UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Departamento de Ciência da Computação

DCC207 -- Algoritmos 2
Prof. Renato Vimieiro

Trabalho Prático 1 -- Manipulação de sequências

Objetivos

Nesse trabalho serão abordados os aspectos práticos de manipulação de sequências. Especificamente, serão explorados aspectos de implementação de árvores de prefixo aplicada ao problema de compressão de arquivos.

O objetivo secundário é fixar o conteúdo. Entende-se que ao implementar a estrutura o aluno conseguirá compreender melhor os conceitos explorados. Dessa forma, o conteúdo teórico será melhor absorvido e fixado. Além disso, os alunos terão a oportunidade de ver conceitos não abordados na disciplina, no caso específico, um método de compressão de arquivos.

Tarefas

Os alunos deverão implementar um algoritmo para resolver o problema de compressão de arquivos através do método Lempel-Ziv-Welch (LZW). Esse método é baseado em dicionários e, basicamente, substitui strings que se repetem no texto por códigos. Como o algoritmo executa muitas buscas e inserções nesse dicionário, é comum utilizar árvores de prefixo na sua implementação, como discutido por Salomon em seu livro (ver referências). O LZW, apesar de simples, é utilizado em várias ferramentas de compressão, e em alguns tipos de arquivo. O exemplo mais conhecido, talvez, é o formato GIF que utiliza o algoritmo na compressão das imagens. O método foi patenteado e houve muita discussão no passado sobre seu uso na transmissão de imagens via internet. Dada sua eficiência e apelo prático, estudaremos esse algoritmo no trabalho prático, com o foco na sua implementação, mais precisamente, em como as estruturas vistas em sala podem ser usadas para viabilizar o algoritmo. A seguir é apresentada uma breve descrição do LZW e da tarefa.

O LZW foi proposto por Terry Welch em 1984 como uma implementação eficiente do método LZ78 proposto por Lempel e Ziv em 1978. Como dito, a ideia central do algoritmo é substituir strings que se repetem no texto por códigos, diminuindo assim o número de bytes gravados na saída. O algoritmo inicia gerando um dicionário contendo o código para todos os símbolos do alfabeto em questão. Conforme a proposta original, esse dicionário possui inicialmente 256 entradas referentes aos símbolos da tabela ASCII. Em seguida, o algoritmo consome símbolos do arquivo de entrada um-a-um. Ao ler um novo símbolo *x*, o algoritmo concatena x a uma string *I*, lida anteriormente, e verifica se a string *Ix* já foi armazenada no dicionário. Caso ela já

tenha sido armazenada, o algoritmo atualiza a string *I* com a nova string *Ix*, e repete o processo. Caso a string *Ix* não tenha sido armazenada ainda, o algoritmo imprime o código referente a *I* no arquivo de saída, insere a string *Ix* associada a um novo código (sequencial) no dicionário, atualiza *I* com *x*, e repete o processo. Esse processo é repetido enquanto ainda houver símbolos na entrada.

A descompressão segue um processo análogo. O dicionário agora é inicializado com códigos associados aos símbolos do alfabeto. O algoritmo então consome códigos do arquivo de entrada (comprimido) um-a-um. Ao ler um código, o algoritmo verifica se esse já foi inserido no dicionário. Caso tenha sido, o primeiro símbolo dessa string é concatenado a string anterior, o resultado é adicionado ao dicionário, caso já não esteja, e impresso na saída. Finalmente, a string anterior é atualizada com a string recuperado pelo código e o processo se repete. Caso o código não esteja no dicionário, a string é concatenada ao seu primeiro símbolo, o resultado é adicionado ao dicionário e impresso na saída.

Uma descrição mais detalhada dos processos de compressão e descompressão pode ser obtida nas referências listadas ao final desse documento.

O código usado no LZW pode ter tamanho fixo ou variável. Em implementações com tamanho fixo, o número de bits usados é fixado antes do início da compressão. Essa é a abordagem proposta inicialmente para implementar o algoritmo. Sua intenção era limitar o uso de memória pelo algoritmo, já que, o tamanho do dicionário fica limitado ao número máximo de códigos possíveis. Usualmente, eram usados 12 bits na codificação, mas há implementações mais recentes com 15 ou 16 bits. Outra possibilidade é usar uma implementação com tamanho variável. Nesse caso, inicia-se a compressão com 9 bits (a tabela de códigos iniciais é fixa em 8 bits pela codificação ASCII) e o tamanho é incrementado bit a bit à medida que mais códigos são necessários. Em implementações de tamanho variável também se adota um tamanho máximo para o número de bits pelas mesmas razões da abordagem fixa. É importante lembrar que, nesse caso, o decodificador deve estar atento às mudanças de tamanho do código durante a leitura dos mesmos.

Outro ponto em relação à implementação do algoritmo diz respeito à escolha da estrutura de dados utilizada para o dicionário. Diversos trabalhos apontam que a implementação se beneficia de árvores de prefixo (Tries) para essa função.

Nesse trabalho, você deverá implementar as duas versões do algoritmo LZW discutidas acima. As implementações podem ser feitas em C/C++ ou Python (preferencialmente). Além da implementação do codificador e decodificador do LZW, você também deverá implementar o dicionário usado pelos métodos. Sua implementação deve ser a de uma árvore de prefixo. Como o codificador associa string -> inteiro, e o decodificador inteiro -> string, você deverá implementar uma trie compacta baseada nas strings binárias dos códigos ou texto. As implementações das árvores devem ser completas, incluindo pesquisa, inclusões e remoções de elementos.

Sua implementação deverá receber os parâmetros pela linha de comando (como métodos no Linux). Caso o número de bits máximo não seja informado (parâmetro opcional), deve-se assumir o padrão de 12 bits em ambas as versões. No caso da versão de tamanho fixo, esse é o tamanho fixo a ser usado.

Você também deverá implementar uma opção de teste em que o programa armazenará estatísticas da codificação/decodificação. Essas estatísticas deverão conter a taxa de compressão ao longo do processamento dos arquivos, tamanho do dicionário (número de elementos armazenados e espaço em memória), tempo total de execução, e outras estatísticas que você julgar importantes. Essas estatísticas deverão ser processadas por um outro script que irá gerar um relatório (gráficos, tabelas, etc.) do processo.

Por fim, você deverá realizar testes de funcionamento e desempenho do método implementado com diferentes tipos de dados. Deverão ser usados arquivos texto, imagens (em bitmap) e qualquer outro formato de entrada não comprimido para avaliar o algoritmo. Deverão ser gerados os relatórios como descrito no parágrafo anterior. Esses relatórios serão agregados e analisados em um relatório final.

O relatório final deve ser feito em formato de página/blog e deverá ser publicado junto com o código em repositório aberto no GitHub. O relatório deverá conter uma explicação com a suas palavras sobre os métodos e as implementações. Deverão ser dados exemplos de funcionamento de compressão e descompressão de texto. Você também deve apresentar a análise dos resultados obtidos com os testes realizados. O nível de elaboração do texto e qualidade das análises serão critérios de avaliação. Em outras palavras, o mesmo cuidado com a implementação deverá ser observado nos testes realizados e no relatório produzido.

O trabalho poderá ser feito em **grupos de até dois alunos**.

O que entregar?

Deverá ser entregue um repositório no GitHub contendo todos os arquivos criados na implementação da ferramenta, bem como exemplos usados na explicação. O link para o repositório deverá ser postado no Moodle. O repositório deverá ser mantido privado até 3 dias após a data de entrega. Então, o repositório deverá ser tornado público. Caso a qualidade do trabalho exceda às expectativas do professor, pontos extras poderão ser atribuídos.

Política de Plágio

Os alunos podem, e devem, discutir soluções sempre que necessário. Dito isso, há uma diferença bem grande entre implementação de soluções similares e cópia integral de ideias. Trabalhos copiados na íntegra ou em partes de outros alunos

e/ou da internet serão prontamente anulados. Caso haja dois trabalhos copiados por alunos/grupos diferentes, ambos serão anulados.

Datas

Entrega: 19/11/2024 às 23h59

Política de atraso

Haverá tolerância de 30min na entrega dos trabalhos. Submissões feitas depois do intervalo de tolerância serão penalizados, incluindo mudanças no repositório.

- Atraso de 1 dia: 30%
- Atraso de 2 dias: 50%
- Atraso de 3+ dias: não aceito

Serão considerados atrasos de 1 dia aqueles feitos após as 0h30 do dia seguinte à entrega (sexta-feira). A partir daí serão contados o número de dias passados da data de entrega.

Referências

- https://web.mit.edu/6.02/www/s2012/handouts/3.pdf
- https://www.davidsalomon.name/DC4advertis/DComp4Ad.html