Readings about Segment Trees

Diego Linares - kiwiAipom

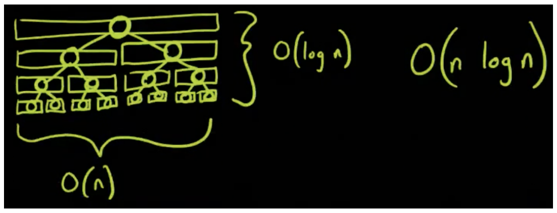
I. Algorithms Live - Segment Trees

More of a way of thinking about a problem, rather than a data structure, much like DP.

Takes the recursion pattern of *MergeSort* and *Divide and Conquer*, and turning it into a data structure.

So you have a `list` and you cut it in half and now you have 2. Do keep doing this until each list is of size 1. Allows for a nice visual representation of why *MergeSort* is $n \lg n$

How do I make it a data structure? Well, you can see each division as a node.



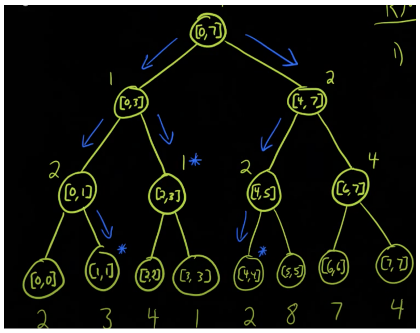
Each leaf node is an individual value. And you can push a function (for example: `min`) upwards through the tree. And I know for each node what particular range is being covered.

A. Range Minimum Queries (RMQ)

Given $[i, j]$ range report the minimum number.

Ex. Take the case of $[2, 7]$. The steps to see:

1) Since $[0, 7]$ is not completely covered by $[2, 7]$ we go to the left child.
2) $[0, 3]$ is also not completely covered by $[2, 7]$ so we go to the left child again.
3) $[0, 1]$ is completely disjoint, so we go back up. And then to the right side.
4) $[2, 3]$ is completely covered by $[2, 7]$, so we can return that value.
5) Unlike a binary search tree, I can go down more than one path, so from the root now I check the right child, and $[4, 7]$ is completely covered, so I return that value.
6) The min of the two will be my answer.



For the **ex.** $[1, 4]$, this time calculate the min of 3 intervals $[1, 1], [2, 3], [4, 4]$, basically, the uppermost segments that are completely covered by the query.

The execution time, well, the depth of the tree is still $\lg n$, but now we are covering more nodes than a regular binary search. But actually, the amount of extra nodes I am covering is a constant factor (at most 1 node for each one that I find), so the query time is still $\lg n$

B. Lazy Propagation

It gives Segment Tree their advantage. Allows to make changes to the sequence by affecting the tree in a ”lazy” way.

Ex. `increase range [i,j] + val`. But still want to do the queries as a tree.

- Now I don’t only store values, but also a change that I want to push down the tree.
- We keep track of it in a δ

II. Simple Approach to Segment Trees 1

III. Simple Approach to Segment Trees 2

IV. Bonus: How to Test your Solution on Competitive Programming

Useful when you are getting `Wrong Answer` and you can’t find a **counter-example**.

First, it is a good idea to write a `brute`, doesn’t matter how slow it is. Try different small tests and compare the results.

You then could use a **generator** for small cases. Be sure that your seed is okay so it produces different values. Then you can generate a script to do it multiple times until it differs. An example is:

```
fo((i=1; i<=100; ++i)) do
  echo $i
  ./gen $i > int
  ./a < int > out1
  ./brute < int > out2
  diff -w out1 out2 || break
done
```

It is recommended that the cases generated are small so the particular pattern is easier to find. If it runs for a while, means that everything works. With `cat int` you are able to print a countertest.

If that still doesn’t work, **check for overflows**, since small cases usually don’t cover them. **Edge cases** that should be generated by yourself. Or your array of declaration might be too small.

Some compilation flags that you might use are the following:

```
g++ -02 std=c++17 -Wno-unused-result -Wshadow
    -Wall -o a a.cpp
g++ -std=c++17 -Wshadow Wall -o a a.cpp -
    fsanitize=address -fsanitize=undefined -
    D_GLIBCXX_DEBUG -g
```

First oneis for faster runninng time and the second one for checking for mistakes.

Remember to change the value of the arrays that you are using to a smaller value for small tests. Use `#warning` to keep track about it.

A. How to generate trees

This is a first script to generate some kind of trees:

```
srand(atoi(argv[1]));
int n = rand(2,20);
printf("%d\n",n);
for(int i = 2; i <= n; ++i)
{
  printf("%d %d\n", rand(1,i-1), i);
}
```

A smarter way that can generate much more kind of trees is the following.

```
vector<pair<int,int>> edges;
for(int i = 2; i <= n; ++i)
  edges.emplace_back(rand(1, i-1),i);

vector<int> perm(n+1); // re-naming vertices
for(int i = 1; i <= n; ++i)
  perm[i] = i;
random_shuffle(perm.begin()+1, perm.end());

random_shuffle(edges.begin(), edges.end()); //
     random order of edges.

for(pair<int,int> edge: edges)
{
  int a = edge.first, b = edge.second;
  if (rand() % 2)
    swap(a,b) // for random order
  printf("%d %d\n",perm[a],perm[b]);
}
```

In the first script, modifying the rand to $(1,1)$ will get you a star, and just putting $i-1$ will get you a line.

Read about proof strings for further info.

References

[1] *How to test your solution in Competitive Programming, on Linux?*, Errichto, From: https://www.youtube.com/watch?v=JXTVOyQpSGM