

INF333 - Programacao Competitiva

Estruturas de Dados não Lineares

Profs. Andre Gustavo, Giovanni Comarela, Salles Magalhaes



Dica do dia

- Antes de tudo, dica: ******* use scanf/printf em vez de cin/cout se possível (evita TLE em algumas situações) *******

Árvores binárias balanceadas

- Problema: dado um conjunto de inteiros S e um conjunto de consultas Q , ver rapidamente se cada inteiro está em S .
 - Ex: $S = \{4, 1, 99, 123, 444, -1, 23\}$, $Q = \{4, 22, -1\}$
- Como fazer isso rapidamente?

Árvores binárias balanceadas

- E se as consultas forem dinamicas?
- Ex:
 - Insira 4
 - Insira 10
 - Insira 5
 - Consulte 5
 - Consulte 6
 - Remova 10
 - Insira 90
 - Insira 10
 - Insira 1
 - Insira 5
 - Consulte 10

Árvores binárias balanceadas

- ABPs - Árvores Binárias de pesquisa (balanceadas...) permitem realizar operações de: inserção, remoção, consulta, findMin, findMax, sucessor, antecessor, travessia, de forma eficiente.
- Elementos precisam do operador < implementado (como sabe que é igual?)

Árvores binárias balanceadas

- Na STL a classe set implementa uma ABP. Se s for um set:
 - s.insert(x): insere x no conjunto em tempo $O(\log n)$ -- não armazena duplicados!
 - Por que insert e não push_back?
 - s.find(x): retorna um iterador para x em tempo $O(\log n)$. E se x não estiver no conjunto?
 - NUNCA altere a chave de um elemento já inserido no set (se necessário, remova, altere e insira novamente -- $O(\log n)$)...
 - s.erase(x): remove x (ou o elemento apontado por x, se iterador) em $O(\log n)$
 - s.size(): tamanho de s (em tempo $O(1)$)
 - s.count(x): retorna quantas vezes x aparece no conjunto (0 ou 1) em $O(\log n)$
 - s.begin()/end(): retorna iteradores (de acesso **SEQUENCIAL** -- qual a consequência disso?). Elementos são acessados em ordem crescente! (isso é muito útil).
 - s.lower_bound(x), s.upper_bound(x): retorna iterador para primeiro elemento \geq ou primeiro elemento $>$ que x (em tempo $O(\log n)$) → Muito útil!
 - Exemplos, construtores, set de struct, set com lambda etc...

Árvores binárias balanceadas

- Exemplos:
 - Contar elementos unicos
 - Unique/sort em vector usando set e `#define all(v)`

Árvores binárias balanceadas

- Maps são muito similares a sets (também usam ABPs).
- Diferença: armazenam pares do tipo <chave, valor>
- Forma simples de pensar em um map: um array dinâmico onde a chave pode ser qualquer coisa!
 - `map<string, double> m;`
 - `m["dpi"] = 3.14;`
 - `m["ufv"] = 0.001592;`
 - `cout << m["dpi"] + m["ufv"] + m["teste"] << "\n";`

Árvores binárias balanceadas

- Maps são muito similares a sets (também usam ABPs).
- Diferença: armazenam pares do tipo <chave, valor>
- Forma simples de pensar em um map: um array dinâmico onde a chave pode ser qualquer coisa!
 - `map<string, double> m;`
 - `m["dpi"] = 3.14;`
 - `m["ufv"] = 0.001592;`
 - `cout << m["dpi"] + m["ufv"] + m["teste"] << "\n";`

Chave que não existe: criada (valor definido com construtor padrão do tipo)

Árvores binárias balanceadas

- Na STL a classe map implementa uma ABP. Se `m` for um map:
 - `m[k] = v`: armazena `v` na “posição `k` de `m`” em tempo $O(\log n)$ -- não armazena chaves duplicadas.
 - Se existe → atualiza ; senão cria novo.
 - Atenção: o simples acesso pode criar um novo elemento no map usando o construtor padrão do valor. Ex: `int i = m[“teste”]` → [] não funciona com map constante!
 - Vantagem disso: por exemplo, contar frequência de strings.
 - `m.find(k)`: retorna um iterador para o elemento de chave `k` em tempo $O(\log n)$. E se `x` não estiver no conjunto?
 - Qual a diferença entre `m.find(k)` e `m[k]` ? (além do tipo de retorno)
 - Maps armazenam pares → iteradores apontam para pares (first é a chave, second é o valor).

Árvores binárias balanceadas

- Na STL a classe map implementa uma ABP. Se m for um map:
 - `m.insert(make_pair(k,v))` : tenta inserir no map o elemento v com chave k. Retorna um pair contando um iterador (apontando para o novo elemento) e um bool (dizendo se foi inserido ou se ja existia)
 - `m.erase(x)`: remove elemento com chave x (ou o elemento apontado por x, se iterador) em $O(\log n)$
 - `m.count(x)` retorna quantas vezes x (chave!) aparece no map (0 ou 1) em $O(\log n)$
 - `m.begin()/end()`: retorna iteradores (de acesso SEQUENCIAL). Elementos sao acessados em ordem crescente! (isso é muito útil).
 - `m.lower_bound(x)`, `m.upper_bound(x)`: retorna iterator para primeiro elemento \geq ou primeiro elemento $>$ que x (em tempo $O(\log n)$)
 - `m.size()`: tamanho ($O(1)$)
 - Exemplos, construtores, map com lambda, insert, etc...

Árvores binárias balanceadas

- Exercícios:
 - Dado uma lista de strings (possivelmente com duplicadas), crie códigos sequenciais únicos para elas.
 - Crie um map onde em cada chave podemos ter vários valores. Exemplo: mapeia para cada cidade (nome da cidade:string) as várias pessoas (nomes) morando nela.
 - Considere o seguinte map: `map<int, string> m;`
 - Isso se parece com que outro tipo/container do C++?
 - Qual a vantagem em relação a esse outro tipo? desvantagem?

Árvores binárias balanceadas

- Mais exercícios...
 - A classe `priority_queue` da STL não permite alterar a prioridade de elementos já inseridos (muitos algoritmos exigem isso).
 - Por simplicidade, assuma que não teremos valores repetidos (mas a prioridade poderá ser repetida).
 - Como poderíamos contornar tal problema?
 - Dica: a função “`begin()`” de um `set` retorna o “menor” elemento (ou maior, dependendo do operador de comparação)
 - Problema: `sets` não permitem valores repetidos... Como contornar isso?

Árvores binárias balanceadas

- C++ também possui multiset e multimap (`#include <set>`, `#include <map>`)
- Basicamente:
 - multiset: set com valores repetidos
 - Maior diferença é: `count()`, `insert()`, `erase()`
 - `m.erase(5)`: apaga TODAS cópias do número 5 do multiset
 - `m.erase(m.find(5))`: encontra uma cópia de 5 e a apaga
 - multimap: map com chaves repetidas
 - Maior diferença: não fornece o operador `[]`
 - `find` retorna o iterador para um par com a chave fornecida (podem ter vários pares com mesma chave)
 - `lower_bound`, `upper_bound`: retorna os iteradores para o intervalo contendo valores com a chave passada.
 - `equal_range` (também presente em `set`, `map`): retorna um par de iteradores definindo um intervalo.
- Veja exemplos...

Tabelas hash

- C++11 possui sets/maps implementados usando um outro tipo de ED: tabelas hash.
- Map com tabela hash: `unordered_map`
- Set com tabela hash: `unordered_set`
- Interface muito similar a sets/maps tradicionais
- Qual é a principal diferença?
- Quais métodos “importantes” você acha que não estão disponíveis?
- O que é uma tabela hash?
- Vantagem?

Tabelas hash

- Operacoes de consulta/alteracao em tempo medio $O(1)$
- Desvantagens:
 - Não ordenados
 - Exigem um “hasher” (já pronto para tipos do C++)
 - Exigem operador de comparacao ==
 - Uso de memoria normalmente um pouco maior do que estruturas tradicionais (normalmente ok, a não ser que você crie um vetor de tabelas hash ou algo parecido).
- Exemplo: `timeInsert.cpp` compara o tempo de se inserir (ou `push_back`) 20M inteiros em `set`, `unordered_set`, `vector` e lista (fatores “log” podem fazer com que a diferenca aumente com entradas maiores).
 - Eficiencia: `unordered_set` e `vector`+`sort`+busca binária (se não houver alteracoes)
 - Se precisar de dados ordenados e atualizacao: `set`

Outras EDs

- Há várias outras EDs importantes (muitas não disponíveis nas bibliotecas padrões)
 - Tries
 - Fenwick tree (código na pasta desta apresentação **MUITA ATENÇÃO COM ÍNDICE 1!!!!!!!!!!***)
 - Espécie de “set de inteiros”, onde é possível adicionar e remover “de uma vez” K (K pode ser até negativo) elementos com valor X.
 - Permite consultar em $\log(N)$ a quantidade de elementos \leq a um valor Y
 - Exemplo: contar trocas do bubble sort (tem problema parecido na lista...)
 - Estruturas para conjuntos disjuntos

*      **Eu avisei !!!!**     



Exemplos de problemas:

- UVA 417: Word Index
- UVA 11286 - Conformity
- UVA 10815 - Andy's First Dictionary
- LightOJ-1112

Lembretes:

- Lista 1
- Campeonato 1 na quinta; chegar 18:20 ; trazer material impresso (terá acesso ao cppreference no computador)

Material recomendado

- <https://www.topcoder.com/community/competitive-programming/tutorials/power-up-c-with-the-standard-template-library-part-2/>
- <http://shygypsy.com/tools/bst.cpp> (BST com count de elementos em um intervalo)
- <https://www.geeksforgeeks.org/binary-indexed-tree-or-fenwick-tree-2/>
(fenwick tree --  **CUIDADO** , pois índice ≥ 1)