Sparse Table

Submitted By

Hetarth Parikh - 19BCE155

Ashray Patel - 19BCE161

Dhvanil Patel - 19BCE165

Submitted To

Professor Anitha Modi

What is a Sparse Table?

A sparse table is a data structure that is used to answer range or interval queries for a given set of numbers.

It is similar to a segment tree or fenwick tree specifically. We can say that a sparse table is a subset of a segment tree but some of the features are more powerful than a segment tree or fenwick tree like it resolves queries in O(1) time only.

It is used to handle operations that have idempotent (x op x = x) properties like we can resolve minimum or maximum or gcd queries because these are all idempotent functions.

Operations :

- Create / Build
- Minimum Query
- Maximum Query

Construction of Sparse Table :

The main idea behind Sparse Tables is to precompute all answers for range queries with a power of two lengths. After that, a different range query can be answered by splitting the range into ranges with the power of two lengths, looking up the precomputed answers, and combining them to receive a complete answer.
To implement a sparse table, we use a 2-dimensional array where st[i][j] is the $F(i,i+1,...,i+2^j-1)$ and can be calculated using previous information as below,

$$st[i][j] = F(st[i][j-1], st[i-1][i+(1<<(j-1)][j-1])$$

Queries On Sparse Table :

Queries are the most crucial part of a sparse table because it resolves queries in O(1). We know that even if the interval overlaps result of the idempotent function doesn't affect so to answer the query we find two segments whose length is the power of two in which the given range is fully contained so using a sparse table we can answer that query as,

$$k = \log2(r - l + 1)$$
$$F(st[l][k],st[r - (1 << k) + 1][k])$$

Code:

```
// Sparse Table
struct Sparse {

  int n, LOG;
  vector<vector<int> > sparse_minimum, sparse_maximum;

  Sparse() {};

  Sparse(int n) {
    this->n = n;
    LOG = log2(n) + 2;
    sparse_minimum = vector<vector<int> > (n, vector<int> (LOG, 0));
    sparse_maximum = vector<vector<int> > (n, vector<int> (LOG, 0));
  }
  void build(vector<int> &a) {
    for (int j = 0; j < LOG; j++) {
```

```
      for (int i = 0; i + (1 << j) - 1 < n; i++) {
        if (j == 0) {
          sparse_minimum[i][j] = a[i];
          sparse_maximum[i][j] = a[i];
        } else {
          sparse_minimum[i][j] = min(sparse_minimum[i][j - 1],
sparse_minimum[i + (1 << (j - 1))][j - 1]);
          sparse_maximum[i][j] = max(sparse_maximum[i][j - 1],
sparse_maximum[i + (1 << (j - 1))][j - 1]);
        }
      }
    }
  }

  int query_minimum(int l, int r) {
    int k = log2(r - l + 1);
    return min(sparse_minimum[l][k], sparse_minimum[r - (1 << k) +
1][k]);
  }

  int query_maximum(int l, int r) {
    int k = log2(r - l + 1);
    return max(sparse_maximum[l][k], sparse_maximum[r - (1 << k) +
1][k]);
  }
};
```

Applications **:**

1. In the database systems, many times we want to find some records in a particular interval whose values are maximum or minimum in that range, for example, we are given a mark sheet of students and we want to find the minimum or maximum in some specific range for analysis purposes.

2. In a system or architecture where we want our query to be solved very fast and the system itself doesn't allow update operation at that place instead of segment tree, sparse table perform very well.

Implementation of Application-1 :

Code :

```
#include<bits/stdc++.h>
using namespace std;
// Sparse Table
struct Sparse {
  int n, LOG;
  vector<vector<int> > sparse_minimum, sparse_maximum;
  Sparse() {};
  Sparse(int n) {
    this->n = n;
    LOG = log2(n) + 2;
    sparse_minimum = vector<vector<int> > (n, vector<int> (LOG, 0));
    sparse_maximum = vector<vector<int> > (n, vector<int> (LOG, 0));
  }
  void build(vector<int> &a) {
    for (int j = 0; j < LOG; j++) {
      for (int i = 0; i + (1 << j) - 1 < n; i++) {
        if (j == 0) {
          sparse_minimum[i][j] = a[i];
          sparse_maximum[i][j] = a[i];
        } else {
          sparse_minimum[i][j] = min(sparse_minimum[i][j - 1],
sparse_minimum[i + (1 << (j - 1))][j - 1]);
          sparse_maximum[i][j] = max(sparse_maximum[i][j - 1],
sparse_maximum[i + (1 << (j - 1))][j - 1]);
        }
      }
    }
  }
```

```
  int query_minimum(int l, int r) {
    int k = log2(r - l + 1);
    return min(sparse_minimum[l][k], sparse_minimum[r - (1 << k) +
1][k]);
  }
  int query_maximum(int l, int r) {
    int k = log2(r - l + 1);
    return max(sparse_maximum[l][k], sparse_maximum[r - (1 << k) +
1][k]);
  }
};
int main() {
  string file_name;
  ifstream fp;
  map<string, int> student_roll_no_to_unique_id;
  vector<int> students_marks;
  int id = 0;
  // Reading File
  do {
    cout << "Enter the file name : ";
    cin >> file_name;
    fp.open(file_name);
    if (!fp) {
      cout << "File doesn't exists\n";
      continue;
    }
    string line;
    while (getline(fp, line)) {
      while ((int)line.size() && line.back() == ' ') line.pop_back();
      string roll_no = "";
      for (char x : line) {
        if (x == ' ') break;
        roll_no.push_back(x);
      }
      reverse(line.begin(), line.end());
      string marks = "";
      for (char x : line) {
        if (x == ' ') break;
```

```
        marks.push_back(x);
      }
      reverse(marks.begin(), marks.end());
      int mark = stoi(marks);
      student_roll_no_to_unique_id[roll_no] = id++;
      students_marks.push_back(mark);
    }
    cout << "File is read successfully\n";
    break;
  } while (true);
  Sparse sp(id);
  sp.build(students_marks);
  do {
    cout << "\nEnter 1. Get Minimum In Range\n";
    cout << "Enter 2. Get Maximum In Range\n";
    cout << "Enter 3. Exit\n";
    int ip;
    cin >> ip;
    if (ip == 1 || ip == 2) {
      cout << "Enter starting roll_no - 1 and ending roll_no - 2 : ";
      string l, r;
      cin >> l >> r;
      int id_l, id_r;
      auto left_itr = student_roll_no_to_unique_id.lower_bound(l);
      auto right_itr = student_roll_no_to_unique_id.upper_bound(r);
      if (left_itr == student_roll_no_to_unique_id.end()) {
        cout << "Roll Numbers are out of range\n";
        continue;
      }
      if (student_roll_no_to_unique_id.upper_bound(r) ==
student_roll_no_to_unique_id.begin()) {
        cout << "Roll Numbers are out of range\n";
        continue;
      }
      right_itr--;
      id_l = (*left_itr).second;
      id_r = (*right_itr).second;
      if (ip == 1) {
```

```
        cout << "Minimum in range is " << sp.query_minimum(id_l,
id_r) << "\n";
      } else {
        cout << "Maximum in range is " << sp.query_maximum(id_l,
id_r) << "\n";
      }
      continue;
    }
    if (ip == 3) {
      break;
    }
    cout << "Invalid Input\n";
  } while (true);
  return 0;
}
```

Input : ADS_Marks.txt

```
19BCE151 51
19BCE153 53
19BCE155 55
19BCE156 56
19BCE157 57
19BCE159 59
19BCE160 60
19BCE162 62
19BCE164 64
19BCE165 65
19BCE166 66
19BCE167 67
19BCE170 70
19BCE172 72
19BCE175 75
19BCE178 78
19BCE180 80
19BCE186 86
19BCE187 87
```

```
19BCE199 99
19BCE213 93
19BCE250 50
19BCE258 58
19BCE262 62
19BCE299 99
```

Execution :

```
D:\Study\B.Tech\SEM 6\ADS\Assignment>g++ Sparse_Table_Assignment.cpp -o Sparse_Table_Assignment

D:\Study\B.Tech\SEM 6\ADS\Assignment>Sparse_Table_Assignment
Enter the file name : xyz
File doesn't exists
Enter the file name : ADS_Marks.txt
File is read successfully

Enter 1. Get Minimum In Range
Enter 2. Get Maximum In Range
Enter 3. Exit
1
Enter starting roll_no - 1 and ending roll_no - 2 : 19BCE001 19BCE100
Roll Numbers are out of range

Enter 1. Get Minimum In Range
Enter 2. Get Maximum In Range
Enter 3. Exit
1
Enter starting roll_no - 1 and ending roll_no - 2 : 19BCE155 19BCE199
Minimum in range is 55

Enter 1. Get Minimum In Range
Enter 2. Get Maximum In Range
Enter 3. Exit
2
Enter starting roll_no - 1 and ending roll_no - 2 : 19BCE155 19BCE300
Maximum in range is 99

Enter 1. Get Minimum In Range
Enter 2. Get Maximum In Range
Enter 3. Exit
3

D:\Study\B.Tech\SEM 6\ADS\Assignment>
```

Thank You!