

B-Tree

Disk, 블록 기반 성능에 좋음으로 솔루션 구현이
쉽힌 경로이다.

Disk I/O 연산을 최적화하는데 있어서 브트리드로 보다 낫다.

노드를 떠올 때 B-tree의 높이는 $O(\lg n)$ 로 계산된다.

B-tree는 노드를 나누는 키의 경우가 경우가 되어
트리의 높이를 더 줄일 수 있다.

B-tree는 BST를 차원별로 일반화하는 것이다.

B-tree 내부 노드가 d 개의 key를 포함한다면
이는 $d+1$ 개의分支을 가진다.

이 외에는 각각의 key들은 노드 자체에 의해

구분하는 key의 범위를 $d+1$ 개의 다른 범위로
분할하는데 사용된다.

B-tree 특징. B-Tree: T, T.root를 주로 쓰는 듯.

1. 모든 노드는 대략 같은 높이 있다.

a. d, n은 모든 노드에 현재 새끼들이 있는 키의 개수이다.

b. $1 \cdot key_1 \leq 2 \cdot key_2 \leq \dots \leq d \cdot key_d$ 이므로
존재하는 키들 중에 있는 d 개의 키들은
 $1 \cdot key_1, 2 \cdot key_2, \dots, d \cdot key_d$ 이다.

c. key_i 는 key_i 값을 가지며 자식 노드인 때 TRUE,
자식 노드인 때 FALSE 값을 가진다.

2. 각 내부 노드는 자식 노드로 가르기는 $d+1$ 개의

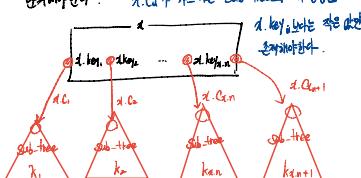
포인터 $1 \cdot C_1, 2 \cdot C_2, \dots, d \cdot C_d, \dots, C_{d+1}$ 을 가진다.

$i \cdot leaf \Rightarrow$ TLeaf는 이런 리프 노드이며 C_i 는 리프 노드이다.

3. key_i 는 각 세션 트리에 저장하는 키의 범위를 정한다.

$key_1 \leq 1 \cdot C_1 \leq key_2 \leq \dots \leq d \cdot C_d \leq key_{d+1}$

단지 예외이다. $1 \cdot C_1 + 1 \cdot key_1$ Sub-tree의 키 범위는
 $1 \cdot key_1$ 이라는 키의 범위를 포함한다.



4. 모든 리프 노드는 트리 높이마다 같은 형태를 갖다.

5. 노드는 그들이 포함할 수 있는 키의 개수에 대해 상한과 하한을
갖다. 이런 한계를 B-트리의 Minimum degree라고 하는

상한은 키의 개수를 이용해 표현된다. ($d-1 \leq d \cdot n \leq d-1$)

a. 트리를 제한한 모든 노드는 $d-1$ 개의 키를 갖다.

따라서 트리를 제한한 모든 노드는 자식 노드를

최대 $d-1$ 개 이상 가질 수 있다. 따라서 트리의

b. 모든 노드는 최대 $d-1$ 개의 키를 갖다. 따라서 트리의

가장 깊은 B-Tree는 $d=2$ 일 때 성립다.

$$1 \leq d, n \leq 3$$

$$\text{개수 } mn \quad \text{개수 } 2m \quad \text{개수 } 2m$$

자식 노드가 2개 자식 노드가 3개 자식 노드가 4개

2-3-4 트리를 갖는다.

실제로는 흡선처럼 많은 이유로 B-트리를 만든다.

B-Tree의 높이

B-트리에서 예상치의 적용을 유래해 필요한 디스크 접근 횟수는

B-트리의 높이에 비례한다. B-Tree가 최대 높이 때 최대 접근 횟수는

정비. $n \geq 2^h$ 때, 높이 h 에 부호 표기 $\lceil h \rceil$ 을 한다.

정의의 원칙 B-Tree Tree의 높이는 정한다.

$$h = \log_d \frac{n+1}{2}$$

증명.

레벨	노드 개수	키 갯수	T.root
0	1	1	
1	2	$2(d-1)$	
2	2^2		
3	2^3		
\vdots	2^{h-1}	$2^{d-1}(d-1)$	

$$\Rightarrow 1 + 2(d-1) + 2^2(d-1) + \dots + 2^{h-1}(d-1)$$

$$\Rightarrow 1 + 2(d-1) \frac{(d^h - 1)}{(d-1)}$$

$$\Rightarrow 1 + 2(d^h - 1) \quad (\text{키의 최대 개수})$$

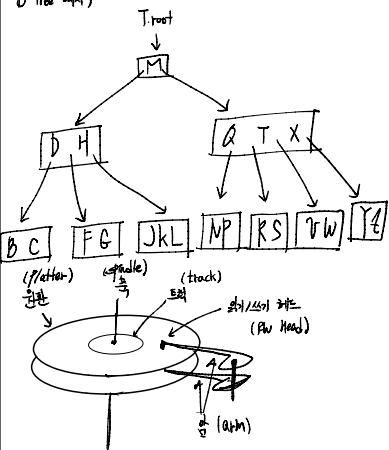
$$\Rightarrow n \geq 1 + 2(d^h - 1)$$

$$\Rightarrow \frac{n-1}{2} \geq d^h - 1$$

$$\Rightarrow \frac{n+1}{2} \geq d^h$$

$$\therefore h \leq \log_d \frac{n+1}{2} \leq \text{정한수}$$

B-Tree call()



디스크 드라이브 일반화

디스크 드라이브를 일반화하는 디스크 드라이브 구조.

Head 상태에는 트랙을 track 해친다.

Platter 및 Disk Drive Storage

B-tree의 특성

- B-Tree의 주트는 항상 미련에 있으므로 주트에 대해 Disk-READ을 수행할 필요가 전혀 없다. 그러나 주트가 비었을 때마다 Disk-WRITE를 한 번 수행해야 한다.
- 마세팅을 관리하는 모든 노드에 대해 Disk-READ 단계가 한번 수행해야 한다.

B-Tree의 경직

BST의 경직은 two-way 키 결정을 하고
 B-Tree의 경직은 multiway 키 결정을 한바탕 정비해서
 풀어 놓아두면 된다. 각 서브 트리마다 $\lceil \frac{n}{2} \rceil$ ~ $\lfloor \frac{3n}{4} \rfloor$
 $(d, n+1)$ 차 분기 결정을 한다.

B-TREE - SEARCH (x, k):

```

1 i = 1
2 while i <= d.n and k > x.keyi:
3   i = i + 1
4 if i <= d.n and k == x.keyi:
5   return (i, i)
6 elseif x.leaf
7   return NIL
8 else
9   Disk-READ(x, ci)
10  return B-TREE-SEARCH (x, ci, k)

```



이거 | & d,n까지 k가 key_{d,n}까지 같을 때까지 loop 돌린다.
 $(1, 2, 3)$

만약 k가 key_{d,n}과 같거나 같은 값은 찾았을 경우에나

종료면 \rightarrow return (i, i) / k 찾음

$(4, 5)$

각자 큰 경우는 자식 노드를 방문해서 더 찾아보아야하는데
 자식 leaf 노드면 k가 있으므로 return NIL (k 찾은)
 $(6, 7)$

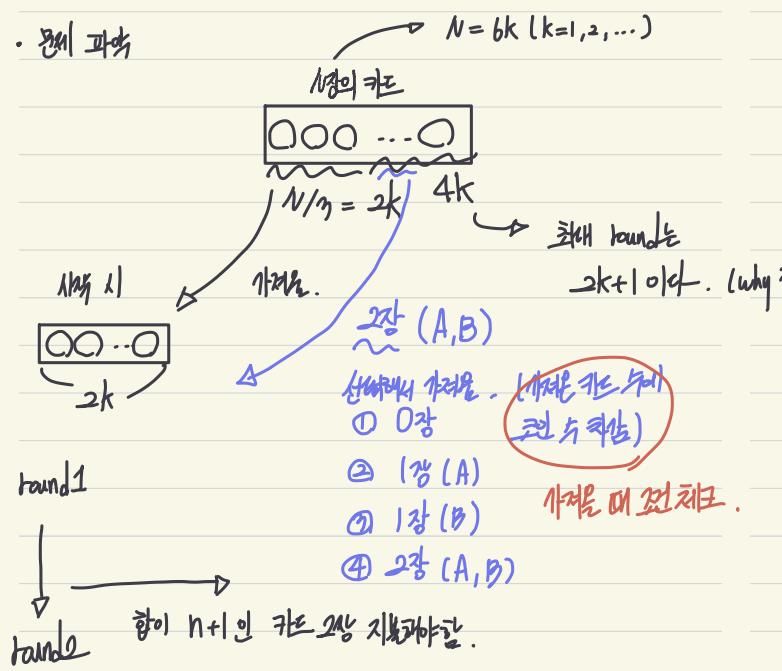
자식 leaf 노드 아니면 C에 B-TREE-SEARCH를 재귀시킨다.

하지만 C는 Main Memory에ことがある Disk-READ 단계로 종료된다.
 (9)

마지막 B-TREE-SEARCH(x, c_i, k)로 재귀시켜서 찾는다.

(10)

• 문제 과정



$$6k = n$$

$$\frac{n}{2} = k$$

평균 1장을.
 $k/2 = 2k$.
 다음 라운드는 한번 더.
 $\therefore 1+2k, (1라운드부터 시작)$

최대로 많은 round를 가지고 싶다.

문제.

입력: N , Coin 수, Cards

• 하지 방법 선택

① Brute Force | 위에서 ①, ②, ③, ④ 선택하는

1 round ○

① / ② / ③ / ④

2 round ○ ○ ○ ○

↓ 1111 111111 11111111 1111111111 4¹H 경우의 수

$\Rightarrow 1 + \underbrace{4 + 4^2 + 4^3 + \dots + 4^k}_{2k+1}$ 회수합의 합.

$$\Rightarrow 1 + \frac{4(4^k - 1)}{4-1} = 1 + \frac{4^{k+1} - 4}{3}$$

$\therefore O(2^n)$ 이다.

이 최대 범위는 1000 이다.

해당 가능하다.

모든 경우를 파악하는 것은 불가능하다.

문제의 숨은 힌트를 더 찾아야 한다. 더 좋은 방법이 필요할 것이다...

가지는 행위를 미리 보면 어떤가? 문제에서는 round마다 카드를

선택한다고 되어있는데 상세해보면 자동으로 필요할 때 coin 사용하고 가져가는 자는 없어졌다.

그 때 봄는 것과 동일한 결과를 얻어야

이렇게 하면 순서에 어떤 문제가 없으므로 차수 세번만은 걸리지 않는다.

$n = 1,000$ 놓으면 차수 세번만 아파요 OK.

★ ② Greedy?

현재 보유한 카드로만 거래해서 ($n+1$ 인 두 카드) 차운드를
넘어 사다가 저렴한 카드가 없다면 현재 round까지
선택할 수 있는 카드를 $0\text{~}n$ 을 차운드에서 가져온다.
최대한 보유한 카드를 사용하는 것이 coins 을 아끼고 많은
round에 살 수 있다.

|이| $1,000$ 이|이|에 대충 계산해도 충분하다. 예상으로 $O(n^2)$ 정도...

i) 보유한 카드를 담을 List

차운 round에서 사용할 수 있는 round 카드 list
카드 list에 포함이 $n+1$ 인 쌍이 있는지 check 하는 method
이 정도의 자료구조만 써도 OK.

ii) 보유 Card list에서 $n+1$ 인 카드 쌍 거래해서 차운드를 아낀다.

그다음 round를 넘기기 수 없을 경우 round Card list에서
/장의 카드를 가져와서 다음 round를 아낀다.

이것 아래 있으면 round Card list에서 그장의 round로 $n+1$ 을 만들어서

다음 round를 아낸다가 결국 안되면 Finish...

이 과정에서 round Card list ^{이거라도} 가져올 경우 가져온 Card 수만큼 coin 수 check 및 update...

• 평가

문제는 어렵지 않지만 문제에 의도와 꼭 일치할 때는 없다.

그 의도와 맞는 동일한 접근법을 찾아야하는 어려움이 있다. (처음에는 뻔하고 ...)

^{으로 한} - Bubble Force