Identificando plágio de documentos

1. Introdução e objetivo do trabalho

A identificação de plágio é um dos grandes desafios na comunidade acadêmica. Infelizmente, em todos os níveis de formação, é possível encontrar estudantes que se utilizam de algumas das diversas formas de plágio para alcançar falsos resultados. Dentre os tipos de plágio, estão os plágios de códigos-fonte, o plágio de documentos textuais, etc. Neste último grupo, existem ainda alguns tipos de plágio, como o plágio direto (*copy-and-paste*), o plágio de tradução (cópia traduzida de documentos) e o plágio parafraseado (aquele onde o autor troca palavras ou inverte o sentido da frase original). Este último, aliás, é considerado ainda o tipo de plágio mais difícil de identificar automaticamente e é objeto de estudo da comunidade de linguística computacional.

Computacionalmente, o problema de verificar trechos de texto que foram copiados de outros documentos pode ser custoso, uma vez que existem milhares de palavras em um idioma e, para um texto com 100 palavras, é necessário testar diversas combinações de frases no banco de documentos.

Neste trabalho você estudará o uso de estrutura de dados para a resolução deste problema com o objetivo de entender como a escolha da estrutura de dados influencia na eficiência do sistema, além, é claro, de treinar suas habilidades de programação.

2. Descrição do trabalho

O algoritmo base para identificar plágios será o algoritmo de Rabin-Karp (ver aula 21 no site da disciplina). O algoritmo de Rabin-Karp funciona atribuindo um valor de hash para cada n-gram (substring de tamanho n) do texto, gerando uma sequência de O(n) valores de hash para cada texto. Estes valores hash precisam ser comparados com os valores de hash gerados de cada texto. Em teoria, os documentos que compartilham a maior quantidade de valores hash são aqueles que mais se parecem. Assim, aqueles que mais se parecem são possíveis plágios.

Há, contudo, dois problemas a se tratar: (1) qual a melhor função de hash para representar esses n-gram?; (2) como armazenar esses valores de hash numa estrutura que permita facilmente buscar pelos valores e identificar quais documentos possuem valores de hash iguais?

O seu trabalho irá resolver os dois problemas acima.

Problema 1: testando funções de hash

O algoritmo de Rabin-Karp possui uma função de hash própria definida pelos autores. Esta função será implementada no trabalho. Porém, para analisarmos se esta função é realmente adequada para este problema, você deve utilizar outras duas funções de hash para strings. Faça uma busca no google para encontrar boas funções. **Importante:** implementar funções "desinteressantes" também resultará numa nota "desinteressante".

Problema 2: qual estrutura armazenar os dados

A estrutura básica para armazenamento dos dados será uma tabela de hash que utiliza a técnica de **endereçamento aberto** com tratamento de colisão **hashing duplo** (ou re-hashing). Contudo, você terá diversos arquivos para testar, então será necessário montar uma estrutura que permita armazenar todos os valores de hash criados de forma que consiga realizar as consultas necessárias mas sem grande gasto de tempo e memória.

3. Experimentos que devem ser feitos

Diversos experimentos vão ser realizados para testarmos a qualidade da nossa solução. Para esses experimentos, vamos utilizar uma base de teste chamada METER corpus. O METER é uma base de textos jornalísticos utilizado para verificar plágio entre documentos. A base foi criada com notícias da Press Agency (PA), uma agência de notícias do Reino Unido e notícias de outros jornais como o Times, The Independent, etc. Acontece que diversos jornalistas destes jornais tendem a cometer plágio com as notícias veiculadas pela PA. O objetivo da base é identificar quais textos foram totalmente copiados (*wholly derived*), quais foram parcialmente copiados (*partially derived*) e quais são originais

(*non_derived*). Seu algoritmo deve classificar cada texto em uma destas 3 categorias. Mais explicações sobre a base de testes, verificar o arquivo README dentro do arquivo zip da base (site da disciplina).

Experimento 1: qual a influência do tamanho da tabela e da função de hash na quantidade de colisões na tabela? Para realizar esse experimento, utilize diferentes tamanhos de tabela e aplique cada função de hash implementada. Para cada experimento, contabilize o número de colisões geradas ao inserir todos os arquivos do PA. Crie um gráfico seus resultados.

Experimento 2: qual a influência da função de hash na identificação de plágios? Quanto melhor a função de hash, melhor ela atribui valores distintos para chaves distintas. Utilize as 3 funções de hash para verificar qual função melhor se adapta a esse problema, gerando melhor acurácia. Faça uma tabela com os seus resultados.

Experimento 3: Qual o melhor tamanho de n-gram para identificar os plágios? O algoritmo de Rabin-Karp atribui um valor de hash a cada n-gram do texto. O tamanho desse n-gram fica a critério do programador. Teste quais tamanhos de n-gram são mais adequados, variando o n de 2 até 30). Crie um gráfico mostrando seus resultados.

Experimento 4: Quais os melhores parâmetros para classificar os documentos? Para classificar se um documento foi plagiado, deve-se utilizar uma métrica. Uma métrica simples seria verificar quantos valores hash do documento são encontrados em um documento da PA. Assim, poderia-se pensar que um documento que possui 90% de interseção com outro documento é um documento totalmente plagiado, enquanto um documento que possui até 60% de interseção é um documento parcialmente plagiado. Vamos chamar o primeiro valor (90%) de alpha e o segundo valor (60%) de beta. Para descobrir quais os valores de alpha e beta melhor classificam seus documentos, faça experimentos com diversos valores, sistematicamente, e crie um gráfico com seus resultados.

Para os gráficos do experimento 2 a 4, você poderá fazer uso de uma matriz de confusão para mostrar quantos dados você errou e quantos acertou. Leia mais sobre como fazer matrizes de confusão aqui: http://www.revistabw.com.br/revistabw/matriz-de-confusao/. Faça a matriz 3x3.

Para testar a acurácia do sistema, pode-se utilizar a Medida-F. Neste caso, vamos considerar que os documentos parcialmente plagiados e os documentos totalmente plagiados vão ser chamado somente de documentos plagiados. Contabilize quantos documentos plagiados você encontrou e quantos você errou. Considere:

- True Positives (TP): quantidade de documentos plagiados que você classificou como plagiados
- False Positives (FP): quantidade de documentos não plagiados que você classificou como plagiados
- True Negatives (TN): quantidade de documentos não plagiados que você classificou como não plagiados
- False Negatives (FN): quantidade de documentos não plagiados que você classificou como plagiados.

Com essas variáveis, você pode calcular a medida F, como: seja R (recall) = TP / (TP + FN), seja P (precision) = TP / (TP + FP), então F = 2PR/(P+R). Nos gráficos, use a medida F para verificar a qualidade da sua solução (quanto mais próxima de 1, melhor).

Um relatório deve ser gerado com todos esses experimentos e com a explicação da sua estrutura de dados. Crie um desenho mostrando sua estrutura final (tabela hash com o struct que criou, se usou árvore, etc… Enfim, desenhe pra eu entender rs).

4. Entrega

O trabalho pode ser feito individualmente ou em grupo de até 3 pessoas. Caso escolha fazer em grupo, tome cuidado: se seu parceiro te abandonar, ainda assim você deverá entregar o trabalho completo. Só serão aceitos trabalhos completos.

No relatório é obrigatório que esteja descrito quais itens cada membro do trabalho fez. Informações do tipo “todo mundo fez tudo” não serão aceitas. Mesmo que, hipoteticamente, todos tenham feito tudo, ainda assim deverá estar descrito no relatório cada pedaço do trabalho que cada membro se responsabiliza. Cada membro será avaliado separadamente pela sua parte.

O grupo deve **agendar** um horário com o professor para entrega **pessoal** do trabalho e apresentação **oral** no DCC. O sistema deverá rodar em Ubuntu. Principalmente se fizer em C ou C++, nem comece a programar no Windows.

Atenção para a data de entrega dos trabalhos: caso você deixe para o último dia, não espere que o professor estará disponível para te atender (outro grupo já pode ter reservado o horário!). Se não tiver espaço vago para a sua apresentação, será considerado entrega com atraso.

Atenção de novo: todo semestre os alunos perdem muito tempo com problemas na implementação que os alunos que fazem no Linux ou OSX não tiveram. Então, se não usa linux, instale AGORA.

6. Pontuação

Dentre os pontos que serão avaliados, estão:

- Execução do programa (caso o programa não funcione, a nota será zero)
- Corretude do programa (se todas as funções geram resultados esperados)
- Código documentado e boa prática de programação (o mínimo necessário de variáveis globais, variáveis e funções com nomes de fácil compreensão, soluções elegantes de programação, código bem modularizado, etc)
- Pontualidade (quanto mais atrasada for a entrega do trabalho, mais pontos o trabalho perde)
- Relatório bem redigido

Considerações finais:

- O trabalho pode ser feito em C, C++ ou Java