Programming: Back to Basics

Tống Tùng Giang

Xin chào!

- Vào ngành game từ 2015.
  - FZ9: Timeshift, Arena of Survivors (Hiker Games).
  - Prison Architect, Grounded: Nintendo Switch (Double Eleven).
  - LEGO® Horizon Adventures™ (Studio Gobo).
  - ... và một vài sản phẩm không được thấy ánh mặt trời.
- Unity, Unreal Engine, custom engine.
- Mobile, PC, console.

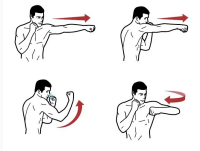
Lịch trình

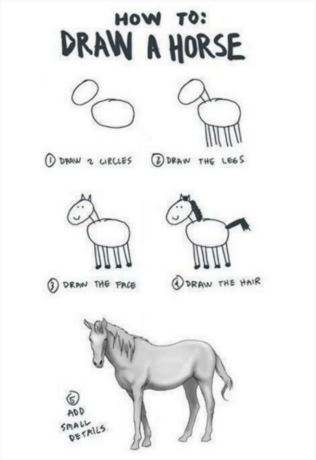
- Ôn lại cơ bản: kiến trúc Von Neumann, bộ nhớ dưới góc nhìn của hệ điều hành.
- Tổ chức kiểu dữ liệu.
- Mảng.
- Danh sách liên kết
- Bảng băm.
- Các cấu trúc dữ liệu khác.

Mục tiêu

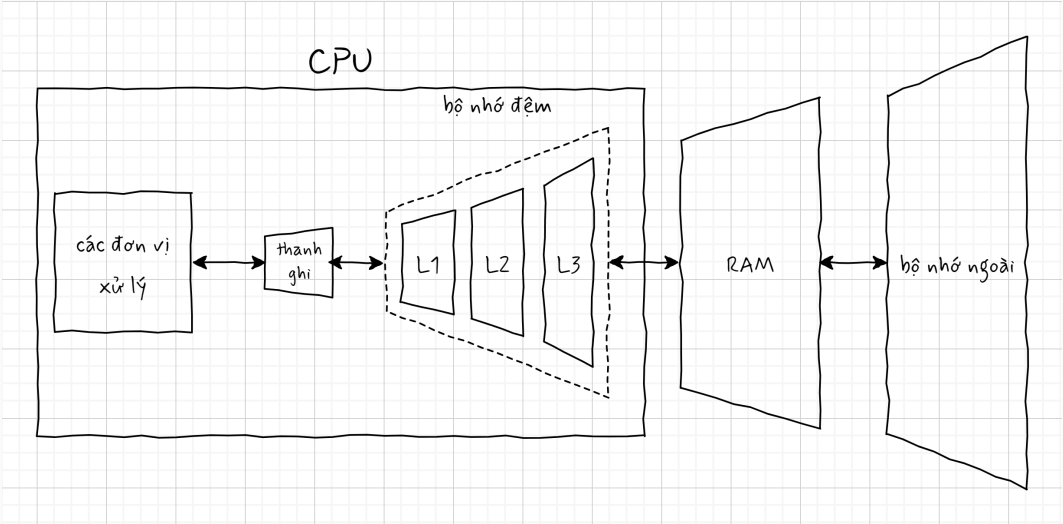
- Lựa chọn cấu trúc dữ liệu phù hợp với bài toán mình đang giải quyết.
- Sửa một vài “lỗi” sử dụng.
- Hiệu năng tốt hơn do dữ liệu được tổ chức một cách thân thiện với phần cứng.
- Logic chương trình gần như giữ nguyên.

Cơ bản != dễ



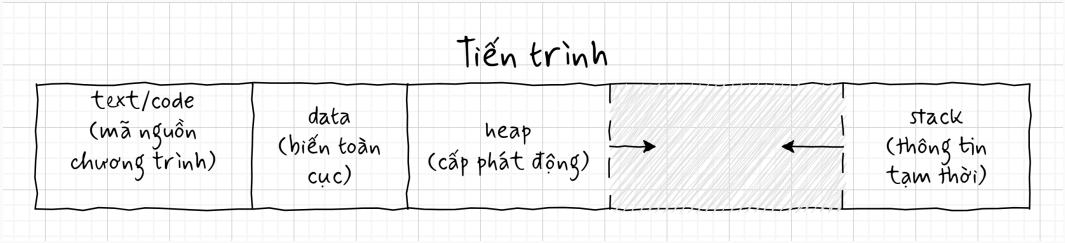

HOW TO:
DRAW A HORSE
1 DRAW 2 CIRCLES
2 DRAW THE LEGS
3 DRAW THE FACE
4 DRAW THE HAIR
5 ADD SMALL DETAILS

Kiến trúc bộ nhớ

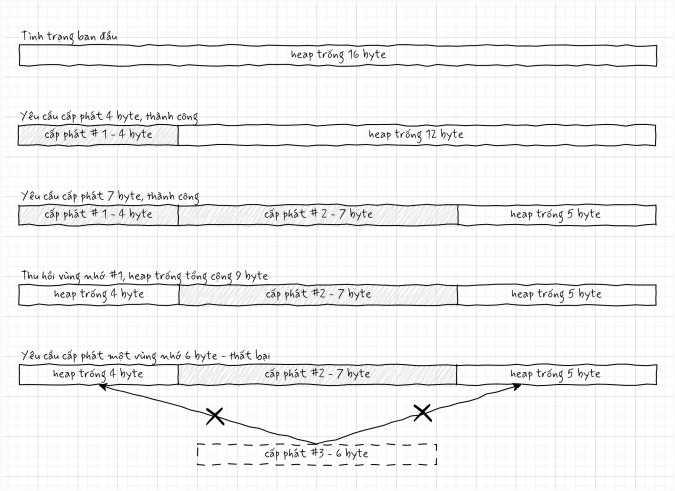


3 tiêu chí: truy cập nhanh, kích thước lớn, giá thành tốt.

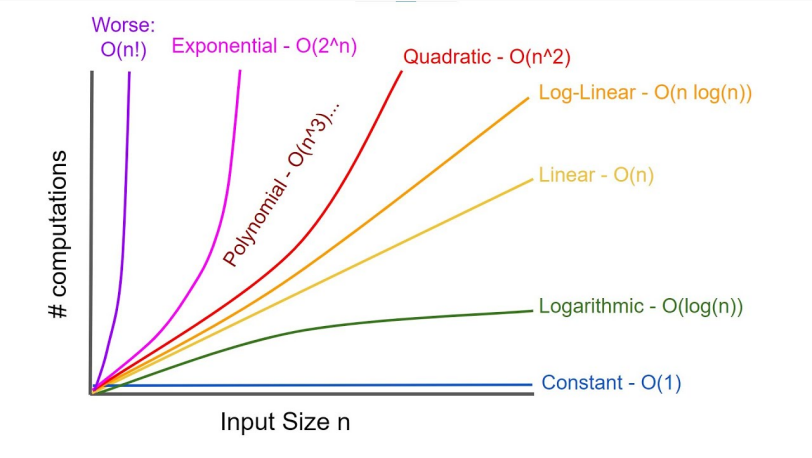
Cách hệ điều hành quản lý bộ nhớ



Cách hệ điều hành quản lý bộ nhớ (tiếp)



Độ phức tạp của thuật toán



Độ phức tạp của thuật toán (tiếp)

std::map<Key,T,Compare,Allocator>::find

```
iterator find( const Key& key );                                  (1)
const_iterator find( const Key& key ) const;                      (2)
template< class K >
iterator find( const K& x );                                      (3)  (since C++14)
template< class K >
const_iterator find( const K& x ) const;                          (4)  (since C++14)
```

1,2) Finds an element with key equivalent to `key`.

3,4) Finds an element with key that compares *equivalent* to the value `x`. This overload participates in overload resolution only if the qualified-id `Compare::is_transparent` is valid and denotes a type. It allows calling this function without constructing an instance of Key.

Parameters

key - key value of the element to search for

x - a value of any type that can be transparently compared with a key

Return value

An iterator to the requested element. If no such element is found, past-the-end (see `end()`) iterator is returned.

Complexity

Logarithmic in the size of the container.

Độ phức tạp của thuật toán (tiếp)

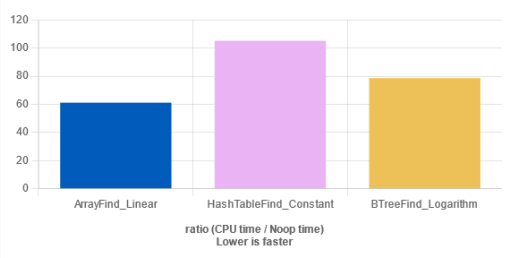
"Dùng thuật toán [tên bất kỳ] đi, $O(1)$ nhanh hơn $O(N)$ là cái chắc!"

Độ phức tạp của thuật toán (tiếp)

- Ký pháp O thể hiện mức tăng trưởng chứ không phải là giá trị tuyệt đối.
- Các yếu tố ảnh hưởng:
  - Tổ chức của cấu trúc dữ liệu trong bộ nhớ.
  - Chi phí tính toán của từng phép toán.
  - Đặc thù của bài toán tương ứng với trường hợp xấu nhất/trung bình/tệ nhất của thuật toán.
  - Kích thước tập dữ liệu.

Độ phức tạp của thuật toán (tiếp)

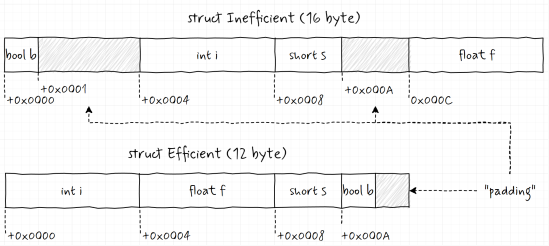
```cpp
// O(N)
std::vector<int> v;
auto vIt = std::find(v.begin(),
                     v.end(),
                     toFind
                    );

// O(1)
std::unordered_set<int> s;
auto sIt = s.find(toFind);

// O(logN)
std::set<int> s;
auto sIt = s.find(toFind);
```



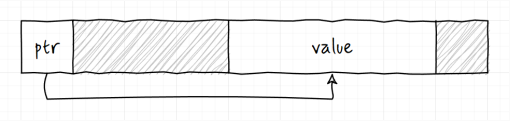
Thứ tự sắp xếp các trường dữ liệu

```
struct Inefficient {
    bool b;     // 1 byte
    int i;      // 4 byte
    short s;    // 2 byte
    float f;    // 4 byte
}; // 16 byte....
```

```
struct Efficient {
    float f;    // 4 byte
    int i;      // 4 byte
    short s;    // 2 byte
    bool b;     // 1 byte
}; // 12 byte
```

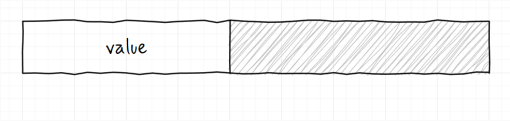


Trực tiếp và gián tiếp




```
struct Indirect {
    T* ptr;
};
```

- Kích thước kiểu dữ liệu có thể nhỏ đi.
- Cache miss khi truy cập giá trị ptr.




```
struct Direct {
    T value;
};
```

- Nếu chỉ cần truy cập giá trị thì ít gặp cache miss hơn.
- Kích thước kiểu dữ liệu lớn.

(*) C#: Kiểu dữ liệu tham chiếu (**class**) và kiểu giá trị (**struct**).

Dữ liệu nóng và nguội

```
class Player
{
    Vector3f     m_Position;
    float        m_Health;
    Vector3f     m_Direction;
    float        m_Speed;
    int          m_Score;
    Item         m_Items[5];
};
```

```
class Player_Cold
{
    float        m_Health;
    int          m_Score;
    Item         m_Items[5];
}


class Player
{
    Vector3f     m_Position;
    Vector3f     m_Direction;
    float        m_Speed;
    Player_Cold* m_Cold;
};
```

Mảng tĩnh

```
T arr[100];
```

- Truy cập ngẫu nhiên: $O(1)$.
- Tìm kiếm: ngẫu nhiên: $O(N)$, có sắp xếp: $O(logN)$.
- Sắp xếp: $O(NlogN)$.
- Kích thước không đổi.

Mảng tĩnh (tiếp)

Cấp phát trong stack: C#

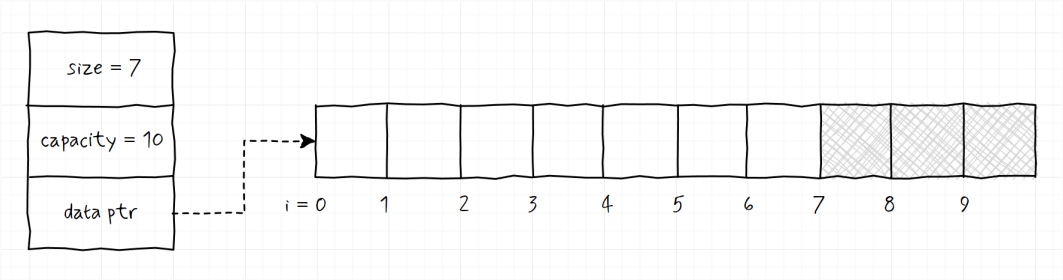
```
int length = ... ;

T[] arr1 = new T[length];                // heap
Span<T> arr2 = stackalloc T[length];     // stack
```

Mảng động

```
std::vector<T> v;
v.push_back([ ... ]);
v.push_back([ ... ]);
```

- Truy cập ngẫu nhiên: $O(1)$.
- Tìm kiếm: ngẫu nhiên: $O(N)$, có sắp xếp: $O(logN)$.
- Sắp xếp: $O(NlogN)$.
- Kích thước thay đổi được.
  - Chèn phần tử mới: vị trí bất kỳ: $O(N)$, cuối: $O(1)$ (*).
  - Xóa phần tử: vị trí bất kỳ: $O(N)$, cuối $O(1)$.

Mảng động (tiếp)



Mảng động (tiếp)

```
std::vector<T> v;
const int NUM = 7749;

for (int i = 0; i < NUM; ++i)
{
    v.push_back([...]);
}
```

Mảng động (tiếp)

```
std::vector<T> v;
const int NUM = 7749;
v.reserve(NUM);              // <--- !
for (int i = 0; i < NUM; ++i)
{
    v.push_back([...]);
}
```

Mảng động (tiếp)

Xóa tất cả các phần tử thỏa mãn điều kiện: $O(N^2)$.

```
for (int i = 0; i < v.size();)
{
    if ([...])
        c.erase(v.begin() + i);      // O(N)
    else
        ++i;
}
```

Xóa tất cả các phần tử thỏa mãn điều kiện bằng cách đổi chỗ với phần tử cuối: $O(N)$.

```
for (int i = 0; i < v.size();)
{
    if ([...])
        v[i] = v[v.size() - 1];      // <--- O(1)
    else
        ++i;
}
```

Mảng động (tiếp)

Xóa tất cả các phần tử thỏa mãn điều kiện bằng erase-move idiom: $O(N)$.

```
int newsize = 0;
for (int i = 0; i < v.size(); ++i)
{
    if (![...])                    // <--- Phủ định
    {
        v[newsize] = v[i];
        ++newsize;
    }
}
v.resize(newsize);
```

Mảng động (tiếp)

Cấp phát trên stack khi có thể:

```
constexpr int MAX_N = 16;
int stackBuffer[MAX_N];
T* arr;
if (n ≤ MAX_N)
    arr = stackBuffer;
else
    arr = new int[n];
```

Mảng nhiều chiều

"Mảng của mảng".

Ví dụ: mảng hai chiều.

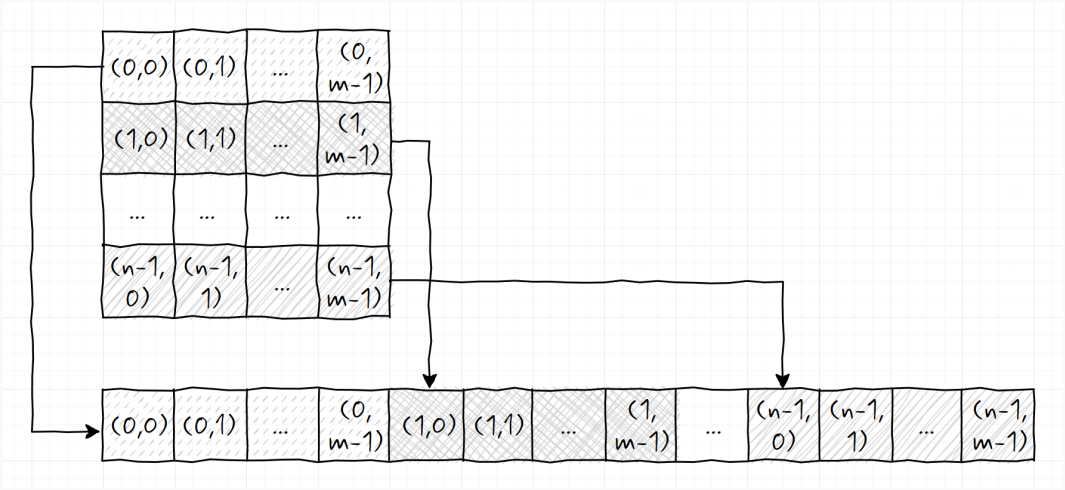
```
T arr[N][M];
```

Bố trí trong bộ nhớ:

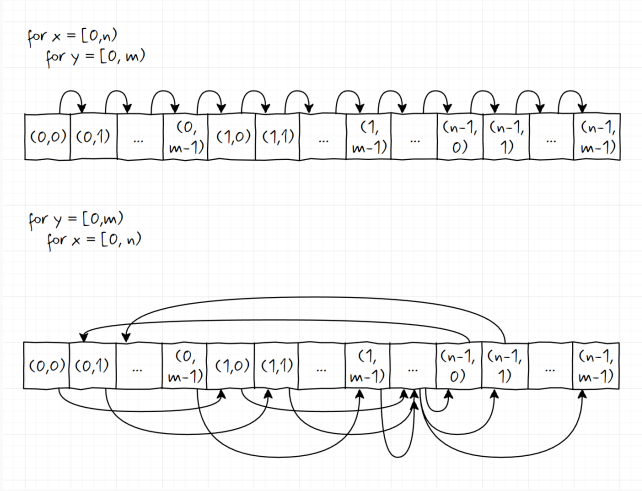
- Trực tiếp hay gián tiếp.
- Theo hàng hay theo cột.

Mảng nhiều chiều (tiếp)

Ví dụ: trực tiếp, theo hàng.



Mảng nhiều chiều (tiếp)



Mảng nhiều chiều (tiếp)

Dùng mảng 1 chiều để biểu diễn mảng nhiều chiều:

```
int get_index(int x, int y, int rows)
{
    return x * rows + y;
}
std::pair<int, int> get_coord(int index, int rows)
{
    return { index / rows, index % rows };
}
```

Mảng nhiều chiều (tiếp)

Dùng mảng 1 chiều để biểu diễn mảng nhiều chiều:

```
T arr[N * M];
// 2D → 1D
int x = rand() % N;
int y = rand() % M;
int i = get_index(x, y, M);
// 1D → 2D
int i = rand() % (N * M);
const auto& [x, y] = get_coord(i, M);
// Duyệt mảng
const int size = N*M;
for (int i = 0; i < size; ++i)
    arr[i] = [...];
```

Bonus: mảng bit

```cpp
bool arrBool[N];
std::bitset<N> arrBit;
```

Xâu

Mảng ký tự + nhiều toán tử.

- Các toán tử này có thể che giấu chi phí thực sự của chúng.

```
std::string s;
s += "Hello";                    // strcat: O(N)
s.push_back(' ');
bool b = (s == "World");         // strcmp: O(N)
```

Xâu (tiếp)

```
std::string s1 { "Anh em oi, " };                    // constructor
std::string s2 = s1;                                 // copy assignment
s2 = "Mot hai ba, do! Hai ba, do! Hai ba, uong!";    // reassign
s1 += s2;                                            // grow
```

Xâu (tiếp)

C#:

- Immutable: `System.String`.
- Mutable: `System.StringBuilder`.

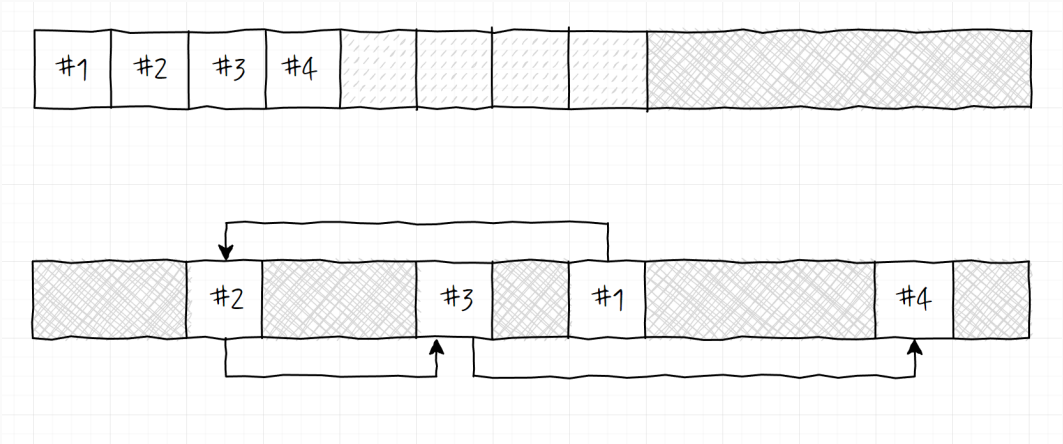
```
string s;
for ([...])
    s += [...];

StringBuilder sb;
sb.EnsureCapacity([...]);
for ([...])
    sb.Append([...]);
```

Danh sách liên kết

```
std::list<T> v;
v.push_back([ ... ]);
v.push_back([ ... ]);
```

- Truy cập ngẫu nhiên: $O(N)$.
- Tìm kiếm: $O(N)$.
- Sắp xếp: $O(NlogN)$.
- Chèn/xóa phần tử: $O(1)$.

Danh sách liên kết (tiếp)



Danh sách liên kết (tiếp)

- Không cần duyệt.
- Chỉ cần thêm và xóa vào vị trí *đã biết*.
- Kích thước tập dữ liệu lớn.

Bảng băm

```
std::unordered_map<K, V> m;
[ ... ]
m[k1] = v1;
```

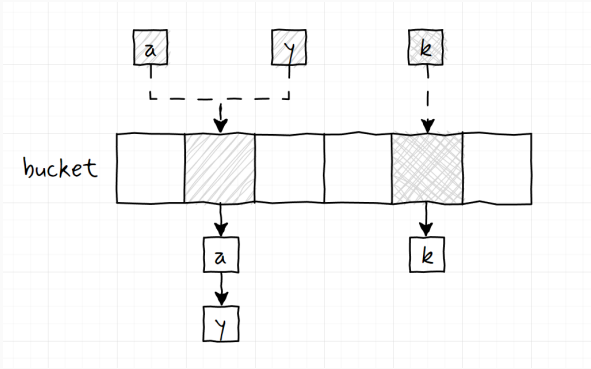
- Truy cập ngẫu nhiên/tìm kiếm: $O(1)$, tệ nhất $O(N)$.
- Kích thước thay đổi được.
  - Chèn phần tử mới: $O(1)$, tệ nhất $O(N)$.
  - Xóa phần tử: $O(1)$, tệ nhất $O(N)$.

Bảng băm (tiếp)

- Bucket?
- Băm (hash)?
- Hệ số tải (load factor)?
- Xung đột (hash collision)?
- Giải quyết xung đột (collision resolution)?

Bảng băm (tiếp)

Giải quyết xung đột bằng kết nối (chaining).



Bảng băm (tiếp)

```
std::unordered_map<std::string, Foo> dict;
dict.reserve(NUM);
```

Bảng băm (tiếp)

Tìm kiếm hai lần: tìm kiếm giá trị

```
std::unordered_map<K, V> m;
[ ... ]
K k1;
auto findIt = m.find(k1);
if (findIt ≠ m.end())
{
    V& v1 = m[k1];
}
```

Bảng băm (tiếp)

Tìm kiếm hai lần: tìm kiếm giá trị

```
std::unordered_map<K, V> m;
[ ... ]
K k1;
auto findIt = m.find(k1);
if (findIt != m.end())
{
    V& v1 = *findIt;            // <--
}
```

Tìm kiếm hai lần: xóa phần tử

```
std::unordered_map<K, V> m;
[ ... ]
K k1;
auto findIt = m.find(k1);
if (findIt ≠ m.end())
{
    m.erase(k1);
}
```

Bảng băm (tiếp)

Tìm kiếm hai lần: xóa phần tử

```
std::unordered_map<K, V> m;
[ ... ]
K k1;
auto findIt = m.find(k1);
if (findIt != m.end())
{
    m.erase(findIt);                    // <--
}
```

Bảng băm (tiếp)

Tìm kiếm hai lần: thêm phần tử

```
std::unordered_map<K, V> m;
[ ... ]
K k1;
auto findIt = m.find(k1);
if (findIt == m.end())
{
    m.insert(k1);
}
```

Bảng băm (tiếp)

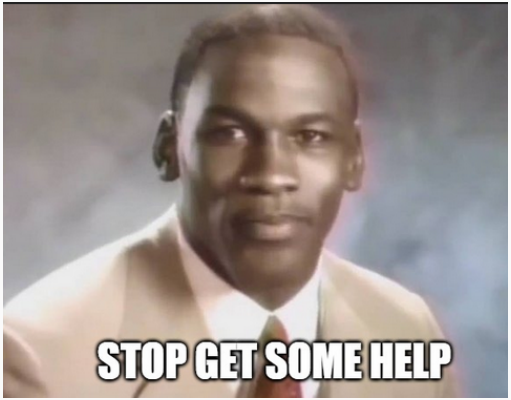
Tìm kiếm hai lần: thêm phần tử

```
std::unordered_map<K, V> m;
[ ... ]
K k1;
auto& [it, inserted] = m.insert(k1);    // <--
if (inserted)
{
    [ ... ]
}
```

Bảng băm (tiếp)

Tìm kiếm trong bảng băm mà không dùng đến giá trị băm

- Tìm kiếm khóa thỏa mãn điều kiện.
- Tìm kiếm theo phần tử.



Bảng băm (tiếp)

Nâng cao: giải quyết xung đột với đánh địa chỉ mở (open addressing):

- Dò tuyến tính/bậc hai (Linear/quadratic probing).
- Cuckoo hash table.
- Robin Hood hash table.

(Có vẻ chưa có giải pháp bên C#)

Thế còn các cấu trúc dữ liệu khác?

- Ngăn xếp (stack).
- Hàng đợi (queue).
  - Cấu trúc vòng (ring buffer).
  - Hàng đợi hai chiều (deque).
- Cây nhị phân (b-tree).

Hỏi đáp