INF 213 - Roteiro da Aula Pratica

Objetivo: praticar o uso de arvores binarias de pesquisa.

→ LEMBREM-SE DE USAR PAPEL E CANETA COMO RASCUNHO ANTES DE IMPLEMENTAR <<--

Arquivos fonte e diagramas utilizados nesta aula:
https://drive.google.com/open?id=1p7FAU_t_mdZvuJ4Z2gDNZAOPfdAb5U12

Etapa 1

O arquivo MySet.h contém a implementação parcial de uma classe Conjunto utilizando uma ABP. Tal implementacao ainda não suporta a operacao *erase* (isso sera visto no futuro), mas permite inserir e pesquisar elementos na arvore.

Essa implementacao está faltando tres métodos MUITO importantes. Quais sao eles? Os implemente e use o arquivo testaMySet.cpp para testa-lo.

Etapa 2 (similar aos exemplos vistos em sala)

Considere o seguinte problema: dadas varias palavras (separadas por espaco, tab ou newlines), conte a quantidade total de palavras fornecidas e a quantidade de palavras unicas.

Crie um programa chamado contaPalavras.cpp . Tal programa devera ler strings da entrada padrao e, no fim, imprimir uma linha contendo dois inteiros: o total de palavras e o numero de palavras unicas. Seu programa devera esperar um unico argumento. Tal argumento podera ser ou a string “myvec” (sem aspas) ou a string “myset”.

Se o argumento for “myvec” → use um objeto do tipo MyVec (use a classe fornecida em MyVecNewIterator.h) para armazenar as palavras unicas ja lidas. Ou seja, para cada palavra lida primeiro veja se ela ja está no container. Se ela não estiver a insira. O numero de palavras unicas sera, obviamente, o numero de elementos no container ao final do seu programa.

Se o argumento for “myset” use um objeto do tipo MySet (use a implementacao que você completou na Etapa 1) para armazenar as palavras unicas.

Exemplos de entrada e saida:

```
$ ./a.out myset < testePalavras1.txt
3 3

$ ./a.out myset < testePalavras2.txt
10 4
```

Meca o tempo de execucao considerando os diferentes arquivos fornecidos na pasta entradasExemploContaPalavras. Explique o motivo da diferenca no tempo de execucao em tais arquivos. Compare os tempos usando a flag -O3 (otimizacao do codigo gerado, não do algoritmo) do g++.

Como poderiamos melhorar a performance da implementacao usando lista por contiguidade (isso pode ser feito facilmente!)? Discuta isso com o professor e colegas e pense nas vantagens e desvantagens de cada alternativa.

Como poderiamos melhorar a performance para o caso de teste onde a implementacao com ABP fica lenta?

Etapa 3

Uma importante aplicacao de ABPs e' na implementacao de "mapas" (tambem conhecidos como dicionarios), que sao basicamente estruturas de dados dinamicas que mapeiam um objeto em outro.

Vejam exemplos de uso em testaMyMap e veja a implementacao do mapa em MyMap.h . Basicamente a diferenca entre o MyMap e o MySet e' que MySet armazena apenas chaves, enquanto o MyMap armazena pares do tipo chave, valor (sendo que apenas as chaves sao utilizadas nas buscas na ABP).

Obs:

- Essa implementacao foi feita rapidamente apenas para mostrar a ideia de mapas. Provavelmente ha varias formas de melhorar a sua eficiencia e qualidade (por exemplo, para evitar complicar a classe não fornecemos versoes constantes de algumas funcoes).
- Essa implementacao não foi muito testada. Logo, pode conter pequenos bugs! Agradeco se encontrarem/indicarem problemas nela (ou em qualquer outro programa implementado nesta disciplina)!
- Note que usamos a classe generica pair da STL (uma classe muito útil utilizada para armazenar pares de tipos arbitrarios). Mais informacoes: http://www.cplusplus.com/reference/utility/pair/

Utilizando a classe MyMap, crie um programa chamado geraCodigo.cpp . Tal programa devera ler palavras (separadas por espaco, tab ou newlines) da entrada padrao e, entao, gerar um codigo unico (de 0 ate n-1, onde n e' o numero de palavras distintas) para cada palavra. O codigo 0 devera ser atribuido a primeira palavra unica lida, o codigo 1 devera ser associado a segunda palavra unica encontrada e assim por diante.

Para cada palavra lida do arquivo, seu programa deve imprimir uma linha com a palavra e seu codigo (separados por um espaco em branco).

Veja abaixo um exemplo de entrada (a entrada e' o mesmo conteudo do arquivo testePalavras2.txt) e a saida esperada (a primeira linha da saida abaixo apresenta o comando utilizado para gera-la).

```
abc 123 123 copia copia
hello abc
abc copia copia
```

```
$ ./a.out < testePalavras2.txt
abc 0
123 1
123 1
copia 2
copia 2
hello 3
abc 0
abc 0
copia 2
copia 2
```

Etapa 4

Note que e' necessario que as chaves possuam pelo menos dois operadores de comparacao: o operador < e o > . Como nossa estrutura de dados sabe se duas chaves sao iguais (e, assim, evita inserir valores duplicados) sem usar o operador ==? E' possivel reimplementar o codigo para que ele exija que a chave possua apenas o operador < ?

Discuta isso com o professor e com colegas (esta etapa não precisa ser entregue)

Submissao da aula pratica:

A solucao deve ser submetida ate as 18 horas da proxima Segunda-Feira utilizando o sistema submitty (submitty.dpi.ufv.br). Envie todos os arquivos fonte (tanto os arquivos .h e .cpp fornecidos neste laboratorio quanto os que você implementou). Atualmente a submissao so pode ser realizada dentro da rede da UFV.