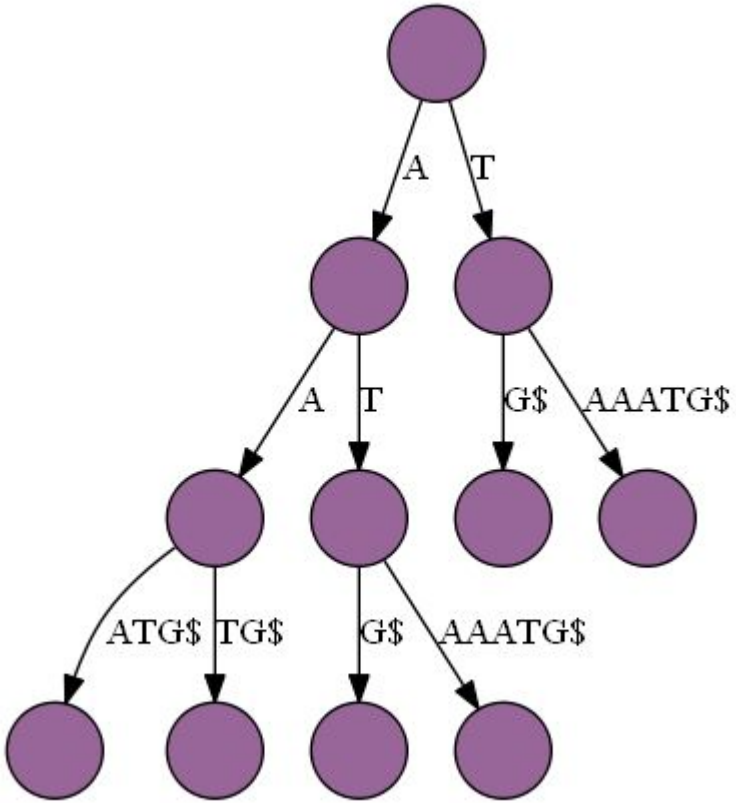
ATAAATG$



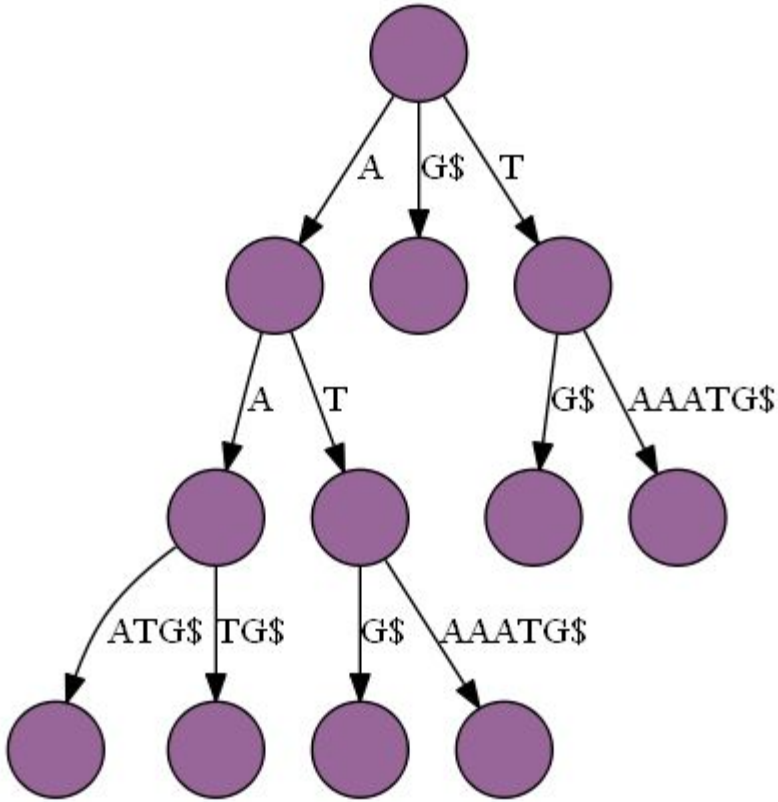
ATAAATG$



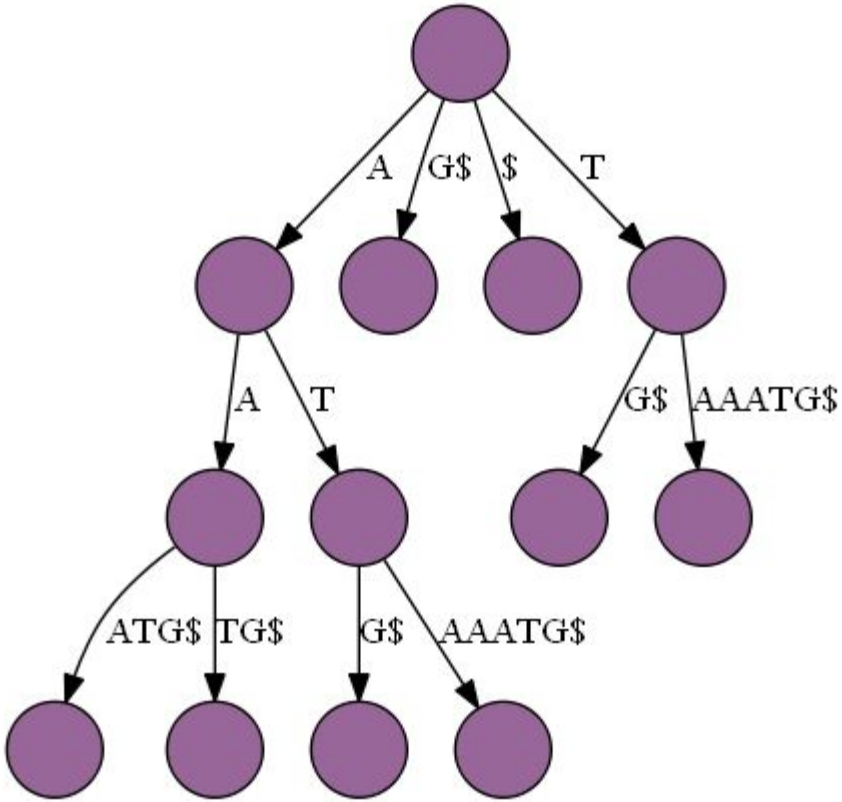
ATAAATG$



ATAAATG$



ATAAATG$



Código

```
struct Node {
    std::string sub = "";    // a substring of the input string
    std::vector<int> ch;     // vector of child nodes

    Node() {
        // empty
    }

    Node(const std::string& sub, std::initializer_list<int>
    children) : sub(sub) {
        ch.insert(ch.end(), children);
    }
};
```

Código

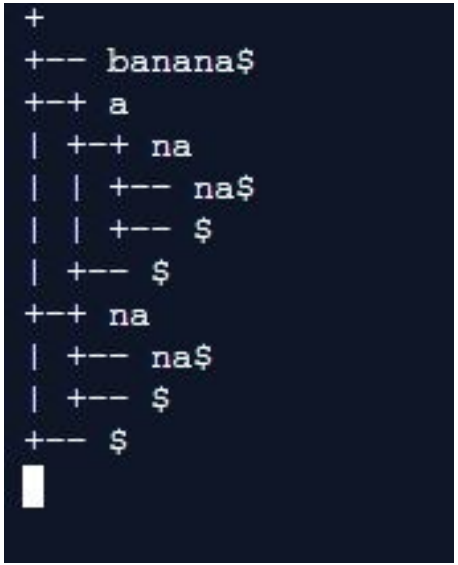
```
struct SuffixTree {
    std::vector<Node> nodes;

    SuffixTree(const std::string& str) {
        nodes.push_back(Node{});
        for (size_t i = 0; i < str.length(); i++) {
            addSuffix(str.substr(i));
        }
    }
```

```
void visualize() {
    if (nodes.size() == 0) {
        std::cout << "<empty>\n";
        return;
    }

    std::function<void(int, const std::string&)> f;
    f = [&](int n, const std::string & pre) {
        auto children = nodes[n].ch;
        if (children.size() == 0) {
            std::cout << "- " << nodes[n].sub << '\n';
            return;
        }
        std::cout << "+ " << nodes[n].sub << '\n';

        auto it = std::begin(children);
        if (it != std::end(children)) do {
            if (std::next(it) == std::end(children)) break;
            std::cout << pre << "+-";
            f(*it, pre + "| ");
            it = std::next(it);
        } while (true);

        std::cout << pre << "+-";
        f(children[children.size() - 1], pre + "  ");
    };

    f(0, "");
}
```

Código

```
private:
    void addSuffix(const std::string & suf) {
        int n = 0;
        size_t i = 0;
        while (i < suf.length()) {
            char b = suf[i];
            int x2 = 0;
            int n2;
            while (true) {
                auto children = nodes[n].ch;
                if (x2 == children.size()) {
                    // no matching child, remainder of suf
                    becomes new node
                    n2 = nodes.size();
                    nodes.push_back(Node(suf.substr(i), {}));
                    nodes[n].ch.push_back(n2);
                    return;
                }
                n2 = children[x2];
                if (nodes[n2].sub[0] == b) {
                    break;
                }
                x2++;
            }
            // find prefix of remaining suffix in common with
            child
            auto sub2 = nodes[n2].sub;
            size_t j = 0;
            while (j < sub2.size()) {
                if (suf[i + j] != sub2[j]) {
                    // split n2
                    auto n3 = n2;
                    // new node for the part in common
                    n2 = nodes.size();
                    nodes.push_back(Node(sub2.substr(0, j), { n3
                    }));
                    nodes[n3].sub = sub2.substr(j); // old node
                    loses the part in common
                    nodes[n].ch[x2] = n2;
                    break; // continue down the tree
                }
                j++;
            }
            i += j; // advance past part in common
            n = n2; // continue down the tree
        }
    }
};

int main() {
    SuffixTree("banana$").visualize();
}
```

Output Código



Performance

Teorema. Suffix tree es una estructura estática que preprocesa una cadena s y admite consultas de subcadenas. Si los nodos trie se realizan como listas enlazadas , la operación `make_suffix_tree` preprocesa una cadena de longitud n sobre un alfabeto A en el tiempo O(| A | n) en una estructura de tamaño O(n), que admite consultas de `find_string` para una cadena q en el tiempo O(| A | longitud(q)).

Aplicaciones

- Para procesamiento de patrones de cadena
- Como las cadenas parametrizadas
  - una cadena parametrizada consta de caracteres del alfabeto subyacente y variables, donde todas las ocurrencias de la misma variable deben reemplazarse por la misma cadena. Esto se puede ver como una clase de cadenas de equivalencia, por ejemplo, un programa bajo el cambio de nombre de las variables.

Aplicaciones

- Para cadenas bidimensionales
  - matrices rectangulares de símbolos de un alfabeto, que se pueden ver como abstracción de imágenes, donde una subcadena bidimensional corresponde a una coincidencia de una traducción de una imagen pequeña en la imagen grande.
- Para encontrar repeticiones en el texto.
- Los suffix tree también pueden construir múltiples cadenas.

Referencias

Advanced Data Structures [Brass 2008-09-08] pp 360 - 367

Ukkonen: https://stackoverflow.com/questions/9452701/ukkonens-suffix-tree-algorithm-in-plain-english/9513423#9513423

Código: https://rosettacode.org/wiki/Suffix_tree

EXTRA: http://www14.in.tum.de/lehre/2018WS/ada/07_SuffixTrees2.pdf