Динамическое программирование

education

Основы динамического программирования

Динамическое программирование (ДП) — это метод решения задач, который позволяет разбивать сложные задачи на множество маленьких подзадач.

Процесс разбиения задач на совокупность более мелких называется декомпозицией.

Принципы ДП

1

Разбиение задачи на подзадачи

Разбиение основной задачи на несколько подзадач и решение каждой из них.

2

Принцип оптимальности

Оптимальное решение задачи зависит от оптимальных решений её подзадач.

3

Мемоизоляция

Сохранение результатов решённых подзадач, чтобы не решать их повторно.

Приёмы

- Топологическая сортировка
- Техника «скользящего окна»
- Стратегия «разделяй и властвуй»
- Префиксные и суффиксные суммы
- Структуры данных для ДП
- Оптимизация пространства
- Состояние и переход
- Профилирование и оптимизация

Приёмы

1

Топологическая сортировка

Позволяет упорядочивать подзадачи таким образом, чтобы гарантировать, что все зависимости для данной подзадачи уже были решены.

2

Техника «скользящего окна»

Скользящее окно предполагает хранение только «окна» последних результатов, что может экономить память.

3

Стратегия «разделяй и властвуй»

Рекурсивное разделение подзадач на меньшие части до тех пор, пока они не станут простыми для решения. Этот метод может сочетаться с ДП для эффективного решения.

4

Префиксные и суффиксные суммы

Предварительное вычисление префиксных или суффиксных сумм для массивов или строк, чтобы ускорить вычисления.

Приёмы

1

Структуры данных для ДП

Определённые структуры данных, такие как сегментные деревья или двоичные индексные деревья, могут быть полезны для эффективного обновления и запроса результатов подзадач.

2

Оптимизация пространства

Оптимизация задач ДП, изначально решённых с использованием двумерного массива, для использования одномерного массива или даже нескольких переменных.

3

Состояние и переход

В задачах ДП очень важно ясно определить, что такое «состояние» (например, позиция в строке, оставшийся вес в задаче о рюкзаке и т. д.) и какие «переходы» возможны из одного состояния в другое.

4

Профилирование и оптимизация

Профилирование кода нужно для выявления узких мест и их оптимизации.

Мемоизоляция (Memoization)

Это сохранение результата предыдущего вызова функции в словаре (или другой структуре данных) и чтение из него, когда мы снова выполним точно такой же вызов.

Фибоначчи

```
int fib(int n) {
    if (n == 0 || n == 1) {
        return n;
    }
    return fib(n - 1) + fib(n - 2);
}
```

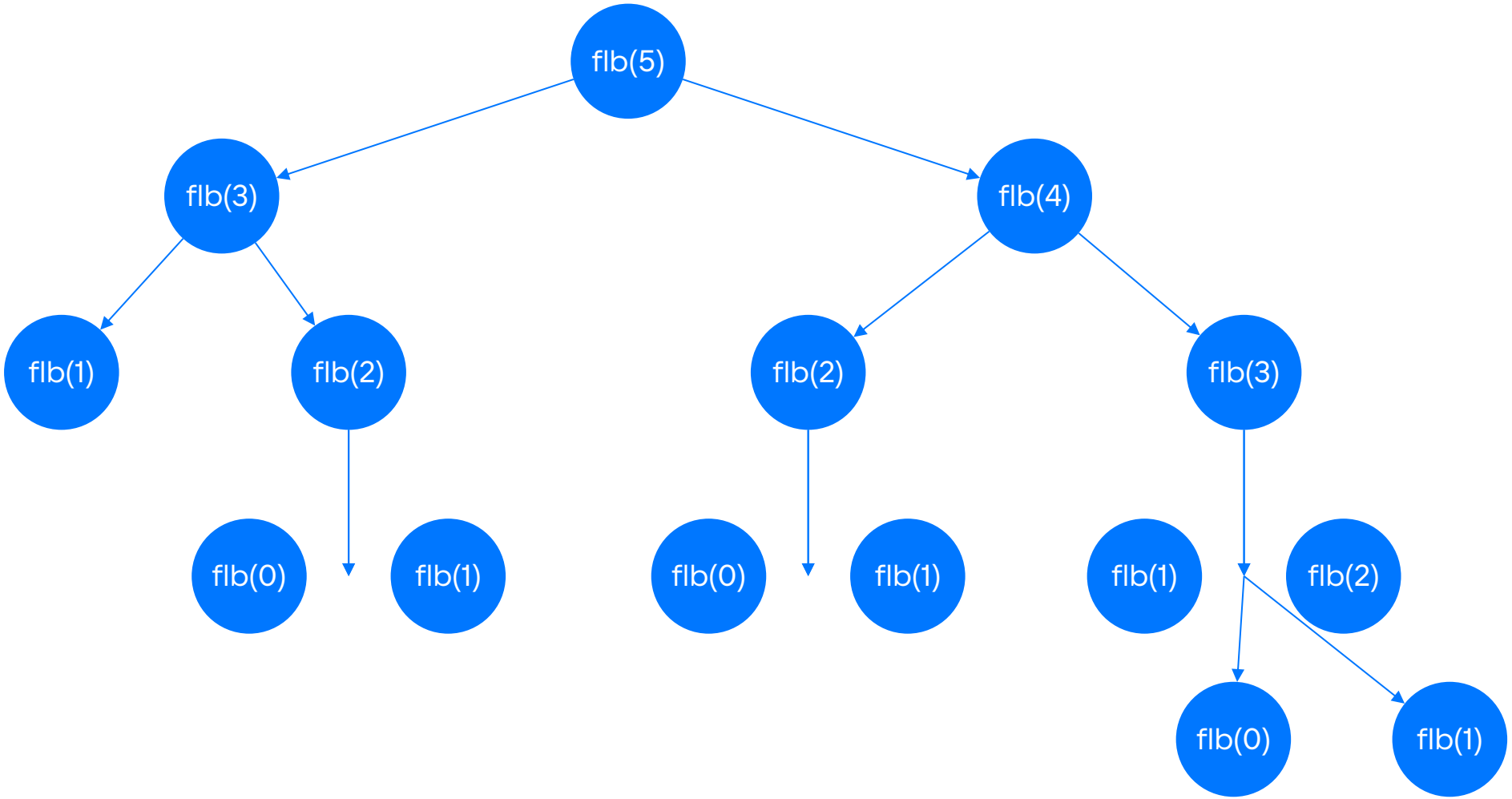
education

Фибоначчи

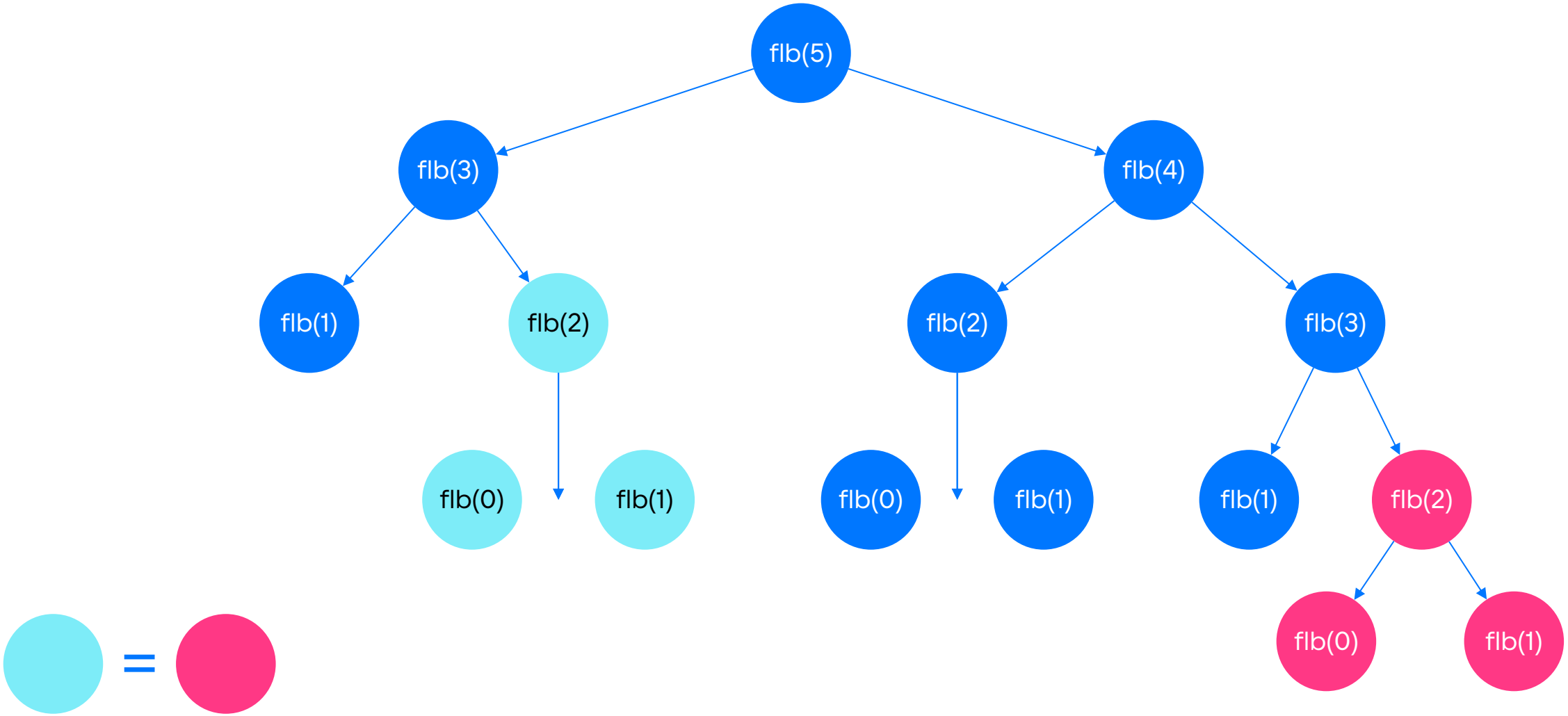
```
int fib(int n) {
    if (n == 0 || n == 1) {
        return n;
    }
    return fib(n - 1) + fib(n - 2);
}
```

```
int fib(int n, int memo[]) {
    // if found in memory – return!
    if (memo[n] != 0) return memo[n];
    if (n == 0 || n == 1) return n;
    int res = fib(n - 1, memo) + fib(n - 2, memo);
    // save it in memory
    memo[n] = res;
    return res;
}
```

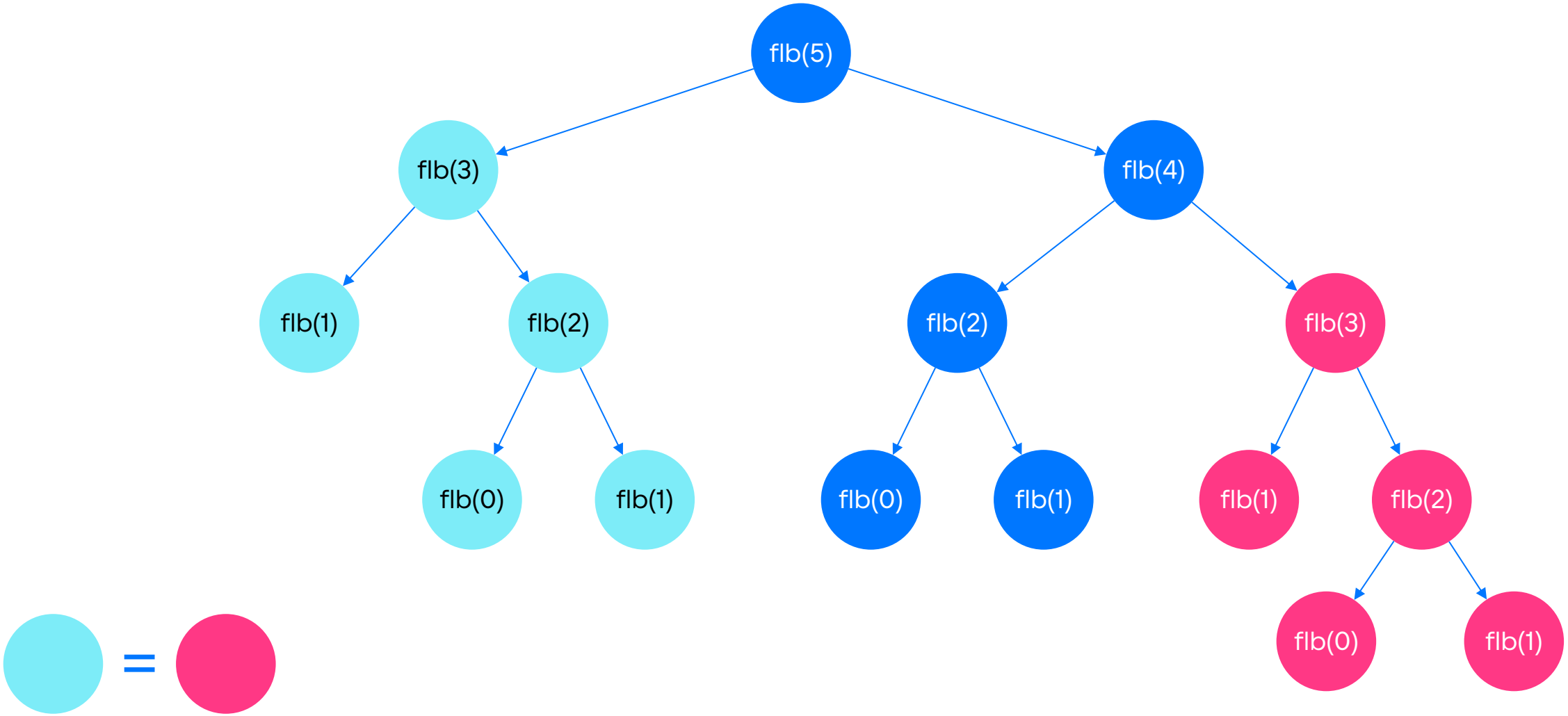
Фибоначчи


flb(5)
flb(3)
flb(4)
flb(1)
flb(2)
flb(2)
flb(3)
flb(0)
flb(1)
flb(0)
flb(1)
flb(1)
flb(2)
flb(0)
flb(1)

Фибоначчи


flb(5)
flb(3)
flb(4)
flb(1)
flb(2)
flb(2)
flb(3)
flb(0)
flb(1)
flb(0)
flb(1)
flb(1)
flb(2)
flb(0)
flb(1)
=

Фибоначчи


flb(5)
flb(3)
flb(4)
flb(1)
flb(2)
flb(2)
flb(3)
flb(0)
flb(1)
flb(0)
flb(1)
flb(1)
flb(2)
flb(0)
flb(1)
=

Фибоначчи

```
int fib(int n, int memo[]) {
    // if found in memory – return!
    if (memo[n] != 0) return memo[n];
    if (n == 0 || n == 1) return n;
    int res = fib(n - 1, memo) + fib(n - 2, memo);
    // save it in memory
    memo[n] = res;
    return res;
}
```

Фибоначчи

```
int fib(int n, int memo[]) {
    // if found in memory – return!
    if (memo[n] != 0) return memo[n];

    if (n == 0 || n == 1) return n;

    int res = fib(n - 1, memo) + fib(n - 2, memo);

    // save it in memory
    memo[n] = res;
    return res;
}
```

Задача

Найдите наибольшую подстроку в строке s, состоящую только из уникальных символов.

```cpp
#include <iostream>
#include <unordered_set>
#include <string>
std::string longestUniqueSubstring(const std::string& s) {
    int left = 0, right = 0, max_len = 0, start = 0;
    std::unordered_set<char> window;
    while (right < s.size()) {
        if (window.find(s[right]) == window.end()) {
            window.insert(s[right]);
            right++;
            if (right - left > max_len) {
                start = left;
                max_len = right - left;
            }
        }
        else {
            window.erase(s[left]);
            left++;
        }
    }

    return s.substr(start, max_len);
}

int main() {
    std::string s;
    std::cin >> s;
    std::cout << longestUniqueSubstring(s) << std::endl;
    return 0;
}
```

education

Задача

Дан массив чисел. Найдите сумму всех элементов между индексами i и j (включительно) за время O(1).

```cpp
#include <iostream>
#include <vector>

class PrefixSum {
private:
    std::vector<int> prefix;
```

education

Задача

Дан массив чисел. Найдите сумму всех элементов между индексами i и j (включительно) за время O(1).

```cpp
#include <iostream>
#include <vector>

class PrefixSum {
private:
    std::vector<int> prefix;

public:
    PrefixSum(const std::vector<int>& nums) {
        prefix.resize(nums.size());
        prefix[0] = nums[0];
        for (int i = 1; i < nums.size(); i++) {
            prefix[i] = prefix[i - 1] + nums[i];
        }
    }
```

Задача

Дан массив чисел. Найдите сумму всех элементов между индексами i и j (включительно) за время O(1).

```cpp
#include <iostream>
#include <vector>

class PrefixSum {
private:
    std::vector<int> prefix;

public:
    PrefixSum(const std::vector<int>& nums) {
        prefix.resize(nums.size());
        prefix[0] = nums[0];
        for (int i = 1; i < nums.size(); i++) {
            prefix[i] = prefix[i - 1] + nums[i];
        }
    }

    int sumRange(int i, int j) {
        if (i == 0) return prefix[j];
        return prefix[j] - prefix[i - 1];
    }
};
```

education

```cpp
int main() {
    int n, i, j;
    std::cin >> n;
    std::vector<int> nums(n);
    for (int& num : nums) {
        std::cin >> num;
    }

    PrefixSum ps(nums);

    std::cin >> i >> j;
    std::cout << ps.sumRange(i, j) << std::endl;
    return 0;
}
```

Будем ВКонтакте!