Hash

Hash em C++

Prof. Edson Alves – UnB/FGA

Sumário

Hash **em C++**

Hash na STL

- A STL da linguagem C++ oferece dois contêiners que utilizam *hashes* para obter complexidade média $O(1)$ para as operações de inserção, remoção e buca
- `unordered_set` é um contêiner que abstrai a ideia de conjuntos, armazenando elementos únicos
- Se for necessário armazenar repetições de um mesmo elemento, deve-se utilizar o `unordered_multiset`
- `unordered_map` e `unordered_multimap` são as abstrações equivalentes para dicionários
- Ambas armazenam pares de chaves e valores
- Estas estruturas tem interface semelhante às suas versões que se baseam em árvores binárias de busca balanceadas (`set`, `map`, etc)

unordered_set

Conjuntos baseados em *hashes*

- O `unordered_set` mantém um conjunto de elementos únicos
- O tipo `T` destes elementos é indicado na declaração do conjunto
- Também podem ser indicados na declaração a função de *hash* a ser utilizada, a função de comparação de igualdade entre os elementos e o alocador de memória
- Este contêiner foi introduzido no C++11
- A sintaxe de declaração é dada abaixo:

```
template<
    class Key,
    class Hash = std::hash<Key>,
    class KeyEqual = std::equal_to<Key>,
    class Allocator = std::allocator<Key>
> class unordered_set;
```

Métodos para inserção, remoção e consulta

- O `unordered_set` compartilha com o `set` a mesma interface para inserção, remoção e consulta
- O número de elementos armazenados é dado pelo método `size()`
- Novos elementos podem ser inseridos por meio dos métodos `insert()` e `emplace()`
- Os elementos podem ser removidos através do método `erase()`
- O método `count()` retorna o número de ocorrências de um dado elemento (zero ou um)
- O método `find()` localiza um elemento e retorna o iterador para sua posição (ou `end()`, caso o elemento não se encontre no conjunto)
- Todos estes métodos tem complexidade média $O(1)$

Exemplo de uso do unordered_set

```cpp
#include <bits/stdc++.h>

using namespace std;

int main()
{
    unordered_set<int> xs { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 };

    auto N = xs.size();          // N = 9, o elemento 1 é inserido
                                 // uma única vez

    auto p = xs.insert(4);       // p.second = true, inserção bem
                                 // sucedida

    p = xs.insert(8);            // p.second = false, 8 já pertence
                                 // ao conjunto

    auto x = *p.first;           // x = 8, p.first aponta para o
                                 // elemento inserido

```

Exemplo de uso do unordered_set

```
22
23      N = xs.size();              // N = 11
24
25      auto it = xs.find(5);       // Localiza o elemento 5
26
27      it = xs.erase(it);          // remove o elemento 5
28                                  // it aponta para o próximo elemento
29
30      x = xs.count(1);            // x = 1, apenas uma cópia de 1
31                                  // no contêiner
32
33      x = xs.count(20);           // x = 0
34
35      for (auto x : xs)
36          cout << x << ' ';       // 10 1 34 2 13 3 8 21 55 4
37      cout << '\n';
38
39      return 0;
40  }
```

Métodos relativos ao *hash*

- O `unordered_set` oferece métodos relativos à implementação por meio de *hashes*
- Os métodos `bucket_count()` e `max_bucket_count()` retornam o número de *buckets* (células ou posições) da tabela e o número máximo de posições permitidas pela implementação, respectivamente
- Para determinar quandos elementos colidem na célula j, utilize o método `bucket_size()`
- O fator de carga, isto é, o número de elementos armazenados dividido pelo tamanho da tabela (número de células) é dado pelo método `load_factor()`

Métodos relativos ao *hash*

- O método `max_load_factor()` indica o fator de carga máximo suportado pela tabela
- Caso uma inserção leve a superação deste máximo, a tabela é reconstruída, com um número maior de células
- O tamanho da tabela pode ser alterado por meio dos métodos `rehash()` e `reserve()`
- O primeiro recebe o tamanho T como parâmetro, e modifica a tabela, respeitando a carga máxima
- O segundo recebe o número de elementos N a serem acomodados e modifica a tabela para para comportar tais elementos sem violar a carga máxima

Exemplo de uso do unordered_set

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    unordered_set<int> xs { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 };
    auto x = xs.bucket_count();     // x = 11 buckets

    // Mostra a quantidade de elementos em cada bucket
    for (size_t i = 0; i < x; ++i)
        cout << xs.bucket_size(i) << ' ';
    cout << '\n';                   // 1 2 2 1 0 1 0 0 1 0 1

    x = xs.max_bucket_count();      // x = 1152921504606846975

    auto y = xs.load_factor();      // y = 0.818182, size()/bucket_count()
    y = xs.max_load_factor();       // y = 1

    xs.insert(4);
    y = xs.load_factor();           // y = 0.909091
```

Exemplo de uso do unordered_set

```
21      xs.insert(6);
22      y = xs.load_factor();           // y = 0.478261
23      x = xs.bucket_count();          // x = 23 buckets
24
25      // 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0
26      for (size_t i = 0; i < x; ++i)
27          cout << xs.bucket_size(i) << ' ';
28      cout << '\n';
29
30      xs.rehash(15);
31      x = xs.bucket_count();          // x = 15
32      y = xs.load_factor();           // y = 0.647059
33
34      // 1 1 1 1 3 1 1 0 1 0 0 0 0 1 0 0 0
35      for (size_t i = 0; i < x; ++i)
36          cout << xs.bucket_size(i) << ' ';
37      cout << '\n';
38
39      return 0;
40  }
```

unordered_map

Dicionários baseados em *hashes*

- O `unordered_map` mantém um conjunto de pares, compostos por uma chave e um valor
- Os tipos `Key`, `T` das chaves e dos valores são indicados na declaração do dicionário
- Também podem ser indicados na declaração a função de *hash* a ser utilizada, a função de comparação de igualdade entre os elementos e o alocador de memória
- Este contêiner foi introduzido no C++11
- A sintaxe de declaração é dada abaixo:

```
template<
    class Key,
    class T,
    class Hash = std::hash<Key>,
    class KeyEqual = std::equal_to<Key>,
    class Allocator = std::allocator<std::pair<const Key, T>>
> class unordered_map;
```

Interface

- O `unordered_map` compartilha a interface apresentada para o `unordered_set`
- As funções relacionadas ao *hash* também estão disponíveis
- Ao contrário dos conjuntos, os elementos do dicionário podem ser acessados por meio de suas chaves, através do método `at()` ou do operador `[]`
- A diferença entre os dois métodos reside nas permissões de acesso ao valor retornado: há uma versão método `at()` com permissão apenas para leitura
- Esta versão é útil em métodos que não alteram a instância: observe as declarações abaixo:

```
T& at(const Key& key);
const T& at(const Key& key) const;

T& operator[](const Key& key);
T& operator[](Key&& key);
```

Exemplo de uso do unordered_map

```cpp
#include <bits/stdc++.h>

using namespace std;

int main()
{
    unordered_map<string, double> hs {
        { "Maria", 1.68 },
        { "Lucas", 1.59 },
        { "Pedro", 1.72 },
        { "Joana", 1.81 },
        { "Alberto", 1.85 },
    };

    auto N = hs.size();                                  // N = 5
    auto p = hs.insert(make_pair("Carlos", 1.79));       // p.second = true,
                                                         // inserção ok

    hs["Maria"] = 1.70;                                  // Altera o valor associado à Maria
    N = hs.size();                                       // N = 6
```

Exemplo de uso do unordered_map

```
22      auto it = hs.find("Roberto");                   // it = hs.end()
23      it = hs.find("Lucas");                          // it->second = 1.59
24
25      auto x = hs.count("Pedro");                     // x = 1
26      it = hs.erase(it);                              // remove Lucas
27
28      x = hs.bucket_count();                          // x = 7 buckets
29
30      // Mostra a quantidade de elementos em cada bucket
31      for (size_t i = 0; i < x; ++i)
32          cout << hs.bucket_size(i) << ' ';
33      cout << '\n';                                   // 0 1 3 0 0 0 1
34
35      x = hs.max_bucket_count();                      // x = 329406144173384850
36
37      auto y = hs.load_factor();                      // y = 0.714286, size()/bucket_count()
38      y = hs.max_load_factor();                       // y = 1
39
40      return 0;
41  }
```

Referências

1. **CORMEN**, Thomas H.; **LEISERSON**, Charles E.; **RIVEST**, Ronald L.; **STEIN**, Clifford. *Introduction to Algorithms*, The MIT Press, 3rd edition, 2009.
2. **DROZDEK**, Adam. *Algoritmos e Estruturas de Dados em C++*, 2002.
3. **RADKE**, Charles E. *The Use of Quadratic Residue Research*, Communications of the ACM, volume 13, issue 2, pg 103–105, 1970[1].
4. **STROUSTROUP**, Bjarne. *The C++ Programming Language*, 2013.
5. C++ Reference[2].

[1] https://dl.acm.org/citation.cfm?id=362036
[2] https://en.cppreference.com/w/