perplexity

Từ A–Z về `std::vector` và `std::map` trong C++

Một cái nhìn ngắn gọn nhưng đầy đủ về hai cấu trúc chứa (container) cơ bản nhất của STL: `std::vector` đại diện cho mảng động tuần tự, còn `std::map` là ánh xạ khóa-giá trị có thứ tự. Bản tổng hợp này trình bày cách dùng, kiến trúc nội bộ, độ phức tạp, "điểm đau" thường gặp và những mẹo tối ưu hoá quan trọng.

Mục lục

Khởi động nhanh

`std::vector<T>` lưu phần tử liên tục trong bộ nhớ, tự động mở rộng dung lượng và đảm bảo toán tử truy cập chỉ số O(1)[1]. `std::map<Key,T>` cất cặp khóa-giá trị trong cây đỏ-đen cân bằng, giữ thứ tự mặc định `std::less<Key>` và bảo đảm mọi thao tác tra cứu, chèn, xoá trong O(log n)[2] [3]. Hai container có triết lý đối lập: vector tối ưu truy cập tuần tự, map tối ưu tìm kiếm có thứ tự.

Phần I — Vector

Tổng quan & cú pháp khai báo

```
#include <vector>
std::vector<int> v1;                // rỗng
std::vector<int> v2(10, -1);        // 10 phần tử -1
std::vector<int> v3 = {1,2,3,4};    // list-init
```

Khởi tạo, chèn (`push_back`, `emplace_back`), truy cập (`operator[]`, `at`), kích cỡ (`size`, `capacity`), xoá (`pop_back`, `erase`) tạo thành API cốt lõi[1].

Bộ nhớ & chiến lược tăng trưởng

1. **Dung lượng (capacity)**: số ô đã cấp phát ≥ `size`[4].
2. **Chiến lược mở rộng**: khi chèn vượt `capacity`, vector xin bộ nhớ mới lớn hơn theo hệ số >1 (libstdc++ thường ×2, MSVC ×1.5)[5]. Việc nhân hệ số bảo đảm `push_back` có **độ phức tạp trung bình O(1)** nhờ phân bố chi phí sao chép[5].
3. **Thu nhỏ**: `shrink_to_fit()` yêu cầu trả lại phần thừa, nhưng KHÔNG bắt buộc trình biên dịch phải giải phóng[6].
4. **Đặt trước**: `reserve(n)` cấp phát một lần duy nhất, giảm reallocations — đặc biệt hữu ích khi biết trước kích thước cuối[7] [8].

Thuộc tính	Ý nghĩa	Hàm kiểm tra	Thao tác thay đổi	Lưu ý
size	Số phần tử hiện tại	`size()`	`resize()`	≤ capacity[4]
capacity	Số ô đã xin	`capacity()`	`reserve`, `shrink_to_fit`	≥ size[4]

Độ phức tạp & invalidation

Hàm	Thời gian	Invalidate iterator?
`operator[]`	O(1)[1]	Không
`push_back`	O(1) trung bình, O(n) khi realloc[5]	Toàn bộ iterator nếu realloc[9]
`insert/erase` giữa	O(n)	iterator từ vị trí đó trở về sau bị hỏng[9]
`clear`	O(n)	Tất cả

Iterators trỏ **trước** vị trí chèn vẫn hợp lệ trừ khi có reallocation[9].

Quản lý ngoại lệ

`at(i)` ném `std::out_of_range` nếu truy cập vượt biên[10]. Các thao tác cấp phát có thể ném `std::bad_alloc`.

Mẹo hiệu năng & cú pháp mới

- C++20 `v.emplace_back(args...)` cho phép dựng thẳng phần tử, tránh copy.
- Dùng vòng `for(auto&& e : v)` để duyệt tối ưu.
- Thêm `v.data()` lấy con trỏ thô tương thích C-API[1].

Phần II — Map

Cấu trúc nội bộ: Cây đỏ-đen

`std::map` áp dụng red-black tree — biến thể cây tìm kiếm tự cân bằng chiều cao O(log n) nhờ quy tắc tô màu nút[11]. Chuẩn yêu cầu mọi phép thao tác (insert, erase, find, lower_bound…) **logarithmic**[3]. Ưu điểm:

- Duy trì thứ tự khóa ⇒ dễ duyệt theo thứ tự tăng dần[12].
- Tìm predecessor/successor nhanh (`lower_bound`, `upper_bound`)[13].

Khai báo & API cốt lõi

```
#include <map>
std::map<std::string,int> m;
m["apple"]  = 3;         // chèn/cập nhật
m.emplace("banana",2);   // C++11
auto it = m.find("apple");
m.erase(it);             // xoá theo iterator
```

Các hàm nổi bật:

Hàm	Ý nghĩa	Độ phức tạp	Ghi chú
`find(key)`	trả iterator	O(log n)[14]	
`insert(pair)`	chèn	O(log n)[14]	thất bại nếu khóa trùng
`operator[]`	truy cập / chèn mặc định	O(log n)[12]	
`lower_bound(k)`	≥k đầu tiên	O(log n)[13]	
`erase(key)`	xoá theo khóa	O(log n)[15]	

Độ phức tạp duyệt

Duyệt toàn bộ `map` O(n). Dù tăng/giảm iterator đơn lẻ tệ nhất O(log n), tổng thể vòng lặp nguyên container vẫn O(n) nhờ tính "cân bằng trượt"[16].

So sánh `map` vs `unordered_map`

Tiêu chí	map	unordered_map
Cấu trúc	Cây đỏ-đen [11]	Bảng băm
Thứ tự	Có	Không
Tìm kiếm	O(log n) [3]	O(1) trung bình, O(n) worst [17]
Bộ nhớ	Thêm bit màu, node pointer [18]	overload factor & bucket mảng
Trường hợp dùng	Cần thứ tự, duyệt theo khoảng, predecessor	Chỉ cần tra cứu khóa cực nhanh

Tuỳ biến bộ so sánh & allocator

```
struct Desc {
  bool operator()(int a,int b) const { return a>b; }
};
std::map<int,std::string,Desc> descMap;
```

Cho phép lưu key theo thứ tự giảm. Tất cả phép so sánh, duyệt sẽ đảo chiều [14].

Hint insert

Nếu biết trước vị trí, `insert(it_hint, pair)` có độ phức tạp **O(1)** amortized khi hint đúng, giảm xuống log n khi sai [14].

Phần III — So sánh tổng hợp & thực chiến

Bảng độ phức tạp chính

Thao tác	Vector	Map
Truy cập theo chỉ số	O(1) [1]	Không hỗ trợ
Tìm kiếm theo giá trị	O(n)	O(log n) [3]
Thêm cuối	O(1) avg [5]	—
Thêm khóa bất kỳ	—	O(log n) [3]
Xoá tùy vị trí	O(n)	O(log n) [15]
Duyệt toàn container	O(n)	O(n) [16]

Khi nào chọn vector?

- Dữ liệu đọc nhiều, thêm cuối, ít xoá giữa.
- Cần mảng liên tục để giao tiếp API C.
- Bộ nhớ chặt chẽ, tránh overhead node.

Khi nào chọn map?

- Tra cứu, chèn/xoá khóa bất kỳ trong tập lớn cần hiệu suất ổn định.
- Cần dữ liệu có thứ tự để xuất báo cáo, tìm range.

Những sai lầm phổ biến & cách tránh

Sai lầm	Hậu quả	Cách khắc phục
Sửa vector khi loop bằng iterator	iterator invalidation, crash [9]	lặp bằng chỉ số hoặc sao lưu size trước
Quên `reserve` khi build list lớn	realloc + copy tốn CPU [7]	Ước lượng kích thước & `reserve(n)`
Dùng `operator[]` với khóa không tồn tại trong map	Tự động chèn phần tử rỗng, sai logic [12]	Dùng `find`/`contains` kiểm tra trước
Lạm dụng `unordered_map` cho khóa có hash xấu	suy thoái O(n) [17]	Tùy biến hàm băm hoặc chuyển sang map

Kết luận

`std::vector` và `std::map` là hai trụ cột không thể thiếu trong lập trình C++. Chúng giải quyết hai nhu cầu khác nhau: lưu trữ tuyến tính hiệu quả và ánh xạ có thứ tự cân bằng. Khi hiểu sâu về nội tại bộ nhớ, độ phức tạp và quy tắc invalidation, lập trình viên có thể khai thác tối đa sức mạnh, tránh các bẫy hiệu năng và viết mã sạch, an toàn hơn.

> "Chọn đúng container ngay từ đầu chính là tối ưu hoá tốt nhất."—Kinh nghiệm từ cộng đồng C++

Điều cốt lõi là: **vector cho tốc độ truy cập và cache locality, map cho tính logic sắp xếp và bảo đảm logarit.** Với kiến thức tổng hợp ở trên, bạn đã có hành trang đầy đủ để ra quyết định sáng suốt trong mọi dự án C++ hiện đại.

⁂

1. https://cplusplus.com/reference/vector/vector/
2. https://www.geeksforgeeks.org/cpp/map-associative-containers-the-c-standard-template-library-stl/
3. https://stackoverflow.com/questions/21740893/what-is-the-time-complexity-of-stdmap
4. https://www.geeksforgeeks.org/cpp/difference-between-size-and-capacity-of-a-vector-in-cpp-stl/
5. https://stackoverflow.com/questions/5232198/how-does-the-capacity-of-stdvector-grow-automatically-what-is-the-rate
6. https://www.educative.io/answers/what-is-the-vectorshrinktofit-function-in-cpp
7. https://stackoverflow.com/questions/6525650/what-are-the-benefits-of-using-reserve-in-a-vector
8. https://www.geeksforgeeks.org/cpp/using-stdvectorreserve-whenever-possible/
9. https://www.geeksforgeeks.org/cpp/iterator-invalidation-cpp/
10. https://stackoverflow.com/questions/23082511/c-vector-class-throwing-exceptions

11. https://en.wikipedia.org/wiki/Red–black_tree
12. https://cplusplus.com/reference/map/map/
13. https://www.geeksforgeeks.org/cpp/map-lower_bound-function-in-c-stl/
14. https://learn.microsoft.com/en-us/cpp/standard-library/map-class?view=msvc-170
15. https://www.geeksforgeeks.org/cpp/map-erase-function-in-c-stl/
16. https://groups.google.com/g/comp.lang.c++.moderated/c/3CDTxV-wbcs
17. https://www.geeksforgeeks.org/map-vs-unordered_map-c/
18. https://www.designgurus.io/answers/detail/why-is-stdmap-implemented-as-a-red-black-tree-in-c