Importância de Estruturas de Dados adequadas para Programação Avançada.

SCC0210 - Alg. Avançados e Aplicações

Motivação

- Estrutura de Dados (ED) é uma maneira de armazenar e organizar dados
- EDs diferentes têm propósitos diferentes
  - Ao projetar um algoritmos é preciso escolher a que permita **inserções, buscas, remoção e consulta de forma eficiente**.
  - A ED não vai resolver sozinha o seu problema, e pode não alterar a complexidade do seu código, mas certamente fará diferença no tempo de execução em certas situações.
- Conhecer como é implementada uma certa ED fará a diferença na implementação:
  - Mapas em C++ (STL) são árvores balanceadas binárias de busca e, portanto, a escolha dela pode deduzir, em muito, o tempo de processamento do seu código

STL C++ - Standand Template Library

- O uso de C/C++ não é mandatório neste curso, mas é altamente recomendada.
- Python é ótimo, mas muitas vezes não é apropriada para a codificação de algoritmos avançados.
  - Um loop aninhado (2 níveis) em python pode ser cerca de 10 vezez mais lento que em C++.
- Sugiro fortemente que vc conheça (e use) os rudimentos da STL em C++
  - https://en.cppreference.com/w/
  - https://cplusplus.com/
  - https://www.geeksforgeeks.org/the-c-standard-template-library-stl/
- Em especial, sempre atente para a complexidade das funções que a ED lhe oferece.

STL C++

- Tudo que vc implementou em ICC1, ICC2, Alg1 e Alg. Avançados e Aplicações é importantíssimo
  - para que serve cada ED (pilha, lista, fila, etc)
  - natureza estática ou dinâmica, etc
  - Vc aprendeu o que é uma árvore binária de busca, uma heap, etc..
- Mas qualquer ED da STL é implementada da forma mais eficiente possível!
  - Sua implementação pode ser igual, mas dificilmente mais eficiente !
- As EDs da STL podem ser:
  - Lineares
  - Não Lineares

EDs Lineares

- Vetores estáticos:
  - suporte nativo em C e C++
  - Uso fortemente encorajado (não é má prática usá-los, se o problema tem um upper bound - limite máximo - bem definido.
  - Aliás, evite usar malloc/calloc e free
- Vetores dinamicamente alocados (redimensionáveis)
  - Vector<?>
    - uso similar ao estático, mas projetado para suportar redimensionamento automático
    - Opte por Vector se o problema não delimita um nro máximo de elementos
    - push_back() e iterador a disposição do programador

Operação em vetores

- Ordenação:
  - $O(n^2)$ - bubble/selection/Insertion: fuja que nem o diabo da cruz!
  - O(n log n) - quicksort, mergesort: são as opções padrão.
    - STL algoritmh: **sort(), partial_sort ou stable_sort()**
- Busca:
  - O(n) - Busca linear: se possível, evite
  - O(log n) - Busca binária: use se sua entrada estiver ordenada (*)
    - STL algorithm: **lower_bound(), upper_bound(), binary_search()**
  - O(1) - hashing: técnica útil se for necessário acesso rápido (pouco empregado no nosso contexto).

(*): se fará inúmeras buscas, talvez valha a pena ordenar…

Outros estruturas de vetor bastante úteis

- Vetores booleanos **STL bitset<>**: valores 0 ou 1 !!
  - manipulável como um vetor comum, mas cada elemento é interpretado como um booleano
  - economia de memória. Usaremos, por exemplo, em teoria dos números
- **Bitmasks**: suporte nativo C/C++. Inteiro é armazenado como uma sequência de bits.
  - inúmeras aplicações e operações úteis (e rápidas).
  - veja o código deixado na aula1 no moodle.

Outras Eds Lineares

- Listas ligadas - **STL list<>**:
  - inserção tem complexidade constante (*)
  - acesso é linear (da cabeça até a cauda)
  - implementação é normalmente uma lista duplamente encadeada
  - Prefira o uso de Vector
- Stack<>: push() e pop(): O(1)
  - é um *'adaptor'* que usa um container que pode ser, por exemplo, um vector ou deque

  Queue<>:

  - é tb um 'adaptor' de O(1) para inserção e remoção.
  - Muito utilizado em várias de nossos códigos.
- STL deque<>: fila com dois ponteiros (double-ended queue)
  - push_back(), push_front(): O(1).

Eds não lineares

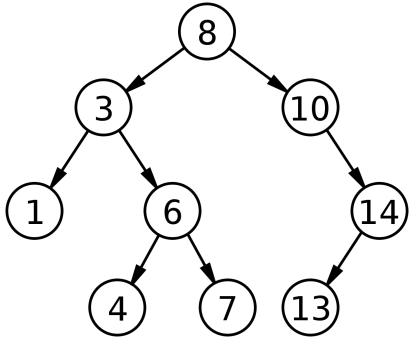
- Melhor opção quando o que se procura é algo de complexidade O(log n)

- Algumas estruturas provêem complexidade de O(log n) para inserção, remoção e busca, como é o caso de **STL map<key,value>**.

- Se fosse armazena em um vetor, estamos falando em complexidade O(n), o que pode ser determinante para exceder o tempo.

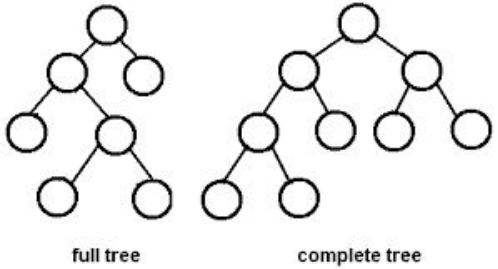
EDs não lineares baseadas em BST (Balanced Binary Search Tree)

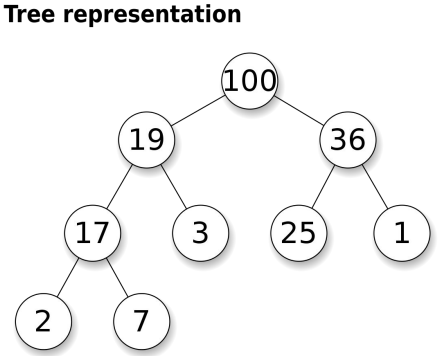
- STL map<key,value> / set<>
  - Estratégia de divisão e conquista !
  - O (log n) para search(k), insertion(k), deletion(k), findMin(), findMax(), successor(k), predecessor(k) dado que, no pior caso, precisamos de apenas log n operações para 'varrer' da raiz até a folha da árvore !
  - Implementação por Árvore rubro-negra (red-black tree)

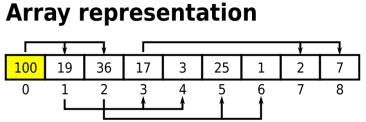


EDs não lineares baseadas em BST (Balanced Binary Search Tree)

- STL priority_queue<>: é também uma BST, exceto que deve ser uma **árvore completa** (*).
- Implementada como uma **max heap** !!
- Nota: priority queues em Python é uma min heap !!
- Muito usada nos nosso problemas:
  - grafos (MST, dijkstra, A*, etc)







(*): cada nível da árvore, exceto possivelmente o último está completamente preenchido. Podem ser armazenadas em um vetor de 1 índice de dimensão n+1

Ainda EDs não lineares baseadas em BST

- STL unordered_map<key, value>
  - é uma hash table !
  - implementação boa a partir de C++11 apenas.. cuidado com isso.
  - veja o que diz o manual em cppreference.com

"Search, insertion, and removal of elements have average constant-time complexity.

Internally, the elements are not sorted in any particular order, but organized into buckets. Which bucket an element is placed into depends entirely on the hash of its key. Keys with the same hash code appear in the same bucket. This allows fast access to individual elements, since once the hash is computed, it refers to the exact bucket the element is placed into."

- O que entendo por isso:
  - acesso a (ou inserção de) tem complexidade requerida de O(1), ou O(n) no pior caso.

Um pequeno desafio para resolvermos em sala.

- Veja PDF (homework.pdf) no moodle, aula 1.
- É bem simples, mas é preciso ter alguns cuidados.
  - Escolha o container apropriado da STL
  - Leia o manual!