(Disjoint) Sparse Table
&
簡潔データ構造

KMC競技プログラミング練習会Advanced

2019/05/24 zaki

Sparse Table

- “静的な”数列Aに対して以下のクエリを”O(1)で”処理する
  - 区間[l, r)に対して min{a_l, ..., a_(r-1)} を答える
  - logは定数！ではない(コドフォでは)

- minだけでなく結合則と冪等則を満たす演算が載せられる
  - 冪等則: 演算opが集合Sの任意の元sに対して op(s, s) = s
  - min, max, gcdなど

- これを実現するデータ構造が空間O(NlogN)で作れる

Sparse Table(min query) 構築編

- 数列Aに対し, table[i][j] := 区間[i, i+(1<<j))の 最小値となるようなテーブルを作成する.

- jごとに更新できる
  - table[i][j] = min(table[i][j-1]. Table[i-1][j+(1<<(j-1))])
  - O(nlogn)

Sparse Table(min query) 構築編

- 例
- A = [2, 4, 5, 3, 8, 1, 7, 6]
- table[i][0] = A[i] (初期化)
- table[0][1] = min(table[0][0], table[1][0]) = 2...
- table[0][2] = min(table[0][1], table[2][1]) = 2...
- table[0][3] = min(table[0][2], table[4][2]) = 1...

Sparse Table(min query) 解答編

- 数列Aの区間 [l, r) に対して最小値をO(1)で答える

- 冪等則を仮定しているので, 区間の半分以上を覆うような区間を2つ持ってきてマージすればよい

- 2^b <= r – l < 2^(b+1) となるようなiを持ってくると, min(table[l][b], table[r-(1<<b)][b]) が答え.
- このようなbは前計算できる(builtin 関数でもok)

Sparse Table(min query) 解答編

- 例
- A = [2,4,5,3,8,1,7,6] に対してクエリを処理する(0-indexed)

- Query([1, 4)) = min(table[1][1], table[2][1])

- Query([3, 8)) = min(table[3][2], table[4][2])

- Query([0, 4)) = min(table[0][1], table[2][1])

Sparse Table

- 実装例
  - (添字がスライドと入れ替えてある)
  - https://github.com/zaki-joho/ProconLibrary/blob/master/Structure/sparse_table.hpp

Disjoint Sparse Table

- 区間sumが欲しいですね(セグ木にできることはできてほしい)
- Add(1, 1) != 1 より冪等則は満たされない

- 結合則を満たす演算(半群)が載せられるのがDisjoint S.T.
- 単位元は必要ない

- Sparse Table の上位互換(多分)
- 静的な数列に対してはセグ木を置換えられる
- 各二冪の位置を始点にして累積和を取るイメージ

Disjoint Sparse Table(sum query)構築編

- 数列Aに対し, table[i][j] := (1<<i) 個ごとに区切った時の, (いい感じの基準点から) A_j までの累積和 を前計算

- いい感じは例と実装から読み取ってください…

Disjoint Sparse Table(sum query)構築編

- 例
- A = [2, 4, 5, 3, 8, 1, 7, 6]

- table[0][i] = A[i]
- table[2][3] = 3, table[2][2] = 8, table[2][1] = 12, table[2][0] = 14
- table[1][1] = 4, table[1][0] = 6, table[1][2] = 5, table[1][3] = 8

Disjoint Sparse Table(sum query)構築編

- いい感じの図

	i=0	1	2	3	4	5	6	7
A[i]	2	4	5	3	8	1	7	6
table[1]	6	4	5	8	9	1	7	13
table[2]	14	12	8	3	8	9	16	22

Disjoint Sparse Table(sum query)解答編

- 数列Aの区間 [l, r] (閉区間！) に対して最小値をO(1)で答える

- [l, r] がまたぐ基準点の内, 一番広い範囲を覆うものを見つけてそこから左右への累積和をマージする

- 基準点は (l XOR r) の一番上のbit位置で決まる

31-__builin_clz(l ^ r) 段目を見る

Disjoint Sparse Table(sum query)解答編

- 例
- A = [2,4,5,3,8,1,7,6] に対してクエリを処理する(0-indexed)

- Query([0,4]) = Query([000, 100]) = table[2][0] + table[2][4]

- Query([1,3]) = Query([001, 011]) = table[1][1] + table[1][3]

- Query([2,2]) = Query([010, 010]) = table[0][2] (例外処理)

Disjoint Sparse Table

- 実装例
- こっちでは __builtin_clz() を使った
- [l, r) でクエリを投げ, 内部的には閉区間で処理する
- https://github.com/zaki-joho/ProconLibrary/blob/master/Structure/disjoint_sparse_table.hpp

- log は定数ではない例
  - https://codeforces.com/contest/1167/submission/54203015
  - https://atcoder.jp/contests/iroha2019-day1/submissions/5326237

LCA/RMQ と簡潔データ構造

- 目次

- 1. 簡潔データ構造とは

- 2. Cartesian Tree, Euler Tour

- 3. LCA, RMQ は空間O(N), 時間O(1) (クエリ当たり)で解ける
  - ↑を処理する過程で Sparse Table を使う
  - 以降では <O(N), O(1)> といった表記を使う

LCA/RMQ と簡潔データ構造

- 簡潔データ構造
  - 簡潔表現と簡潔索引から成る
  - 簡潔表現: 集合U に対する簡潔表現は $\forall x \in U$に対し, x を表すビット列の長さが log|U| + o(log|U|) であるようなもの
    （このような表現は必ず存在する, 証明は読者への課題とする）
  - 簡潔索引: データ構造のビット数が o(log|U|) であるものの内, クエリを簡潔でないデータ構造と同じ計算量で処理できるもの
    - （ここでの”簡潔でないデータ構造”はO(nlogn)ぐらいが想定されていそう）
  - つまり, 漸近的に元のデータを保持するのと同じ領域でクエリを処理できるようなデータ構造
    - Wavelet Matrix とかが有名?

LCA/RMQ と簡潔データ構造

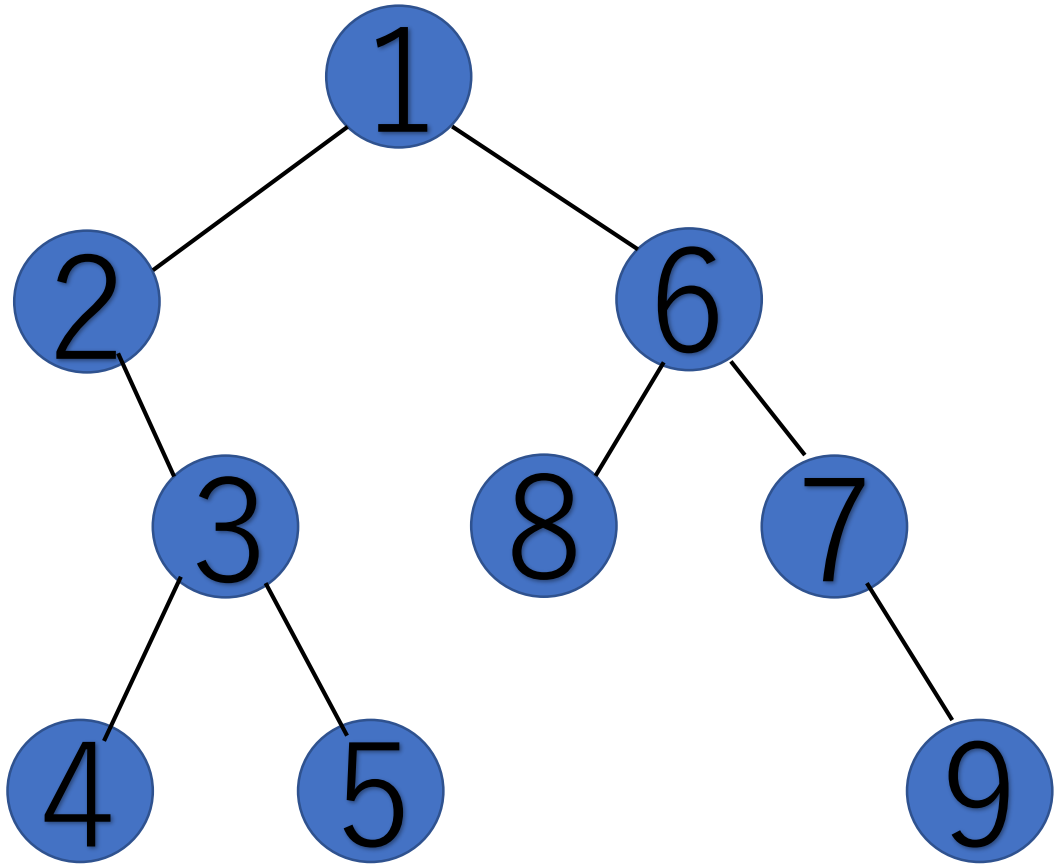
- 情報理論的下限
  - あるデータを表現するのに必要最低限のビット数
  - 例: 長さn のビットベクトル({0, 1}^n)
    - n bits
  - 例: 集合U = {1, 2,⋯, n} のサイズがmの部分集合集合
    - log(nCm) bits
  - 例: 頂点数n+1の根付き木
    - 個数はカタラン数に対応 C_n = (2nCn)/(n+1) ≒ 4^n/(n^(1.5)*sqrt(pi))
    - これは () をいい感じに並べる方法の数とか
    - 2n – Θ(logn) bits
  - 例: m種類のアルファベットからなるn文字の文字列の集合
    - nlogm bits
    - これ以上減らすと1対1で復元できなくなる

LCA/RMQ と簡潔データ構造

- Cartesian Tree(デカルト木)
  - 数列A[1...n] に対応するCartesian Tree は以下の再帰的手続きにより得られる
    - A[i] が A の最小値となるようなインデックスi を取る(複数ある場合は最小のi)
    - 根でA[i] を持ち, 左の子を A[1..i-1]に対するCartesian Tree, 右の子をA[i+1..n]に対するCartesian Tree とする.
    - これを配列の長さが0になるまで繰り返す

LCA/RMQ と簡潔データ構造

- Cartesian Tree(デカルト木)
  - 例: A = [2, 4, 3, 5, 1, 8, 6, 7, 9]
  - 構築O(n)(ジョイジョイジョイ参照)
  - Cartesian Tree上でLCAを解けば RMQ に答えられる

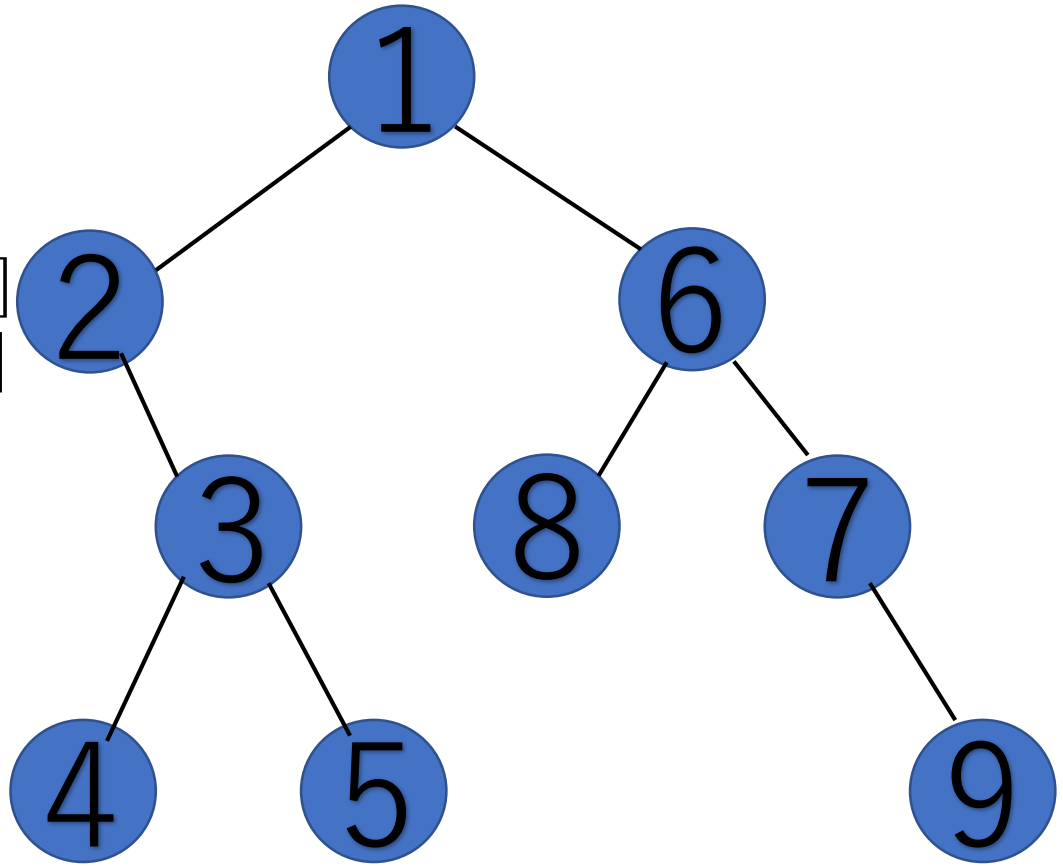


LCA/RMQ と簡潔データ構造

- Euler Tour
  - 木上でdfsすると列にできる
  - 例: 右の木で Euler Tour する
  - A = [1,2,3,4,3,5,3,2,1,6,8,6,7,9,7,6,1]
  - d = [0,1,2,3,2,3,2,1,0,1,2,1,2,3,2,1,0]
  - (A := 頂点番号, d = 深さ配列)

  - 構築O(n)

  - Euler Tour で作成した深さ配列上で RMQを解けばLCAに答えられる



LCA/RMQ と簡潔データ構造

- ±1 RMQ
  - 深さ配列の隣接要素の差は常に1
  - A = [1,2,3,4,3,5,3,2,1,6,8,6,7,9,7,6,1]
  - d = [0,1,2,3,2,3,2,1,0,1,2,1,2,3,2,1,0]
  - (A := 頂点番号, d = 深さ配列)

  - このような列上でのRMQを±1RMQと呼ぶ
  - ±1RMQが<O(n), O(1)>で解ければLCAも<O(n), O(1)>

LCA/RMQ と簡潔データ構造

- ここまでの要素を使えばRMQが<O(n), O(1)> で解ける
- 色々頑張ると 2n + o(n) bits になる
- 概要
  - 1. RMQ を Cartesian Tree 上での LCA に帰着
  - 2. LCA を Euler Tour で展開し ±1RMQ に帰着
  - 3. ±1RMQ を <O(n), O(1)> で解く
    - i. 列を logn/2 のブロックで分割(平方じゃない分割)
    - ii. 各ブロック内の最小値を前計算
    - iii. ii. の結果を Sparse Table に載せる(!!)
    - iv. 平方分割っぽくクエリに答える

LCA/RMQ と簡潔データ構造

- RMQ
  - 競プロでは定数倍が重すぎて普通にSparse Table でいいね…
  - そもそもメモリのlogが落ちて嬉しいかというと…

- LCA
  - こっちは高速. AOJ GRL_5_C (1e5 query LCA) で 0.04s(2位タイ, 入出力が遅そう) (http://judge.u-aizu.ac.jp/onlinejudge/review.jsp?rid=3585757)

- 省略したこと
  - Cartesian Tree の簡潔表現
  - クエリ時にブロックから余った部分の処理テク

  - 詳細は reference の参考書, スライドに載っています

LCA/RMQ と簡潔データ構造

- 色々頑張ると 2n + o(n) bits になると書いた
  - これは既に 長さn の配列に対するRMQデータ構造の情報理論的下限になっている
  - 証明:
    - 補題? 1: ノード数nの任意の二分木はある配列に対するCartesian Treeである
      - 帰納法で
    - 補題? 2: RMQデータ構造からCartesian Tree がO(n) で構成できる
      - 配列の任意の区間最小値がO(1)で分かるので
    - 補題? 3: ノード数nの異なる二分木は(2nCn)/(n+1)種類存在する
      - カタラン数 C_n
    - 以上よりRMQデータ構造の情報理論的下限は log(C_n) = 2n – Θ(n) bits となる

reference

- Sparse Table
- http://tookunn.hatenablog.com/entry/2016/07/13/211148

- Disjoint Sparse Table
- http://noshi91.hatenablog.com/entry/2018/05/08/183946
- https://inside.kmc.gr.jp/~kazuma/procon/slide/disjoint_sparse_table.pdf

- RMQ/LCA簡潔データ構造
- 簡潔データ構造(アルゴリズムサイエンスシリーズ)
  - https://www.kyoritsu-pub.co.jp/bookdetail/9784320121744
- http://joisino.hatenablog.com/entry/2017/08/13/210000
- https://www.slideshare.net/yumainoue965/lca-and-rmq