Aula 03 - Estruturas de dados em árvore

Vinícius Brito

Março de 2020

Tópicos da aula

1. Map
2. Set
3. Fila de prioridade
4. Bônus: busca binária
5. Exercícios indicados

Mapas

São um componente muito útil da STL, pois permitem tratar elementos inicialmente intangíveis como outros tangíveis. Em outras palavras, pode-se mapear os primeiros nos seguintes. E o melhor: o *map* já ordena os elementos intangíveis, e permite também um acesso rápido à memória (mas não é a estrutura mais eficiente). Exemplo de elemento "intangível" é uma *string*, que pode ser simplesmente tratada como um inteiro. A biblioteca pertinente é a <map>.

No entanto, é um pouco complicado percorrer os elementos do map. Isso porque são necessários ponteiros chamados *iterators*, e vamos evitar usá-los. Felizmente, o acesso individual é tão simples quanto num vetor tradicional. A seguir alguns exemplos de uso do map, os quais estão puramente na linguagem C++. Nos dois primeiros exemplos, é necessário fornecer uma palavra de entrada.

Mapas

Estruturas de dados (II)

Vinícius Brito

Map

Set

Fila de prioridade

Bônus: busca binária

Exercícios indicados

Exemplos de uso do map

```
#include <iostream>
#include <map>

using namespace std;

map<string, int> mp;

int main( ){
    string a;
    mp["arroz"] = 22;
    mp["feijao"] = 23;
    mp["nutella"] = 24;
    cin >> a;
    cout << "Turma " << mp[a];
    return 0;
}
```

Mapas

Exemplos de uso do map

```
#include <iostream>
#include <map>

using namespace std;

map<string, int> mp={{"arroz", 22}, {"feijao", 23},
{"nutella", 24}};

int main( ){
    string a;
    cin >> a;
    if(mp.find(a)!=mp.end( )) cout << "Turma " << mp[a];
    else cout << "Turma nao encontrada";
    return 0;
}
```

Mapas

Exemplos de uso do map

```
#include <iostream>
#include <map>

using namespace std;

map<string, int> mp={{"nutella", 24}, {"arroz", 22},
{"feijao", 23}};

int main( ){
    string a;
    map<string, int>::iterator stonk;
    for(stonk=mp.begin( ); stonk!=mp.end( ); stonk++) cout
<< (*stonk).first << " " << (*stonk).second << endl;
    return 0;
}
```

Conjuntos

Eles funcionam exatamente como o nome sugere: são conjuntos de elementos **sem repetição** e ordenados. São uma estrutura muito queridinha da STL porque realizam todas as operações (inserção, consulta e remoção) em tempo logarítmico. Seu funcionamento é muito parecido com os maps, e a biblioteca associada é <set>.

No entanto, para inserir elementos é necessária a função *insert* (que funciona analogamente ao *push* das filas e pilhas). Elementos inseridos no set são automaticamente ordenados e cada elemento aparece exatamente uma vez, o que se confirma pela função *size*.

Conjuntos

Estruturas de dados (II)

Vinícius Brito

Map

Set

Fila de prioridade

Bônus: busca binária

Exercícios indicados

Exemplo de uso do set

```
#include <iostream>
#include <set>

using namespace std;

set<string> comidas={"nutella", "arroz", "feijao"};

int main( ){
    string a;
    cin >> a;
    comidas.insert(a);
    set<string>::iterator stonk;
    for(stonk=comidas.begin( ); stonk!=comidas.end( );
stonk++) cout << *stonk << endl;
    return 0;
}
```

Fila de prioridade

São outra estrutura queridinha da STL, também por ter tempo de operação logarítmico. Já estão embutidas na <queue>. No entanto, ao contrário da *queue*, o elemento há mais tempo é chamado pela função *top*, e não pela *front*.

Em particular, a fila de prioridade é construída de modo que o elemento no topo seja o maior, embora isso possa ser modificado com auxílio do *vector* (uma estrutura da STL muito usada, e que se parece muito com o *deque*). Em especial, o *vector* serve tanto como vetor dinâmico como uma matriz dinâmica, tornando-o muito interessante, especialmente em grafos!

Fila de prioridade

Estruturas de dados (II)

Vinícius Brito

Map

Set

Fila de prioridade

Bônus: busca binária

Exercícios indicados

Exemplo de uso da priority queue

```
#include <iostream>
#include <queue>

using namespace std;

priority_queue<int> pq;

int main( ){
    pq.push(22);
    pq.push(21);
    cout << pq.top( ) << endl;
    pq.pop( );
    cout << pq.top( );
    return 0;
}
```

Fila de prioridade

Exemplo de uso da priority queue

```
#include <iostream>
#include <queue>
#include <vector>

using namespace std;

priority_queue<int, vector<int>, greater<int> > pq;

int main( ){
    pq.push(22);
    pq.push(21);
    cout << pq.top( ) << endl;
    pq.pop( );
    cout << pq.top( );
    return 0;
}
```

Busca binária

Feita a apresentação de algumas estruturas muito úteis (relaxe, o *vector* será mais explorado depois), agora começaremos a ver a parte mais desafiadora da programação competitiva: os algoritmos! Para isso, comecemos com um clássico e simples algoritmo denominado busca binária. Apresentaremos aqui um exemplo.

Suponha que se deseje saber se um número está no vetor $\{1, 2, 3, 4, 5, 6, 7, 8\}$. Em particular, vejamos se o 8 está. Ora, o elemento na "posição do meio" é o 4, que é < 8. Assim, só precisamos nos preocupar com a metade da direta do vetor. No novo vetor, a "posição do meio" tem um 6, que é < 8. Novamente, descartamos metade desse vetor, obtendo na posição mediana um 7, que é < 8. Finalmente, descartando metade desse novo vetor, obtemos apenas o 8, que é o elemento desejado.

Busca binária

Note que só precisamos de 4 verificações para saber se o 8 está no vetor, e não 8 (percorrê-lo todo). O truque é o "descarte de metades", que sugere uma complexidade logarítmica. Em especial, esse algoritmo tem complexidade $O(log_2 N)$, sendo N o número de elementos no vetor. Fica como exercício verificar que, de fato, essa é a complexidade do algoritmo citado.

Também observe que podemos generalizar essa ideia: suponha que certa propriedade varie monotonicamente. Então, se existir um ponto onde ela ocorra, ele será único. Quem estiver "à esquerda" terá um desvio, enquanto quem estiver "à direita" terá outro. Sabendo o tipo de desvio, podemos saber pra que direção ir, até talvez descobrir onde a propriedade ocorra. Em particular, esse deslocamento descarta metade do espaço amostral.

Busca binária

No código seguinte, queremos saber se um número *a* está numa partição, de *c* a *f*, de um vetor ordenado *v*.

Exemplo recursivo da busca binária

```
bool busca(int a, int c, int f, int v[]){
    if(c<=f){
        int m=(c+f)/2;
        if(v[m]<a) return busca(a, m+1, f, v);
        if(v[m]>a) return busca(a, c, m, v);
        return true;
    }
    return false;
}
```

Fica como exercício implementar o mesmo algoritmo, mas sem usar função recursiva :). Note que o *return* dispensa usarmos o *else*.

Exercícios indicados

Por fim, a prática leva à perfeição. Mais uma vez, cada nome já é um link para o respectivo problema.

- ⇒ Ida à Feira
- ⇒ Dijkstra (nada a ver com grafos)
- ⇒ Soma de Casas
- ⇒ Tradutor do Papai Noel
- ⇒ Transporte de Painéis Solares
- ⇒ Ajude Girafales
- ⇒ Eu Posso Adivinhar a Estrutura de Dados!
- ⇒ Fila do supermercado
- ⇒ Bolsa de Valores