Advanced Data Structures

14004 구재현

헉 할거 많아.. ㅁㄴㅇㄹ

- 1. Heap
- 2. Prefix
- 3. Binary Search Tree
- 4. Coordinate Compression
- 5. Non-Recursive Segment Tree
- 6. Segment Tree (a lot of segment trees)
- 7. Related Topics / Challenges

Why Data Structures?

- ~~because your code is fucking slow~~
- 다양한 문제들의 시간 복잡도를 크게 향상시켜줌
- DP나 트리 등과도 궁합이 매우 잘 맞아서 복잡도 향상을 꾀할 수 있음
- 대부분 **이진 트리**를 사용해서 구현
- 수업은 19p부터 시작. 그전부분 모르면 읽어오세요

Heap

- 1. 원소 삽입
- 2. 최대 / 최소값 반환
- 3. 최대 / 최소값 삭제
- 가 O(lgn) 만에 되는 자료구조.
- http://koistudy.net/?mid=prob_page&NO=180
- 설명은 링크로 생략.

Heap 2 / 연습 문제 1

- 간단한 자료구조지만 굉장히 유용함.
- 다익스트라 알고리즘이 힙을 사용한 대표적 풀이법!
- priority_queue를 사용하면 push, pop, top 등의 접근자로 간단히 접근할 수 있음. 직접 짤 일이 별로..
- http://koistudy.net/?mid=prob_page&NO=180
- http://koistudy.net/?mid=prob_page&NO=1187
- http://koistudy.net/?mid=prob_page&NO=479

Prefixes

- Pmin[i] = min(Pmin[i-1],a[i]);
- Pmax[i] = min(Pmax[i-1],a[i]);
- Psum[i] = Psum[i-1] + a[i];
- 모든 이항 연산자에 대해 정의 가능

Prefixes 2

- postfix에 대해서도 정의가 가능할까?
- 2차원 일반화는? (-> IOI 2005 정원 분할 풀이 참조)
- 부분합이 적분의 개념이라면, 미분의 개념은?

연습 문제 2

- 정말 시도때도 없이 등장하는 개념이에요!
- http://koistudy.net/?mid=prob_page&NO=540
- http://koistudy.net/?mid=prob_page&NO=463
- http://koistudy.net/?mid=prob_page&NO=629
- http://koistudy.net/?mid=prob_page&NO=1036
- http://koistudy.net/?mid=prob_page&NO=380
- http://koistudy.net/?mid=prob_page&NO=1124
- http://koistudy.net/?mid=prob_page&NO=1229

Binary Search Tree

- 이진 탐색을 하기 위한 필수 조건은?
- 원소들이 특정한 “**순서**” 를 만족해야 함!
- -> 정렬이 필요.
- 이 때 원소 1개를 넣었다 뺄 때? **O(n)**
- 하지만 Binary Search Tree는? **O(lgn) !**

Binary Search Tree 2

- Binary Search Tree (BST)를 직접 구현하는 것은
- **코딩이 상당히** 어렵습니다 (200줄 ~ 300줄)

- 다행이 C++의 STL에 BST의 구현체가 있음.
- #include <set> 후, set<int> s; 로 트리를 선언.
- s.insert(x) <- 원소 삽입

Binary Search Tree 3

- 원소를 접근하는 것이 배열처럼 쉽지는 않음.
- iterator라는 포인터 비슷한 것으로 원소를 접근함.
  - s.begin()->[] ->[] ->[] ->[] ->[] s.end()->NULL
  - s.find(x) -> 원소가 존재하면 원소의 iterator를, 없으면 s.end() 를.

Binary Search Tree 4

- s.lower_bound(x) -> [x,inf) 범위의 원소 중 최대 **(이상)**
- s.upper_bound(x) -> (x,inf) 범위의 원소 중 최대 **(초과)**

- `set<int> ::iterator it = s.lower_bound(x);`
- `if(it == s.end()) puts("Bigger than Maximum");`
- `else printf("%d",*it);`
- it == s.end() 일 경우 접근시 런타임 에러

Binary Search Tree 5

- BST는 너무 중요해서 연구가 너무 많이 됐어요
- 우리가 흔히 쓰는 BST는 1972년 개발된 Red-Black Tree
- 연관자 배열은 BST의 주요 사용 예중 하나죠! (map)
- 정보를 하면 할수록 사람들이 **이걸 왜 연구했는지**를 알게 됩니다.

연습 문제 3

- BST만 써서 풀리는 문제.
- 생각보다 어렵네요. 못 푼다고 좌절하지 마세요.
- **O(NlgN)**에 풀어오세요!
- http://koistudy.net/?mid=prob_page&NO=624
- http://koistudy.net/?mid=prob_page&NO=385
- http://koistudy.net/?mid=prob_page&NO=540

Coordinate Compression

- koistudy.net/?mid=prob_page&NO=919
- N <= 10일 뿐인데, 좌표가 너무 크다!
- 하지만 칠했을 때를 상상해보면 아무리 생각해도 낭비가 굉장히 심한데…
- .. **좌표를 적당히 줄인 후** 배열에 칠해볼까?

Coordinate Compression 2

- 직사각형 1개는 X좌표 2개, Y좌표 2개를 가짐.
- X좌표와 Y좌표의 개수는 **많아야 2N**개!
- 이렇게 좌표를 배열에 들어갈 만큼 압축하고, 배열을 사용한 후 계산하는 것을 좌표 압축이라고 함.
- 좌표압축 외에도 정규화, 양자화 등의 용어도 간혹 사용.
- 좌표압축은 어떻게 하나요 그럼?

Coordinate Compression 3

- 1. 구조체 정렬을 사용하는 방법
- 2. 이진 탐색을 사용하는 방법

- 1번이 직관적이지만 2번을 사용하면 상당히 편해짐.
- 좌표들을 모두 vector 등에 때려박은 후 정렬.
- px[i] = lower_bound(vector,px[i]) - vector.begin()
- 이런 식으로 vector에서 px[i]의 인덱스를 새 좌표로 사용.

연습 문제 4

- http://koistudy.net/?mid=prob_page&NO=921
- http://koistudy.net/?mid=prob_page&NO=1087

- 좌표압축을 혼자 쓸일은 적음.
- 이후 세그먼트 트리를 배우다 보면 많아질 거임 ㅎㅎ

Segment Tree

- [1,1] / [1,2] / [3,3] / [1,4] / [5,5] / [5,6] / [7,7] / [1,8]
- 이런 식으로 구간을 나누면, 어떠한 구간 [s,e] 가 들어와도 O(lgN) 개의 구간만 관리하면 대표값을 찾을 수 있다!
- 왜 이런 식으로 구간을 나눌까?
- “분할 정복 알고리즘!”

Segment Tree

- 세그먼트 트리는 “**분할 정복 과정을 메모이제이션**” 했다고 생각해도 좋음.

- 분할 정복 과정을 메모이제이션 하면 상태는 O(n) 개 (트리를 그려보세요!)

- 한 쿼리당 보는 노드는 O(lgN)!

- 원소가 중간에 갱신되어도 O(lgN) 변화로 상태 유지 가능!

Segment Tree (non-recursive)

- Segment Tree는 분할 정복에 기반했듯이 재귀 함수로 짜는 경우도 많은데, 그렇게 짜지 않아도 됨.
- 재귀를 사용하지 않으면 고속으로 작동하며 구현도 쉬움
- lim이라는 적당한 크기의 2^k 꼴 정수를 잡고 원래 원소는 맨 뒤에, 대표값은 앞에
- 모범 코드 (최소값 트리) : http://codepad.org/7fPByS0g
- Prefix와 마찬가지로 모든 이항 연산자에 적용 가능

연습 문제 5

- http://koistudy.net/?mid=prob_page&NO=1075
- http://koistudy.net/?mid=prob_page&NO=1198
- http://koistudy.net/?mid=prob_page&NO=1082
- http://koistudy.net/?mid=prob_page&NO=766
- http://koistudy.net/?mid=prob_page&NO=670
- http://koistudy.net/?mid=prob_page&NO=1249
- http://koistudy.net/?mid=prob_page&NO=1188
- http://koistudy.net/?mid=prob_page&NO=1067
- http://koistudy.net/?mid=prob_page&NO=1218
- 1달 안에 다 풀기 힘들거에요 ^^;;

Segment Tree

- 구간 쿼리는 해결할 수 있다.
- 하지만.. 만약 **갱신이 구간으로 들어온다면**?

- 재귀를 사용해야지만 구간 갱신을 할 수 있음!
- 재귀를 사용해서 세그먼트 트리를 짜는 건 비재귀랑 비슷해요!

Segment Tree

- 구간 갱신을 하는 테크닉을 Lazy Propagation이라고 함
- “여기에 갱신이 들어왔어요 다음 쿼리랑 한꺼번에 처리하세요!” 정도의 플래그를 심음
- 때문에 **결합이 가능한 연산**만이 갱신으로 들어올 수 있음

Segment Tree

- 트리 말고 lazy라는 배열을 잡은 후, lazy[p]에 체크를 해 주면 서브트리는 모두 처리를 해야하는 상황
- 이후 어떠한 갱신 등이 들어왔을 때 lazy[p]를 서브트리에 넘겨주면서 해결
- 솔직히 말로 하니까 ㅈ같져? ㅋㅋㅋ 소스로 보여드림
- 모범 코드 : http://codepad.org/LeDVCwyY

연습 문제 6

- http://koistudy.net/?mid=prob_page&NO=1117
- http://oj.uz/problems/view/JOI13_construction
- http://koistudy.net/?mid=prob_page&NO=2000
- http://koistudy.net/?mid=prob_page&NO=740

Segment Tree w/ semi-Dynamic operation

- 이러한 연산은 굳이 크기를 고정시키지 않아도 동적으로 해결할 수 있음.
- 포인터를 사용하며 이것이 필수적이거나 편한 경우도 꽤 있음.
- 근데 해보니까 속도도 코딩도 그냥 좌표압축 하는게 보통 낫더라고요..
- 하지만 필요할 때가 있으니 꼭 익혀두세요 코드 생략.

Order Statistics w/ Segtree

- 다음과 같은 자료 구조가 있음
- **1) 원소를 삽입 삭제**
- **2) K번째 원소를 반환**

- Order Statistics라고 불리며 세그트리로 풀 수 있음.
- 왼쪽의 합, 오른쪽의 합을 비교하며 이진 탐색. 자세한 설명 생략
- 여담으로 BIT로 푸는게 더 편함.
- BIT 모범소스 : http://codepad.org/GoxYdkkG

Related Topics

- Sliding Window Minimum
  - Offline Batch Algorithm
    - which can replace RMQ in linear time
- Convex Hull Trick
  - Geometry-influenced Data Structure
  - Similar scheme with Sliding Window Minimum

Far more Advanced

- Splay Tree
- Persistent Segment Tree
- 으악 수고하셨습니다
- 수업할라다 만 Binary Indexed Tree 소개는 부록에.

Binary Indexed Tree

- Prefix sum을 다시한번 상기시켜보면
- S[s,e] = S[1,e] - S[1,s-1] = Psum[e] - Psum[s-1]
- Prefix sum도 Binary Search Tree처럼 동적으로 갱신시키려면?
- -> Binary Indexed Tree!
- **코딩 쉽고 유용하기까지 한 자료구조!**
- **.. 하지만 Segment Tree에 비해 큰 장점이 없으므로, 부록으로.**

Binary Indexed Tree 2

- Binary Indexed Tree 역시 부분합이 저장됨.
- 그런데 좀 이상함..
- Tree[1 ~ 8] 이 저장하는 부분합을 순서대로 말하자면
- [1,1] / [1,2] / [3,3] / [1,4] / [5,5] / [5,6] / [7,7] / [1,8]
- .. 이런 식? (입체적으로 그려봐요!)

Binary Indexed Tree 3

- [1,1] / [1,2] / [3,3] / [1,4] / [5,5] / [5,6] / [7,7] / [1,8]
- 5를 포함한 구간
- [1,1] / [1,2] / [3,3] / [1,4] / [5,5] / [5,6] / [7,7] / [1,8]
- 2을 포함한 구간
- 5 -> 6 -> 8 / 2 -> 4 -> 8 순으로 갱신
- 101(2) -> 110(2) -> 1000(2), 10(2) -> 100(2) -> 1000(2)
- 이진수 Last Bit을 더해주는 순서임.
- 2^0 대표값. 2^1 대표값 등을 쭉 올라가면서 본다 생각하면…

Binary Indexed Tree 4

- [1,1] / [1,2] / [3,3] / [1,4] / [5,5] / [5,6] / [7,7] / [1,8]
- 5를 포함한 구간 (5 -> 4)

- [1,1] / [1,2] / [3,3] / [1,4] / [5,5] / [5,6] / [7,7] / [1,8]
- 2을 포함한 구간 (2 -> 0)
- 이진수 Last Bit을 빼주는 형태.

Binary Indexed Tree 5

- x번째 값을 갱신해줄때는 x += LAST_BIT(x)
- x번째 값의 합을 구해줄때는 x -= LAST_BIT(x)
- 이를 임계점 (x > SIZE or x == 0)에 도달할때까지 반복
- LAST_BIT(x) = x & -x
- 자세한 증명은 모두 생략할게요 ㅠㅠ
- 코드 : http://codepad.org/AcA1bGhn