Sparse Table

Sparse Table is a data structure that answers *static* Range Minimum Query (RMQ). It is recognized for its relatively fast query and short implementation compared to other data structures.

Contents

Introduction

The main idea of Sparse Table is to precompute $RMQ[j, j + 2^i)$ for all pairs (i, j).

EXAMPLE

Build a Sparse Table on array `3 1 5 3 4 7 6 1`

0	1	2	3	4	5	6	7
3	1	5	3	4	7	6	1

0	1	2	3	4	5	6	7

where cell (i, j), $0 \leq i < 4$ and $0 \leq j < 8$, stores $RMQ[j, j + 2^i)$.

To answer $RMQ[l, r)$, we can select two pre-computed data with one starts from l and the other one ends at r such that their combined interval covers interval $[l, r)$.

PROOF

There always exist a pair of precomputed interval such that they cover any range $[l, r)$

Let p be the largest integer such that $l + 2^p \leq r$. Then, let the first and second interval be $[l, l + 2^p)$ and $[r - 2^p, r)$ respectively. If the two intervals don't overlaps, this gives us $r - 2^p > l + 2^p \rightarrow l + 2^{p+1} < r$ and leads to a contradiction to our initial assumption that p is the largest integer such that $l + 2^p \leq r$.

Implementation in C++

The implementation for Sparse Table can be done with simple dynamic programming approach.

Construction

The first row is a copy of the initial array. From the second row onward, we can avoid recalculations by optimally picking two green cells from the previous row to get the desired interval. For example, interval $[l, l + 2^k)$ can be achieved by combining intervals $[l, l + 2^{k-1})$ and $[l + 2^{k-1}, l + 2^k)$.

```
C++
1  void build(int A[MAXN], int ST[LOGN][MAXN], int n) {
2
3      int h = floor(log2(n));
4
5      // base case
6      for (int j = 0; j < n; j++) ST[0][j] = A[j];
7
8      // iterative dynamic programming approach
9      for (int i = 1; i <= h; i++)
10         for (int j = 0; j + (1<<i) <= n; j++)
11             ST[i][j] = min(ST[i-1][j], ST[i-1][j + (1<<(i-1))]);
12 }
```

Query

As proven in the previous section, there always exist a pair of precomputed interval in the same row to achieve our desired interval. The trickier part is to find the value p. A clever method is to observe that $2^p \leq r - l$ and look at the binary representation of $r - l$. p is the position of the most significant bit. For example, `10` returns 1, `1011` returns 3, `11111` returns 4 and `1` returns 0. Fortunately, in C++ there's a built-in function `__builtin_clz()` that returns the number of leading `0`'s until the first `1` bit. Since there are a total of 32 bits for a C++ `int` data type, the desired answer is `31-__builtin_clz(r-l)`.

```
C++
1  int query(int l, int r) {        // query in range [l,r)
2      int p = 31 - __builtin_clz(r-l);
3      return min(ST[p][l], ST[p][r-(1<<p)]);
4  }
```

Analysis of Time and Memory Complexity

Construction

In a macro level, since there are n columns and $\lg n$ rows and each cell takes $O(1)$ time to compute, the overall complexity is $O(n \lg n)$.

Similarly, memory complexity is also $O(n \lg n)$.

Query

We only need two cells for any pairs (l, r), hence complexity is $O(1)$.