Trabalho Prático 3:
MyTeX

Algoritmos e Estruturas de Dados III – 2016/1
Entrega: 28/06/2016

1 Introdução

Ao construir um texto, é desejável que ele fique visualmente agradável. Para isso, os programas de edição de texto em geral possuem uma opção para justificação do texto, a fim de manter a uniformização do comprimento das linhas. Neste trabalho, você terá de implementar três algoritmos para inserir quebras de linha em um texto: uma solução ótima usando programação dinâmica, uma heurística gulosa e um algoritmo exaustivo (força bruta).

2 Função de custo

Para que a estética do texto fique harmoniosa, o ideal é que cada linha possua o mesmo comprimento e que não sobrem linhas ao final de uma página. No entanto, dependendo do texto de entrada, nem sempre isso será possível. Você pode assumir que o texto de entrada não terá quebras de linha ("$\backslash n$"), e deverá quebrar as linhas apenas onde houver um espaço em branco. Isso quer dizer que o espaço em branco onde a linha foi quebrada dará lugar à quebra ("$\backslash n$"). Assuma também que não existirão dois espaços em branco consecutivos.

Em geral, em cada linha sobrará uma quantidade de espaços em branco ao final, e também poderão sobrar linhas sem texto ao final da página. Assim, seja L o comprimento máximo que uma linha suporta e H o número máximo de linhas que uma página suporta, ambos números inteiros, sua tarefa é minimizar a seguinte função de custo:

$$k(H - |l|)^X + \sum_{\forall l_i \in l} k(L - length(l_i))^X$$

onde l é o conjunto de linhas em que o texto foi dividido, $|l|$ é o número de linhas geradas, $length(l_i)$ é o comprimento da linha i gerada, sem contar com a quebra de linha, e k e X são parâmetros.

Note que o expoente serve para balancear o comprimento das linhas, gerando um custo maior quando os espaços estão concentrados em poucas linhas do que quando estão bem distribuídos. O LaTeX utiliza expoente 3, mas para este trabalho, o expoente deverá ser mantido como parâmetro. O custo para uma linha que excede a capacidade L (fazendo com que $(L - length(l_i))$ seja negativo) ou para uma página que excede H (fazendo com que $(H - |l|)$ seja negativo) deve ser infinito. Todo o texto deverá caber em uma página. Você pode assumir que os valores caberão em variáveis do tipo **int** e que sempre haverá uma forma de quebrar o texto sem estourar a capacidade das linhas e da página.

3 O que deve ser implementado

Você deve implementar três algoritmos: um algoritmo ótimo que utiliza programação dinâmica, uma heurística gulosa e um algoritmo ótimo força bruta.

O `Makefile` que será submetido com o código deve permitir que o programa seja compilado ao digitar o comando *'make all'* no terminal, que deve produzir o executável `tp3.exe`. É de extrema importância que esses critérios sejam seguidos à risca, para que seu código seja compatível com os scripts de teste. Caso contrário, o seu TP não poderá ser testado e, consequentemente, você irá tirar 0.

4 Entrada e Saída

A entrada será passada como um arquivo de texto que terá três linhas: a primeira linha possui dois inteiros separados por espaço, que são os valores de L e H, respectivamente. A segunda linha também possui dois inteiros separados por espaço, que são, respectivamente, o expoente da função de custo (X) e o fator da função de custo (k). A linha seguinte conterá o texto a ser dividido. Eis um exemplo de arquivo de entrada:

```
12 6
3 1
Este eh o texto que deve ser dividido em diversas linhas
```

A saída deverá ser em um arquivo que conterá na primeira linha um inteiro indicando o valor da função de custo da respectiva solução, e nas demais

linhas, o texto com as quebras de linhas inseridas. Abaixo, um exemplo de arquivo de saída:

```
399
Este eh o
texto que
deve ser
dividido
em diversas
linhas
```

Pode ser que haja mais de uma configuração de quebras de linha que gerem o custo mínimo. No entanto, o texto impresso será contrastado com o valor do custo reportado. Tome cuidado para não inserir espaços extras.

O seu programa deverá ler da linha de comando três parâmetros, **nessa ordem:** (i) o paradigma a ser usado (-d para programação dinâmica, -g para guloso, -b para força bruta); (ii) nome do arquivo de entrada; (iii) nome do arquivo de saída. A seguir, há três exemplos de como seu programa será executado pelo sistema de teste:

```
./tp3.exe -d teste.txt resposta_pd.txt
./tp3.exe -g teste.txt resposta_gu.txt
./tp3.exe -b teste.txt resposta_fb.txt
```

Note que a extensão dos arquivos de entrada e saída não é relevante, seu programa deve tratar com arquivos de texto de quaisquer extensões.

5 O que deve ser entregue

Deverá ser submetido um arquivo *.zip* contendo somente uma pasta chamada `tp3` e dentro desta deverá ter a documentação e sua implementação.

Documentação Poderá ter no máximo 10 páginas e deverá seguir tanto os critérios de avaliação discutidos na Seção 6.1, bem como as diretrizes sobre a elaboração de documentações disponibilizadas no *moodle*. Além desses requisitos básicos, a documentação do TP3 deverá ter:

1. A equação de recorrência do algoritmo de programação dinâmica, se implementado, e uma explicação clara de como e por que ela funciona.

2. Explicação de por que sua heurística gulosa não é ótima (dica: procure um contra-exemplo e use-o para explicar quais são os critérios que são ignorados no seu algoritmo guloso).

3. Experimentos mostrando a variação no tempo de execução de acordo com o paradigma utilizado.

4. Experimentos mostrando a variação da qualidade de solução ao usar programação dinâmica e o algoritmo guloso.

5. Distribuição do número de caracteres por linha, variando L, H e os parâmetros da função de custo.

Implementação Código fonte do seu TP (*.c* e *.h*) e um arquivo *Makefile* para realizar a compilação do seu código, atendendo aos critérios descritos na Seção 3.

6 Avaliação

Sua implementação deverá conter, no mínimo, a heurística gulosa e o algoritmo força bruta. No entanto, o algoritmo de programação dinâmica NÃO É EXTRA.

Seu trabalho será avaliado de acordo com a qualidade do código e documentação submetidos. Eis uma lista **não exaustiva** dos critérios de avaliação que serão utilizados.

6.1 Documentação

Introdução Inclua uma breve explicação do problema que está sendo resolvido no seu trabalho e um resumo da sua solução.

Solução do Problema Você deve descrever a solução do problema de maneira clara e precisa, detalhando e justificando os algoritmos e estruturas de dados utilizados. Para tal, artificios como pseudo-códigos, exemplos ou diagramas podem ser úteis. Note que documentar uma solução não é o mesmo que documentar seu código. **Não** é necessário incluir trechos de código em sua documentação nem mostrar detalhes de sua implementação, exceto quando estes influenciem o seu algoritmo principal, o que se torna interessante.

Análise de Complexidade Inclua uma análise de complexidade de tempo e espaço dos principais algoritmos e estrutura de dados utilizados. Cada complexidade apresentada deverá ser devidamente **justificada** para que seja aceita.

Avaliação Experimental Sua documentação deve incluir os resultados de experimentos que avaliem o tempo de execução de seu código em função de características da entrada. Cabe a você gerar entradas para esses experimentos. Por exemplo: se esse trabalho fosse sobre ordenação, seria interessante mostrar como o tempo de execução de cada algoritmo varia quando o número de itens a serem ordenados aumenta. Para tal, um gráfico mostrando o tempo de execução em função do tamanho da entrada pode ser interessante. Você também deve interpretar os resultados obtidos. Comente sobre cada gráfico ou tabela que você apresentar mostrando o que é possível concluir a partir dele. **DICA:** Use o guia e exemplos de documentação disponíveis no moodle.

6.2 Implementação

Linguagem & Ambiente O seu programa deverá ser implementado na linguagem **C** e poderá fazer uso de funções da biblioteca padrão da linguagem. Trabalhos que utilizem qualquer outra linguagem de programação e/ou que façam uso de outras bibliotecas que não a padrão serão zerados. Além disso, certifique-se que seu código compile e funcione corretamente nas máquinas **LINUX** dos laboratórios do DCC.

Casos de teste A sua implementação passará por um processo de correção automatizado e, portanto, o formato da saída do seu programa deve ser idêntico àquele descrito nessa especificação. Saídas com qualquer divergência serão consideradas erradas, mesmo que as divergências sejam *whitespaces*. e.g. espaços, *tabs*, quebras de linha, etc. Para auxiliá-lo na depuração do seu código, será fornecido um pequeno, **não-exaustivo**, conjunto de entradas e suas respectivas saídas. É seu dever certificar-se que seu código funciona corretamente para qualquer entrada válida.

Alocação Dinâmica Algoritmos e estruturas de dados deverão fazer uso de memória alocada dinâmicamente (`malloc()` ou `calloc()`). Certifique-se que seu programa utiliza essas regiões de memória corretamente, pois os monitores penalizarão implementações que realizam *out-of-bounds access* e que tenham vazamento de memória (não desalocar memoria dinâmica).
DICA: Utilize `valgrind` antes de submeter o seu TP.

Qualidade do código Seu código também será avaliado no quesito de legibilidade, dando atenção, porém não limitando-se, aos seguintes items: (i) **INDENTAÇÃO**; (ii) nomes de variável e função descritivos e claros; (iii) Modularização adequada; (iv) Comentários dentro de funções, explicando o

que certos trechos mais complicados fazem; (v) Comentários fora de funções, explicando, em alto-nível, o que as funções mais importantes fazem; (vi) funções concisas que desempenham somente uma tarefa; (vii) **Proibido uso de variáveis globais**; (viii) Quem usar `goto` zera a matéria =].
DICA: Consulte o exemplo de código no moodle

7 Considerações Finais

Dicas Tome cuidado para não gerar overflow ao somar o custo nas variáveis. Estude as referências indicadas ([Erik Demaine, 2011], [Cormen, 2009])

Atrasos ESTE TP NÃO TERÁ ENTREGAS COM ATRASO, NEM ENTREVISTAS, isso porque é o último trabalho do semestre, e a correção tem de ser mais rápida para o fechamento das notas.

Referências

[Cormen, 2009] Cormen, T. H. (2009). *Introduction to algorithms*. MIT press.

[Erik Demaine, 2011] Erik Demaine, M. (2011). Dynamic programming ii: Text justification, blackjack. Disponível em https://www.youtube.com/watch?v=ENyox7kNKeY.