稀疏表（Sparse Table）是一种高效的数据结构，主要用于解决静态数组上的区间查询问题，特别是最值查询（最大值、最小值等）。它的主要优势在于预处理时间和查询时间都非常高效，适用于数据不变的情况。

稀疏表的基本思路

1. **预处理阶段**：
   - 构建一个二维数组 `st` ，其中 `st[i][j]` 表示从位置 `i` 开始长度为 `2^j` 的区间的最值。
   - 利用动态规划的思想，逐步计算不同长度的区间最值，利用前面计算的结果来推导更长区间的结果。
2. **查询阶段**：
   - 对于任意区间 `[L, R]` ，将其拆分为两个可以重叠的最大幂次长度的区间，然后利用预处理的结果快速查询。

具体步骤

1. 初始化与预处理

假设有一个数组 `arr` ，其长度为 `n` 。首先，初始化一个二维数组 `st` 和一个一维数组 `log` 。 `log` 数组用于快速计算区间长度的对数值。

```
int n = arr.size();
int K = log2(n) + 1;
vector<vector<int>> st(n, vector<int>(K));
vector<int> log(n + 1);

// 初始化 log 数组
log[1] = 0;
for (int i = 2; i <= n; ++i)
    log[i] = log[i / 2] + 1;

// 初始化 st 数组
for (int i = 0; i < n; ++i)
    st[i][0] = arr[i];
```

2. 动态规划预处理

使用动态规划来填充 `st` 数组。 `st[i][j]` 表示从 `i` 开始长度为 `2^j` 的区间的最值。可以通过两个长度为 `2^(j-1)` 的子区间的最值来得到。

```
for (int j = 1; j <= K; ++j) {
    for (int i = 0; (i + (1 << j)) <= n; ++i) {
        st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
    }
}
```

3. 查询阶段

对于任意区间 `[L, R]` ，可以利用预处理的结果快速查询。将 `[L, R]` 拆分为两个可以重叠的长度为 `2^k` 的区间，其中 `k` 为 `log[R - L + 1]` 。

```
int query(int L, int R) {
    int j = log[R - L + 1];
    return min(st[L][j], st[R - (1 << j) + 1][j]);
}
```

代码示例

以下是完整的 C++ 代码示例：

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

class SparseTable {
private:
    vector<vector<int>> st;
    vector<int> log;
    int n;

public:
    SparseTable(const vector<int>& arr) {
        n = arr.size();
        int K = log2(n) + 1;
        st.assign(n, vector<int>(K));
        log.assign(n + 1, 0);

        // 初始化 log 数组
        log[1] = 0;
        for (int i = 2; i <= n; ++i) {
            log[i] = log[i / 2] + 1;
        }

        // 初始化 st 数组
        for (int i = 0; i < n; ++i) {
            st[i][0] = arr[i];
        }

        // 动态规划填充 st 数组
        for (int j = 1; j <= K; ++j) {
            for (int i = 0; (i + (1 << j)) <= n; ++i) {
                st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
            }
        }
    }

    int query(int L, int R) {
        int j = log[R - L + 1];
        return min(st[L][j], st[R - (1 << j) + 1][j]);
    }
};
```

```cpp
int main() {
    vector<int> arr = {1, 3, 5, 7, 9, 11};
    SparseTable sparseTable(arr);
    cout << "Min value in range [1, 4]: " << sparseTable.query(1, 4) << endl;
    return 0;
}
```

总结

- 稀疏表的预处理时间为 $O(n \log n)$，查询时间为 $O(1)$，适合静态数据场景。
- 预处理过程通过动态规划构建不同区间长度的最值，查询通过预处理结果快速得到答案。

这个方法对静态数组的多次区间查询效率非常高，但不适用于需要频繁更新的场景。

好的，下面我们详细讲解稀疏表的动态规划预处理过程。

动态规划预处理详细解释

稀疏表的核心在于利用动态规划的思想预处理数组中的区间最值，从而实现高效的查询。预处理过程中，我们会构建一个二维数组 `st`，其中 `st[i][j]` 表示从索引 `i` 开始长度为 `2^j` 的区间的最小值（当然你也可以用来处理最大值问题，只需要把 `min` 改成 `max`）。

1. 初始化

我们需要初始化一个二维数组 `st`，并且初始化一个 `log` 数组，`log[i]` 表示 `i` 的对数值（向下取整）。

```cpp
int n = arr.size();
int K = log2(n) + 1;  // K 是最大区间长度的对数
vector<vector<int>> st(n, vector<int>(K));
vector<int> log(n + 1);

// 初始化 log 数组
log[1] = 0;
for (int i = 2; i <= n; ++i)
    log[i] = log[i / 2] + 1;
```

在这个过程中，`log[i]` 记录的是 `i` 的对数值（向下取整）。例如，`log[16]` 是 4，因为 $2^4 = 16$，而 `log[10]` 是 3，因为 $2^3 = 8$。

2. 初始化稀疏表的第一列

`st[i][0]` 表示从索引 `i` 开始长度为 $2^0 = 1$ 的区间的最小值，也就是元素本身。

```
for (int i = 0; i < n; ++i)
    st[i][0] = arr[i];
```

3. 动态规划填充稀疏表

为了填充 `st` 数组，我们使用动态规划的方法。对于每个 `j > 0` ，我们用 `st[i][j-1]` 和 `st[i + (1 << (j-1))][j-1]` 来计算 `st[i][j]` 。

```
for (int j = 1; j <= K; ++j) {
    for (int i = 0; (i + (1 << j)) <= n; ++i) {
        st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
    }
}
```

这里的 `st[i][j]` 表示从 `i` 开始长度为 `2^j` 的区间的最小值。我们把长度为 `2^j` 的区间分成两半，每一半长度为 `2^(j-1)` ，然后分别计算这两半的最小值，并取这两个最小值中的较小者。

例子说明

假设我们有数组 `arr = [1, 3, 5, 7, 9, 11]` 。

1. 初始化 `st` 的第一列：

   ```
   st[i][0] = arr[i]
   ```

   得到：

   ```
   st[0][0] = 1
   st[1][0] = 3
   st[2][0] = 5
   st[3][0] = 7
   st[4][0] = 9
   st[5][0] = 11
   ```

2. 计算 `st[i][1]` ，区间长度为 `2^1 = 2` ：

```
st[0][1] = min(st[0][0], st[1][0]) = min(1, 3) = 1
st[1][1] = min(st[1][0], st[2][0]) = min(3, 5) = 3
st[2][1] = min(st[2][0], st[3][0]) = min(5, 7) = 5
st[3][1] = min(st[3][0], st[4][0]) = min(7, 9) = 7
st[4][1] = min(st[4][0], st[5][0]) = min(9, 11) = 9
```

3. 计算 `st[i][2]` ，区间长度为 `2^2 = 4` ：

```
st[0][2] = min(st[0][1], st[2][1]) = min(1, 5) = 1
st[1][2] = min(st[1][1], st[3][1]) = min(3, 7) = 3
st[2][2] = min(st[2][1], st[4][1]) = min(5, 9) = 5
```

注意：此时 `i + 2^2` 必须小于等于 `n` ，所以我们只计算到 `st[2][2]` 。

查询过程

预处理完成后，我们可以在 $O(1)$ 时间内完成区间最值查询。具体方法是，对于查询区间 `[L, R]` ，找到 `k = log[R - L + 1]` ，然后查询 `st[L][k]` 和 `st[R - (1 << k) + 1][k]` 的最小值。

```
int query(int L, int R) {
    int j = log[R - L + 1];
    return min(st[L][j], st[R - (1 << j) + 1][j]);
}
```

完整代码示例

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

class SparseTable {
private:
    vector<vector<int>> st;
    vector<int> log;
    int n;

public:
    SparseTable(const vector<int>& arr) {
        n = arr.size();
        int K = log2(n) + 1;
        st.assign(n, vector<int>(K));
        log.assign(n + 1, 0);

        // 初始化 log 数组
        log[1] = 0;
        for (int i = 2; i <= n; ++i) {
            log[i] = log[i / 2] + 1;
        }

        // 初始化 st 数组
        for (int i = 0; i < n; ++i) {
            st[i][0] = arr[i];
        }

        // 动态规划填充 st 数组
        for (int j = 1; j <= K; ++j) {
            for (int i = 0; (i + (1 << j)) <= n; ++i) {
                st[i][j] = min(st[i][j - 1], st[i + (1 << (j - 1))][j - 1]);
            }
        }
    }

    int query(int L, int R) {
        int j = log[R - L + 1];
        return min(st[L][j], st[R - (1 << j) + 1][j]);
    }
```

```
};

int main() {
    vector<int> arr = {1, 3, 5, 7, 9, 11};
    SparseTable sparseTable(arr);
    cout << "Min value in range [1, 4]: " << sparseTable.query(1, 4) << endl;
    return 0;
}
```

这个例子中， `SparseTable` 类在构造函数中完成了预处理，并且提供了 `query` 方法用于快速查询任意区间的最小值。通过这种预处理，我们可以在 $O(1)$ 时间内完成区间查询，非常高效。