CPSC 490 – Problem Solving in Computer Science

Lecture 14: Binary Indexed Tree

Jason Chiu and Raunak Kumar

Based on slides by Paul Liu (2014), Kuba Karpierz and Bruno Vacherot (2015), TopCoder Tutorial

2017/02/05

University of British Columbia

Last Time

Last time: Segment Tree for range queries

- Can do online update and query in $O(\log n)$ per query
- Range updates using lazy propagation
- Memory usage: 2 times the size of the input array

Today's goal: Improve memory usage

But we are only restricted to sum queries

Range Query One More Time

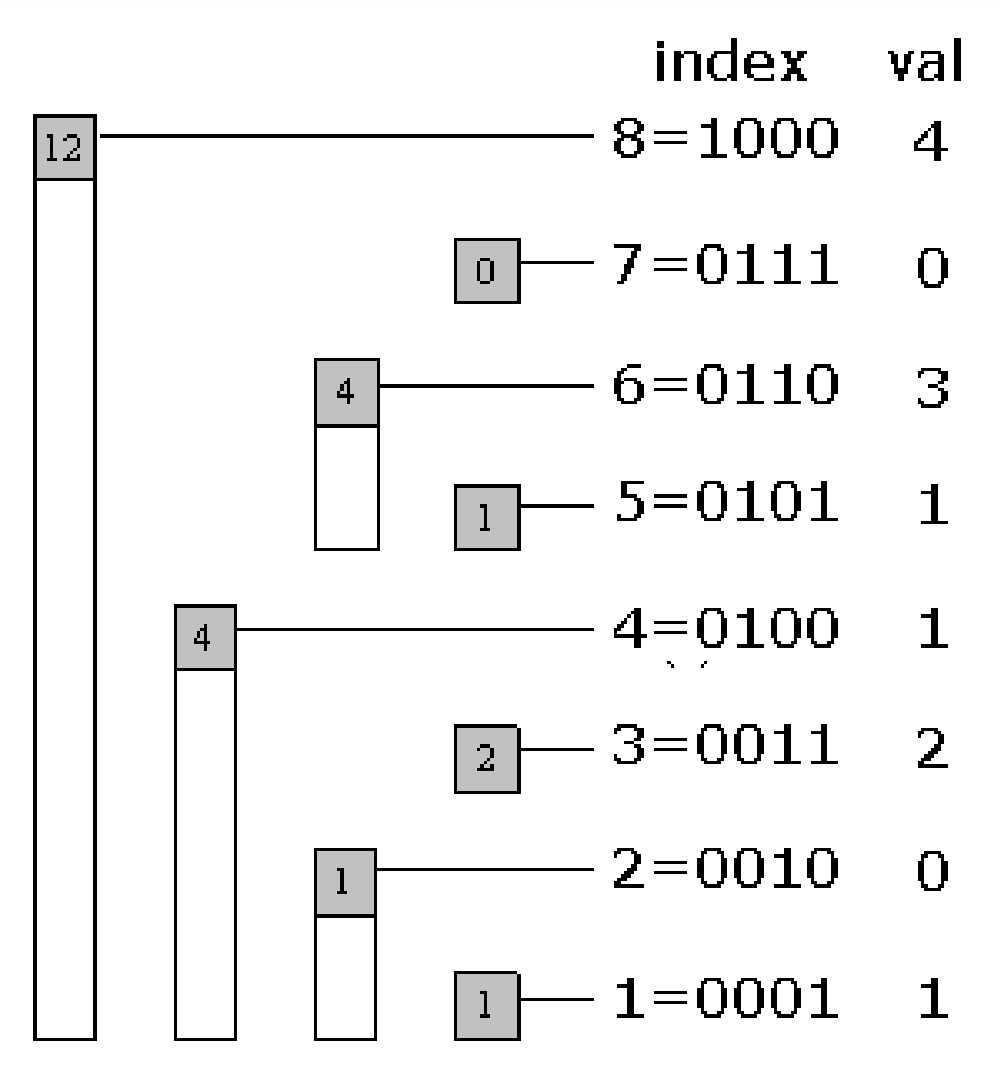
Given an array A of n integers, answer these queries quickly:

- `update(i, x)`: sets `A[i] = x`
- `sum(i, j)`: return `A[i] + ... + A[j]`

We will use **Binary Indexed Tree** or **Fenwick Tree** to solve this in $O(\log n)$ per query, with n storage and it's EASY to code!

Binary Indexed Tree

Idea: take a segment tree and delete every other node in every layer!



BIT Indices

- Let `idx` be an index of the BIT
- Let `r` be the position of the last 1-bit of `idx` (in binary)
  - Example, `idx` = 4 = 100_2, `r` = 2
  - Example, `idx` = 6 = 110_2, `r` = 1
- `bit[idx]` will store the sum for the range $[idx - 2^r + 1, idx]$
  - Example, `bit[4]` stores the sum for $[4 - 2^2 + 1, 4] = [1, 4]$
  - Example, `bit[6]` stores the sum for $[6 - 2^1 + 1, 6] = [5, 6]$
- To get `sum(1, 6)`
  - 6 = 4 + 2
  - We will return `bit[4] + bit[4+2]`

BIT - Initialize

```
1  const int MAXN = 100000;
2  int bit[MAXN+1];
```

BIT - Query

```
1  // Gets the sum of [1 ... x]
2  int query(int x) {
3    int sum = 0;
4    // Keep clearing the last bit of x until it is 0
5    while (x > 0) {
6      sum += bit[x];
7      x -= (x & -x);
8    }
9    return sum;
10 }
```

Time Complexity: number of 1-bits in x = $O(\log n)$.

BIT - Update

```
1  // Adds value val to index x
2  void add(int x, int val) {
3    // Keep adding to the last bit of x until it
4    // exceeds the input array length
5    while (x <= MAXN) {
6      bit[x] += val;
7      x += (x & -x);
8    }
9  }
```

Time Complexity: number of bits left of last 1-bit in x = $O(\log n)$.

Entire BIT

```
1  const int MAXN = 10000000;
2  int bit[MAXN+1];
3
4  void add(int val, int x) {
5    for (; x <= MAXN; x += (x & -x)) bit[x] += val;
6  }
7
8  int query(int x){
9    int sum = 0;
10   for (; x; x -= (x & -x)) sum += bit[x];
11   return sum;
12 }
```

Range Queries One Last Time

Given an **tree** T of n nodes, answer these queries quickly:

- `update(u, x)`: sets `T[v] = x` for all nodes `v` in the subtree rooted at `u`
- `sum(u)`: return sum of all nodes in the subtree rooted at `u`

Range Queries One Last Time

Given an **tree** T of n nodes, answer these queries quickly:

- `update(u, x)`: sets `T[v] = x` for all nodes `v` in the subtree rooted at `u`
- `sum(u)`: return sum of all nodes in the subtree rooted at `u`

We know about different techniques to perform range queries ...

Range Queries One Last Time

Given an **tree** T of n nodes, answer these queries quickly:

- `update(u, x)`: sets `T[v] = x` for all nodes `v` in the subtree rooted at `u`
- `sum(u)`: return sum of all nodes in the subtree rooted at `u`

We know about different techniques to perform range queries ...

But they all work on arrays, not trees!

Range Queries One Last Time

Given an **tree** T of n nodes, answer these queries quickly:

- `update(u, x)`: sets `T[v] = x` for all nodes `v` in the subtree rooted at `u`
- `sum(u)`: return sum of all nodes in the subtree rooted at `u`

We know about different techniques to perform range queries ...

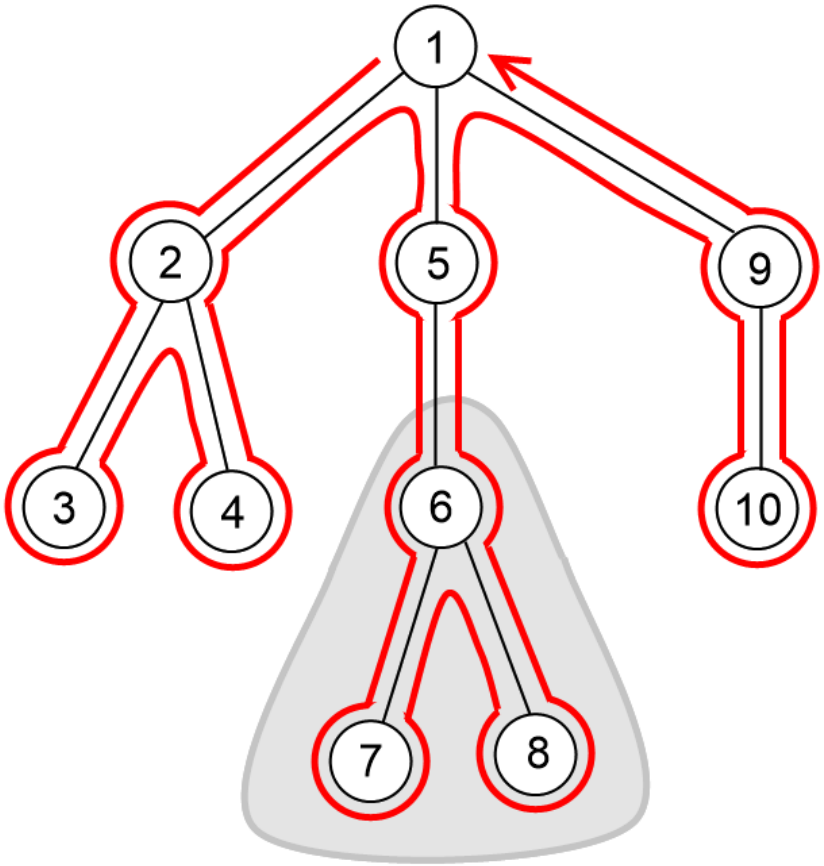
But they all work on arrays, not trees!

Idea: just convert tree into array!

Euler Tour on a Tree I – subtree queries

List nodes in pre-order traversal, then subtree = subarray!

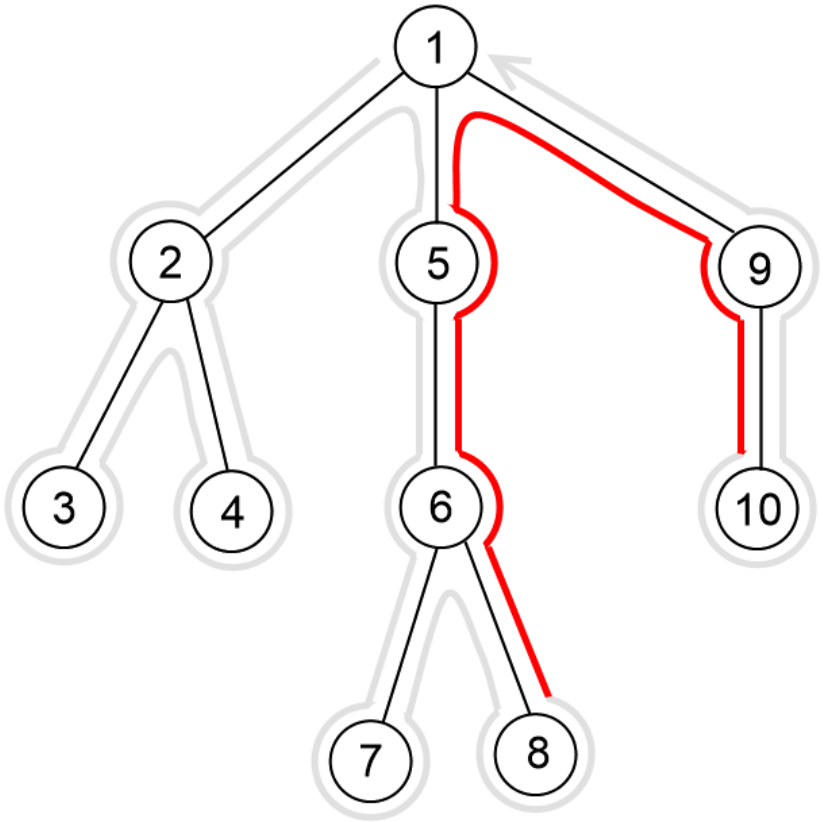
Start = pre-order number, end = post-order number



[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Euler Tour on a Tree II – LCA and path queries

List node every time you enter and exit on a “tour” then LCA is easy!
Path *sum* to root: on a BIT, +x at first(u), -x at last(u), query(first(u))



[1, 2, 3, 3, 2, 4, 4, 2, 1, 5, 6, 7, 7, 6, 8, 8, 6, 5, 1, 9, 10, 10, 9, 1]

Implementation

```
1  int cnt = 0;
2  int tour[N], begin[N], end[N]; bool seen[N];
3  void dfs(int u) {
4    seen[u] = true;
5    begin[u] = cnt;
6    tour[cnt] = u; cnt++;
7    for (int v : adj[u]) {
8      if (!seen[v]) {
9        dfs(v);
10       //tour[cnt] = u; cnt++; // type II tour
11     }
12   }
13   //tour[cnt] = u; cnt++; // type II tour
14   end[u] = cnt;
15 }
```

End of material for Assignment 4

Final word: what you can do with a balanced BST

Everything we did can be done with a balanced binary search tree!

Exercise: figure out how to do these operations in $O(\log n)$ with an AVL tree (or splay tree, red-black tree, treap, etc)

- Find k-th largest element for any k (we saw this in tree DP)
- Simulate an "array" with the following operations
  - Insert/delete/get element at any index
  - Cut an array into two pieces
  - Concat two arrays together
- Use this "array" and Euler Tour Tree to simulate Link/Cut Tree
- Keep a history of all modifications to the tree
- Range query and range update

Caveat: larger constant factors on the $O(\log n)$

Things we did not cover

- 2D/3D Segment Tree, BIT, etc.
- Range query and range update on a BIT
- Heavy Light Decomposition
- Link/Cut Tree
- Square-root Decomposition on a graph

Next Class

Presentations & Homework Help