

- New chat
- Convert CFG to CNF
- Apastando modelo logaritmico
- Convertir una CFG a FNC
- New chat
- Coefficiente de Determinación
- Coefficiente Correlación Pearson
- Función escribir lista Prolog
- Instalar Interpreter Prolog Linux
- Instalar Prolog en Linux

Convierte el siguiente pseudocódigo que es el Algoritmo de Convertir una CFG a su normal normal de Chomsky en C++:

```
for all (A → X1X2 ... Xn (con n ≥ 2, Xi ∈ (Σ ∪ V)) do
  for all (Xi) do
    if (Xi = a ∈ Σ) then
      Add the production Ca → aX;
      Replace Xi with Ca in A → X1X2 ... Xn;
    end if
  end for
end for
for all (A → B1B2 ... Bm (con m ≥ 3, Bi ∈ V)) do
  Add m - 2 non-terminal symbols D1D2 ... Dm-2;
  Replace the production A → B1B2 ... Bm with productions:
    A → B1D1
    D1 → B2D2
    ...
    Dm-2 → Bm-1Bm
end for
```

```
void convertCFGtoNormalCFG(vector<Production> &productions) {
    for (auto& prod : productions) {
        int n = prod.right.size();
        for (int i = 1; i < n; i++) {
            if (isTerminal(prod.right[i])) {
                string newNonTerminal = generateNonTerminal();
                addProduction(newNonTerminal, (prod.right[i]));
                prod.right[i] = newNonTerminal;
            }
        }
    }

    for (auto& prod : productions) {
        int n = prod.right.size();
        if (n == 3) {
            vector<string> newNonTerminals;
            for (int i = 1; i < n - 1; i++) {
                newNonTerminals.push_back(generateNonTerminal());
            }

            vector<Production> newProductions;
            newProductions.push_back({prod.left, {prod.right[1], newNonTerminals[0]}});
            for (int i = 1; i < n - 1; i++) {
                newProductions.push_back({newNonTerminals[i], {prod.right[i + 1],
                newNonTerminals[i + 1]}});
            }
            newProductions.push_back({newNonTerminals[n - 2], {prod.right[n - 2],
            prod.right[n - 1]}});

            removeProduction(prod);
            for (auto& newProd : newProductions) {
                addProduction(newProd);
            }
        }
    }
}
```

Note: The above code snippet is just an example and may not be the exact implementation. The function `generateNonTerminal()`, `isTerminal(string)`, `addProduction(Production)`, `addProduction(string, vector<string>)`, `removeProduction(Production)` are not implemented here.

Can you implement those functions?

Sure, here are some possible implementations for the missing functions:

```
set<string> terminals;
set<string> nonTerminals;

bool isTerminal(string symbol) {
    return terminals.count(symbol) > 0;
}

string generateNonTerminal() {
    string newNonTerminal = "N";
    while (nonTerminals.count(newNonTerminal) > 0) {
        newNonTerminal += "1";
    }
    nonTerminals.insert(newNonTerminal);
    return newNonTerminal;
}

void addProduction(Production prod) {
    nonTerminals.insert(prod.left);
    for (auto symbol : prod.right) {
        if (!isTerminal(symbol)) {
            nonTerminals.insert(symbol);
        }
    }
    productions.push_back(prod);
}

void addProduction(string left, vector<string> right) {
    addProduction({left, right});
}

void removeProduction(Production prod) {
    for (auto it = productions.begin(); it != productions.end(); it++) {
        if (*it->left == prod.left && it->right == prod.right) {
            productions.erase(it);
        }
    }
}
```

Note: The above code snippet is just an example and may not be the exact implementation. It requires a vector called `productions` which stores the production and a set of terminal symbols and non-terminal symbols defined as global variables.

- Clear conversations
- Dark mode
- OpenAI Discord
- Updates & FAQ
- Log out