Relatório de MAC0323 - EP4

Nome: Gabriel Haruo Hanai Takeuchi NUSP: 13671636

Como compilar e rodar

Devem estar no diretório os arquivos `main.cpp`, `lib.h`, `Makefile`, e esse `relatorio.pdf`.

Um makefile foi criado para facilitar a minha e sua vida.

```
$ make
$ ./main
...
$ make clean
```

A entrada e saída do programa seguem o formato dos exemplos do enunciado.

Explicando a implementação

Foram usadas 3 estruturas de dados: `Node`, `Digraph` e `NFA` (non-deterministic finite automaton).

Enquanto a classe `Node` suporta a classe `Digraph`, a classe `Digraph` suporta a classe `NFA`.

Node

Um `Node` possui um `vector<int> adj` dos nós adjacentes. Um `int` do `vector` se refere ao índice em relação à `string` da expressão regular.

Digraph

Um `Digraph` possui um `vector<Nodes*> nodes`. Este vetor possui `M+1` nós, um para cada caractere da expressão regular e um extra que representa o estado de satisfação.

NFA

A classe `NFA` possui as seguintes informações:

- `string RE` da expressão regular
- `int M` do tamanho da expressão regular
- `Digraph *G` do autômato finito não-determinístico com `M+1` nós.

O construtor recebe uma expressão regular e

1. Transforma a expressão regular com `+` em uma equivalente com `*` com a função `simplify()`;
2. Cria um autômato finito não-determinístico com `M+1` nós (basicamente, é o código que o Carlinhos já fez).

Reconhecimento de padrões

A função `recognizes()` recebe uma string `txt` e retorna `true` se a string é reconhecida pelo autômato finito não-determinístico, e `false` caso contrário.

Novamente, é o código-base do Carlinhos com a adição de `if`'s para tratar os casos especiais `.`, `\`, `[]`, `[-]` e `[^]`.

Está presente também uma função `dfs()` simples.