INF333 - Programacao Competitiva Estruturas de Dados Lineares

Profs. Andre Gustavo, Salles Magalhaes

Objetivo hoje

- Aprender a ideia básica das principais estruturas de dados lineares: vector, stack e queue
- Ver rapidamente o potencial de várias funcionalidades da STL e do C++11.
  - Vocês “saberão que isso existe”
  - Para realmente aprender → pesquisar e praticar.
  - Fonte de pesquisa recomendada: tutoriais online, cplusplus.com, etc

Estruturas de dados (EDs)

- Armazenam dados (int, bool, objetos, outras EDs)
- Fornecem métodos para realizar operações (de forma eficiente) nesses conjuntos de dados.
  - Inserir
  - Consultar
  - Percorrer

EDs em programacao competitiva

Importante:

- Vantagens e desvantagens de cada estrutura
- Eficiência de diferentes operações nas estruturas
- **Como utiliza-las** (não é o foco de INF213, mas visto de forma resumida)
  - Importante na maioria dos problemas
- Como reimplementa-las
  - Às vezes necessário em alguns problemas
- Normalmente uma mesma operação pode ser feita de inúmeras formas
  - Saber as formas mais fáceis (e suficientemente eficientes) é muito importante
    - Reutilização, código mais fácil → menos chance de erro, implementação mais rápida (para o programador)
  - C++11/17/etc ajudam BASTANTE

Estruturas de dados lineares

- Dados são armazenados de forma linear (em sequência)
- Exemplo: array tradicional

Estruturas de dados lineares

Arrays tradicionais de C/C++:

- Muito utilizados em maratonas. Tamanho pode ser definido com base no tamanho máximo do problema (evitar….).
- Cuidado ao declarar arrays grandes como variáveis locais: são armazenados na pilha e podem causar stack overflow se forem muito grandes.
- Pode ser útil (**pode** ser mais rápido que usar um for): memset, memcpy (cstring)
  - int array[n];
  - memset(array, 0, n*sizeof(int)); //seta os “3o argumento” primeiros bytes de array com 0 (atenção: não funciona com qualquer número!!!)
  - int array2[n];
  - memcpy(array2,array,n*sizeof(int)); //copia os “3o argumento” primeiros bytes de array para array2
- Evite: arrays com alocação dinâmica e strings no “estilo C”

Estruturas de dados lineares

vector (STL): vetor (array) dinamico (cresce dinamicamente)

- push_back(val), pop_back(): O(1) (amortizado para o push_back)
- ~~push_front()~~ (normalmente operações que seriam lentas não são providas)
- [] : O(1)
- size(), empty() : O(1)
- front(), back() : O(1)
- erase(), clear()
- resize()
- reserve() → eficiência!
- Algoritmos da STL podem ser utilizados com ele: e.g. sort()
- Exemplos, resize + [] vs push_back, formas de criar matriz, reinicializar vector (atrib, resize, ..), ...

Estruturas de dados lineares

Iteradores: forma unificada de acesso a estruturas de dados

- v.begin() : retorna iterador para primeiro elemento do vector v
- v.end() : retorna iterador para elemento **APÓS** o ultimo (inexistente..)
- Se it for iterador:
  - *it é uma referência para o elemento apontado por it
  - It++ avança para o próximo elemento
  - It-- volta
  - It+5 : avança 5 passos.
  - it -5 : volta 5 passos
  - it1-it2 : distancia entre dois iteradores
  - it< it2 : true se it estiver antes de it2
  - Não há garantias de que it estará válido se alguns tipos de modificações forem feitas ao vector
- Ha vários tipos de iteradores. Os de vector são de acesso aleatório (permitem it+5), os de lista são apenas bidirecionais
- Exemplos, inc/decremento, soma com int, ...

Estruturas de dados lineares

vector (STL): operacoes com iteradores

Iteradores: forma unificada de acesso a estruturas de dados

- v.insert(it, val): insere o elemento val na posição “it” (no vector v), O(N)
- v.erase(it): apaga o elemento apontado por it (O(N))
- v.erase(beg, end): apaga os elementos em [beg,end) (O(N))

Exemplos de insert, erase

Estruturas de dados lineares

vector (STL): exemplo de utilidade de iteradores: funcoes da biblioteca algorithm (http://www.cplusplus.com/reference/algorithm/)

- Não exclusivas de vectors (mas ha restricoes...)…
- find(beg, end, val): retorna um iterador para a primeira ocorrência de val no intervalo de iteradores [beg,end) -- retorna end se não encontrar
- count(beg, end, val): quantas vezes val aparece em [beg,end) ?
- sort(beg,end): ordena os elementos em [beg,end).
- next_permutation(beg, end): gera a próxima permutação (lexicograficamente) **no intervalo [beg,end)** … retorna false quando a permutação gerada está ordenada. MUITO útil na maratona!
- reverse(beg,end): inverte elementos
- lower_bound, upper_bound (para vector ordenado): tipo de busca binária..
- Exemplos: sort+unique+erase, unique+erase apagando consecutivos, sort criterios

Estruturas de dados lineares

Observacao:

- Um vector<bool> não e’ um container !??!!!??? (isso está inclusive errado no livro CP)
  - Por que?

Estruturas de dados lineares

Observacao:

- Um vector<bool> não e’ um container !??!!!??? (isso está inclusive errado no livro CP)
  - Por performance, quando um programador usa um vector<bool> o C++ codifica os booleanos em bits (um bool = 8 bits normalmente → 8x de economia)
  - Isso causa inúmeros problemas.
  - Exemplo: o código abaixo não compila!

```
int main() {
        vector<bool> v{true,false,true,true,false};
        bool &bref = v[0];
        bref = false;
        for(bool b:v) cout << b << endl;
}
```

Estruturas de dados lineares

list (STL): estrutura de dados lista duplamente encadeada

- Similar ao vector
- Não fornece acesso aleatório aos elementos (não fornece o operador [] , iteradores só podem avançar/voltar uma unidade,...)
- Possuem um sort proprio (sort “tradicional” exige acesso aleatorio)
- push_front (e pop_front) disponíveis e eficientes.
- insert e erase eficientes em qualquer posição da lista (O(1))
- Não muito utilizada em maratonas (falta de acesso aleatório, algumas operações da STL não são eficientes com ela), mas pode ser útil.
- Exercicio: qual algoritmo tradicional importante não funciona com listas encadeadas?
- Exercício 2: qual operação funciona bem com listas (mas não com outras estruturas) ?

Exemplos: push_back, push_front, splice, iteradores

Stack/pilha

- É um caso especial de lista
- Toda inserção e remoção é feita em um dos extremos, o topo
- Operam da forma LIFO: Last In First Out
- Principais operações (e funções da std::stack)
  - Empilhar (push(elem))
  - Desempilhar (pop())
  - Obter item do topo (top())
  - Size (size())
  - Empty (empty())

Queue (fila)

- É um caso especial de lista
- Toda inserção é feita em um dos extremos (frente)
- remoção no outro extremo (trás)
- Operam da forma FIFO: First In First Out
- Principais operações (e funções da std::queue)
  - Enfileirar (push())
  - Desenfileirar (pop())
  - Obter item da frente (front())
  - Size (size())
  - Empty (empty())

Deque (Double-ended queue)

- Similar ao vector (ex: permite acesso aleatório), mas…
- Também permite inserção/remoção eficiente no início:
  - push_front()
  - pop_front()
  - push_back()
  - pop_back()

Priority_queue (fila de prioridades)

- Similar a uma fila, mas os elementos são removidos em ordem de prioridade (quanto “maior o valor”, maior prioridade).
- Principais operações (e funções da std::priority_queue)
  - Enfileirar (push())
  - Desenfileirar (pop())
  - Obter item da frente (**top**() -- estranho! Quem lembra o motivo da função ter esse nome?)
  - Size (size())
  - Empty (empty())

Resumo

- Vector:
  - bom para acesso aleatório (ex: ordenar, busca binária, etc), localidade de memória, inserção de elementos no final, etc.
  - Ruim para: remover/inserir elemento (exceto no final)
- Deque:
  - Quase tão bom quanto o vector, e bom para inserir/remover no início.
- List:
  - Bom para percorrer (não tão boa quanto vector) de forma sequencial, remover/inserir elementos em qualquer lugar, transferir parte de uma lista para outra em tempo O(1)
- Stack/queue/priority_queue:
  - Bons para resolver problemas específicos onde precisamos de LIFO, FIFO ou inserção/remoção com prioridade

C++11

- O C++11 possui inúmeros recursos que facilitam **bastante** a programacao.
- Exemplos:
  - auto (não funciona com parâmetros de funções)
  - range-based for, range com arrays criados por {}
  - Lambda, any_of, all_of, none_of, count_if
  - find_if : retorna iterador para primeira ocorrência onde predicado é true
  - find_if_not…
  - copy_if (atencao: container deve ter tamanho certo, retorna iterador…)
    - Ex: copiar pares de um vector para outro
    - Ex2: dado um vector de vectors, copiar para um segundo vector apenas os que contem apenas numeros pares.
  - Inicialização com {}:
    - Exemplo: push_back em um vector de struct em vez de vector de pair...

Array (C++11)

- Array estatico (tamanho fixo) do C++11
- Declaração: array<int, 3> v; (array com 3 inteiros)
- Vantagens em relação a arrays tradicionais:
  - São “objetos” (podem ser passados facilmente por cópia ou referência)
  - Sabem seus próprios tamanhos (size())
  - Possuem iteradores
  - Podem ser utilizados com algoritmos da STL de forma similar a outros containers (uniformidade)
- Ou seja, tão “barato” quanto arrays e quase tão convenientes quanto vectors.
- Exemplo de como pode ajudar: minimo entre 4 elementos.

Dica

- Containers normalmente possuem construtores que recebem dois iteradores (apontando para os elementos a serem inseridos).
- Exemplo de utilidade: ordenar um vector e pegar elementos únicos com um set (ED que armazena elementos em ordem e remove valores duplicados -- será estudado futuramente).

Exemplo de problema:

- **Balanco de parentesis: UVA 673**
  - https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=614
  - E se entrada pudesse ter também outros caracteres?
    - (2+2)
    - (2-)[3*3]

Extra...

- Pesquise sobre pair (e talvez tuple)
- Pairs sao MUITO uteis em maratonas…
- Veja exemplos de construção, make_pair, …
- Pairs podem ser comparados

Links uteis

- https://www.quora.com/As-a-competitive-programmer-what-C++11-features-should-I-be-aware-of-and-what-are-some-examples-that-illustrate-their-usage
- http://www.modernescpp.com/index.php/std-array-dynamic-memory-no-thanks
- C++14 em maratonas: https://codeforces.com/blog/entry/16262
- Crash course in the STL: https://community.topcoder.com/tc?module=Static&d1=features&d2=082803
- Topcoder STL tutorial: https://www.topcoder.com/community/competitive-programming/tutorials/power-up-c-with-the-standard-template-library-part-1/