Last update: April 7, 2023 Original

Sparse Table

Sparse Table is a data structure, that allows answering range queries. It can answer most range queries in $O(\log n)$, but its true power is answering range minimum queries (or equivalent range maximum queries). For those queries it can compute the answer in $O(1)$ time.

The only drawback of this data structure is, that it can only be used on *immutable* arrays. This means, that the array cannot be changed between two queries. If any element in the array changes, the complete data structure has to be recomputed.

Intuition

Any non-negative number can be uniquely represented as a sum of decreasing powers of two. This is just a variant of the binary representation of a number. E.g. $13 = (1101)_2 = 8 + 4 + 1$. For a number x there can be at most $\lceil \log_2 x \rceil$ summands.

By the same reasoning any interval can be uniquely represented as a union of intervals with lengths that are decreasing powers of two. E.g. $[2, 14] = [2, 9] \cup [10, 13] \cup [14, 14]$, where the complete interval has length 13, and the individual intervals have the lengths 8, 4 and 1 respectively. And also here the union consists of at most $\lceil \log_2(\text{length of interval}) \rceil$ many intervals.

The main idea behind Sparse Tables is to precompute all answers for range queries with power of two length. Afterwards a different range query can be answered by splitting the range into ranges with power of two lengths, looking up the precomputed answers, and combining them to receive a complete answer.

Precomputation

We will use a 2-dimensional array for storing the answers to the precomputed queries. $\mathrm{st}[i][j]$ will store the answer for the range $[j, j + 2^i - 1]$ of length 2^i. The size of the 2-dimensional array will be $(K + 1) \times \mathrm{MAXN}$, where MAXN is the biggest possible array length. K has to satisfy $\mathrm{K} \geq \lfloor \log_2 \mathrm{MAXN} \rfloor$, because $2^{\lfloor \log_2 \mathrm{MAXN} \rfloor}$ is the biggest power of two range, that we have to support. For arrays with reasonable length ($\leq 10^7$ elements), $K = 25$ is a good value.

The MAXN dimension is second to allow (cache friendly) consecutive memory accesses.

```
int st[K + 1][MAXN];
```

Because the range $[j, j + 2^i - 1]$ of length 2^i splits nicely into the ranges $[j, j + 2^{i-1} - 1]$ and $[j + 2^{i-1}, j + 2^i - 1]$, both of length 2^{i-1}, we can generate the table efficiently using dynamic programming:

```
std::copy(array.begin(), array.end(), st[0]);

for (int i = 1; i <= K; i++)
    for (int j = 0; j + (1 << i) <= N; j++)
        st[i][j] = f(st[i - 1][j], st[i - 1][j + (1 << (i - 1))]);
```

The function f will depend on the type of query. For range sum queries it will compute the sum, for range minimum queries it will compute the minimum.

The time complexity of the precomputation is $O(\mathrm{N} \log \mathrm{N})$.

Range Sum Queries

For this type of queries, we want to find the sum of all values in a range. Therefore the natural definition of the function f is $f(x, y) = x + y$. We can construct the data structure with:

```
long long st[K + 1][MAXN];

std::copy(array.begin(), array.end(), st[0]);

for (int i = 1; i <= K; i++)
    for (int j = 0; j + (1 << i) <= N; j++)
        st[i][j] = st[i - 1][j] + st[i - 1][j + (1 << (i - 1))];
```

To answer the sum query for the range $[L, R]$, we iterate over all powers of two, starting from the biggest one. As soon as a power of two 2^i is smaller or equal to the length of the range ($= R - L + 1$), we process the first part of range $[L, L + 2^i - 1]$, and continue with the remaining range $[L + 2^i, R]$.

```
long long sum = 0;
for (int i = K; i >= 0; i--) {
    if ((1 << i) <= R - L + 1) {
        sum += st[i][L];
        L += 1 << i;
    }
}
```

Time complexity for a Range Sum Query is $O(K) = O(\log \mathrm{MAXN})$.

Range Minimum Queries (RMQ)

These are the queries where the Sparse Table shines. When computing the minimum of a range, it doesn't matter if we process a value in the range once or twice. Therefore instead of splitting a range into multiple ranges, we can also split the range into only two overlapping ranges with power of two length. E.g. we can split the range $[1, 6]$ into the ranges $[1, 4]$ and $[3, 6]$. The range minimum of $[1, 6]$ is clearly the same as the minimum of the range minimum of $[1, 4]$ and the range minimum of $[3, 6]$. So we can compute the minimum of the range $[L, R]$ with:

$$\min(\mathrm{st}[i][L], \mathrm{st}[i][R - 2^i + 1]) \quad \text{where } i = \log_2(R - L + 1)$$

This requires that we are able to compute $\log_2(R - L + 1)$ fast. You can accomplish that by precomputing all logarithms:

```
int lg[MAXN+1];
lg[1] = 0;
for (int i = 2; i <= MAXN; i++)
    lg[i] = lg[i/2] + 1;
```

Alternatively, log can be computed on the fly in constant space and time:

```
// C++20
#include <bit>
int log2_floor(unsigned long i) {
    return std::bit_width(i) - 1;
}

// pre C++20
int log2_floor(unsigned long long i) {
    return i ? __builtin_clzll(1) - __builtin_clzll(i) : -1;
}
```

This benchmark shows that using `lg` array is slower because of cache misses.

Afterwards we need to precompute the Sparse Table structure. This time we define f with $f(x, y) = \min(x, y)$.

```
int st[K + 1][MAXN];

std::copy(array.begin(), array.end(), st[0]);

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

And the minimum of a range $[L, R]$ can be computed with:

```
int i = lg[R - L + 1];
int minimum = min(st[i][L], st[i][R - (1 << i) + 1]);
```

Time complexity for a Range Minimum Query is $O(1)$.

Similar data structures supporting more types of queries

One of the main weakness of the $O(1)$ approach discussed in the previous section is, that this approach only supports queries of idempotent functions. I.e. it works great for range minimum queries, but it is not possible to answer range sum queries using this approach.

There are similar data structures that can handle any type of associative functions and answer range queries in $O(1)$. One of them is called Disjoint Sparse Table. Another one would be the Sqrt Tree.

Practice Problems

- SPOJ - RMQSQ
- SPOJ - THRBL
- Codechef - MSTICK
- Codechef - SEAD
- Codeforces - CGCDSSQ
- Codeforces - R2D2 and Droid Army
- Codeforces - Maximum of Maximums of Minimums
- SPOJ - Miraculous
- DevSkill - Multiplication Interval (archived)

- Codeforces - Animals and Puzzles
- Codeforces - Trains and Statistics
- SPOJ - Postering
- SPOJ - Negative Score
- SPOJ - A Famous City
- SPOJ - Diferencija
- Codeforces - Turn off the TV
- Codeforces - Map
- Codeforces - Awards for Contestants
- Codeforces - Longest Regular Bracket Sequence
- Codeforces - Array Stabilization (GCD version)

Contributors:

jakobkogler (41.48%) wangwillian0 (18.75%) Irvideckis (15.91%) madhur4127 (7.39%) Morass (7.39%) adamant-pwn (3.4099999999999997%) Vidit-Jain (1.7%) algmyr (1.7%) gvnee (0.57%) SiddharthEEE (0.57%) AjayMaheshwari23 (0.57%) wikku (0.57%)