第八章 非变异算法总结与分析

1. 什么是非变异算法

非变异算法是指那些不会改变容器内容的算法。这些算法主要用于查询、计数和比较等操作，确保在操作过程中容器的状态保持不变。非变异算法的优势在于它们可以安全地在不改变数据的情况下进行数据处理。

2. 四类函数讲解

以下是根据您提供的函数列表进行的详细讲解，包括每个函数的功能、代码实例和运行结果。

1. 循环函数

`for_each`

- **功能**: 遍历容器中的每个元素，并对每个元素执行指定的操作。

- **原型**:

```
template<class InputIt, class UnaryFunc>
UnaryFunc for_each(InputIt first, InputIt last, UnaryFunc f);
```

- **示例代码**:

```
#include <iostream>
#include <vector>
#include <algorithm>

void print(int n) {
    std::cout << n << " ";
}

int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5};
    std::for_each(vec.begin(), vec.end(), print);
    return 0;
}
```

- **运行结果**:

```
1 2 3 4 5
```

2. 查询函数

`find`

- **功能**: 在指定范围内查找第一个等于给定值的元素。

- **原型**:

```
template<class InputIt, class T>
InputIt find(InputIt first, InputIt last, const T& value);
```

- **示例代码**:

```
#include <iostream>
#include <vector>
#include <algorithm>
```

```
int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5};
    auto it = std::find(vec.begin(), vec.end(), 3);
    if (it != vec.end()) {
        std::cout << "Found: " << *it << std::endl; // 输出: Found: 3
    }
    return 0;
}
```

- **运行结果**:

```
Found: 3
```

find_if

- **功能**: 在指定范围内查找第一个满足指定条件的元素。
- **原型**:

```
template<class InputIt, class UnaryPredicate>
InputIt find_if(InputIt first, InputIt last, UnaryPredicate p);
```

- **示例代码**:

```
#include <iostream>
#include <vector>
#include <algorithm>

bool is_even(int n) {
    return n % 2 == 0;
}

int main() {
    std::vector<int> vec = {1, 3, 5, 4, 2};
    auto it = std::find_if(vec.begin(), vec.end(), is_even);
    if (it != vec.end()) {
        std::cout << "First even number: " << *it << std::endl; // 输出: First
even number: 4
    }
    return 0;
}
```

- **运行结果**:

```
First even number: 4
```

find_first_of

- **功能**: 查找在范围内第一个出现在另一个范围中的元素。
- **原型**:

```
template<class InputIt1, class InputIt2>
InputIt1 find_first_of(InputIt1 first1, InputIt1 last1, InputIt2 first2,
InputIt2 last2);
```

- **示例代码**:

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> vec1 = {1, 2, 3, 4, 5};
    std::vector<int> vec2 = {3, 6, 7};

    auto it = std::find_first_of(vec1.begin(), vec1.end(), vec2.begin(),
vec2.end());
    if (it != vec1.end()) {
        std::cout << "First match: " << *it << std::endl; // 输出: First
match: 3
    }
    return 0;
}
```

- **运行结果**:

```
First match: 3
```

adjacent_find

- **功能**: 查找在范围内第一个相邻的相等元素。
- **原型**:

```
template<class ForwardIt>
ForwardIt adjacent_find(ForwardIt first, ForwardIt last);
```

- **示例代码**:

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> vec = {1, 2, 2, 4, 5};
    auto it = std::adjacent_find(vec.begin(), vec.end());
    if (it != vec.end()) {
        std::cout << "First adjacent equal: " << *it << std::endl; // 输出:
First adjacent equal: 2
    }
    return 0;
}
```

- **运行结果**:

```
First adjacent equal: 2
```

find_end

- **功能**: 在一个序列中查找另一个序列最后一次出现的位置。

- **原型**:

```
template<class ForwardIt1, class ForwardIt2>
ForwardIt1 find_end(ForwardIt1 first1, ForwardIt1 last1, ForwardIt2 first2,
ForwardIt2 last2);
```

- **示例代码**:

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> vec1 = {1, 2, 3, 2, 3};
    std::vector<int> vec2 = {2, 3};
```

```
    auto it = std::find_end(vec1.begin(), vec1.end(), vec2.begin(),
vec2.end());
    if (it != vec1.end()) {
        std::cout << "Last occurrence found at: " << *it << std::endl; // 输
出: Last occurrence found at: 3
    }
    return 0;
}
```

- **运行结果**:

```
Last occurrence found at: 3
```

search

- **功能**: 查找一个序列中是否存在另一序列。
- **原型**:

```
template<class ForwardIt1, class ForwardIt2>
ForwardIt1 search(ForwardIt1 first1, ForwardIt1 last1, ForwardIt2 first2,
ForwardIt2 last2);
```

- **示例代码**:

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> vec1 = {1, 2, 3, 4, 5};
    std::vector<int> vec2 = {3, 4};

    auto it = std::search(vec1.begin(), vec1.end(), vec2.begin(), vec2.end());
    if (it != vec1.end()) {
        std::cout << "Found sequence starting at: " << *it << std::endl; // 输
出: Found sequence starting at: 3
    }
    return 0;
}
```

- **运行结果**:

```
Found sequence starting at: 3
```

search_n

- **功能**: 在一个序列中查找连续的 n 个相同的元素。
- **原型**:

```
template<class ForwardIt, class T>
ForwardIt search_n(ForwardIt first, ForwardIt last, size_t count, const T&
value);
```

- **示例代码**:

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> vec = {1, 2, 3, 3, 3, 4, 5};
    auto it = std::search_n(vec.begin(), vec.end(), 3, 3);
    if (it != vec.end()) {
        std::cout << "Found 3 in a row starting at: " << *it << std::endl; //
输出: Found 3 in a row starting at: 3
    }
    return 0;
}
```

- **运行结果**:

```
Found 3 in a row starting at: 3
```

3. 计数函数

count

- **功能**: 统计容器中等于指定值的元素的次数。
- **原型**:

```
template<class InputIt, class T>
size_t count(InputIt first, InputIt last, const T& value);
```

- **示例代码**:

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> vec = {1, 2, 2, 3, 4, 2};
    size_t countOfTwos = std::count(vec.begin(), vec.end(), 2);
    std::cout << "Count of 2s: " << countOfTwos << std::endl; // 输出: Count
of 2s: 3
    return 0;
}
```

- **运行结果**:

```
Count of 2s: 3
```

count_if

- **功能**: 统计满足指定条件的元素的次数。
- **原型**:

```
template<class InputIt, class UnaryPredicate>
size_t count_if(InputIt first, InputIt last, UnaryPredicate p);
```

- **示例代码**:

```
#include <iostream>
#include <vector>
#include <algorithm>

bool is_even(int n) {
    return n % 2 == 0;
}

int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5};
```

```
    size_t evenCount = std::count_if(vec.begin(), vec.end(), is_even);
    std::cout << "Count of even numbers: " << evenCount << std::endl; // 输出:
Count of even numbers: 2
    return 0;
}
```

- **运行结果**:

```
Count of even numbers: 2
```

4. 比较函数

equal

- **功能**: 比较两个序列是否相同。
- **原型**:

```
template<class InputIt1, class InputIt2>
bool equal(InputIt1 first1, InputIt1 last1, InputIt2 first2);
```

- **示例代码**:

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> vec1 = {1, 2, 3};
    std::vector<int> vec2 = {1, 2, 3};

    bool areEqual = std::equal(vec1.begin(), vec1.end(), vec2.begin());
    std::cout << "Are the two vectors equal? " << (areEqual ? "Yes" : "No") <<
std::endl; // 输出: Yes
    return 0;
}
```

- **运行结果**:

```
Are the two vectors equal? Yes
```

- **功能**: 找到两个序列中第一个不匹配的元素。
- **原型**:

```
template<class InputIt1, class InputIt2>
std::pair<InputIt1, InputIt2> mismatch(InputIt1 first1, InputIt1 last1,
InputIt2 first2);
```

- **示例代码**:

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    std::vector<int> vec1 = {1, 2, 3, 4};
    std::vector<int> vec2 = {1, 2, 0, 4}; // 0 是不匹配的

    auto result = std::mismatch(vec1.begin(), vec1.end(), vec2.begin());
    if (result.first != vec1.end()) {
        std::cout << "First mismatch: "
                  << *(result.first) << " and " << *(result.second) <<
std::endl; // 输出: 3 and 0
    }
    return 0;
}
```

- **运行结果**:

```
First mismatch: 3 and 0
```