트리 성능 분석

Tree performance analysis

Binary Tree & Red Black Tree

2020년 9월

노 영 래

목 차

제 1 장 서론

대표적인 트리인 Binary Tree와 이를 성능 개선한 Red Black Tree를 구현하고 삽입, 삭제, 삽입에 대해 성능 분석을 했다.

제 2 장 Tree 구현&분석

2.1 구현 테스트

잘 구현되었는지 테스트하기 위해 윈도우앱으로 삽입 삭제 과정을 랜더하여 트리를 육안으로 확인했고 또 자동으로 삽입, 삭제를 반복하며 전체 트리의 규칙에 위배되지 않는지 확인하는 기능을 넣어 로직에 문제가 없는지 확인했다.

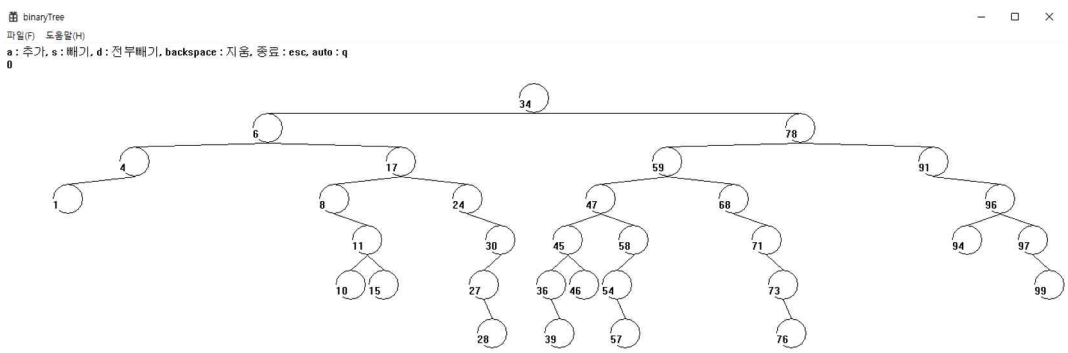

그림 1 Binary Tree by window application

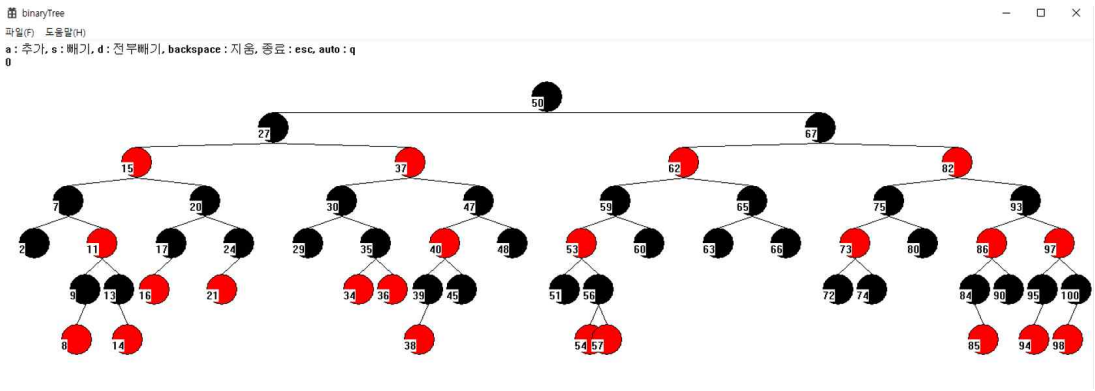

그림 2 RedBlackTree by window application

2.2 분석 테스트

각 트리에 0~N개의 데이터를 무작위로 삽입한 후 검색, 삭제, 삽입 의 성능이 어떻게 나오는지 자체 제작 프로파일러로 측정했다. 또 초기에 삽입한 N개의 데이터를 늘려가며 경향을 확인했다.

표 1 n:1000 insert, delete, search profiling(binary vs redblack)

N:1000				
Name	Average	Min	Max	Call
BinaryTree::Search	0.093	0	48.7	4923236
RedBlackTree::Search	0.086	0	101.4	4923236
BinaryTree::Delete	0.169	0	80.9	4923236
RedBlackTree::Delete	0.16	0	24.1	4923236
BinaryTree::Insert	0.163	0	51.4	4923236
RedBlackTree::Insert	0.175	0	32.9	4923236

테스트시에 주의 할 것은 같은 데이터를 가지고 검색, 삭제, 삽입하고 있기 때문에 검색의 경우 삽입 삭제보다 캐시 미스가 날 확률이 높고 실제로 테스트 했을

때 삽입 삭제보다 성능이 안 나왔다. (삽입 삭제는 검색을 포함하고 있는 구조라 검색만 했을 때 삽입, 삭제보다 성능이 안 나오는 것은 원하는 결과가 아니고 제대로 된 분석을 할 수 없다.) 그래서 프로파일링을 할 때 우선적으로 검색을 하고(이것은 프로파일링 대상이 아니다. 다음 행위들을 캐시 히트 시키기 위한 사전작업) 검색, 삭제, 삽입 테스트를 진행했다.

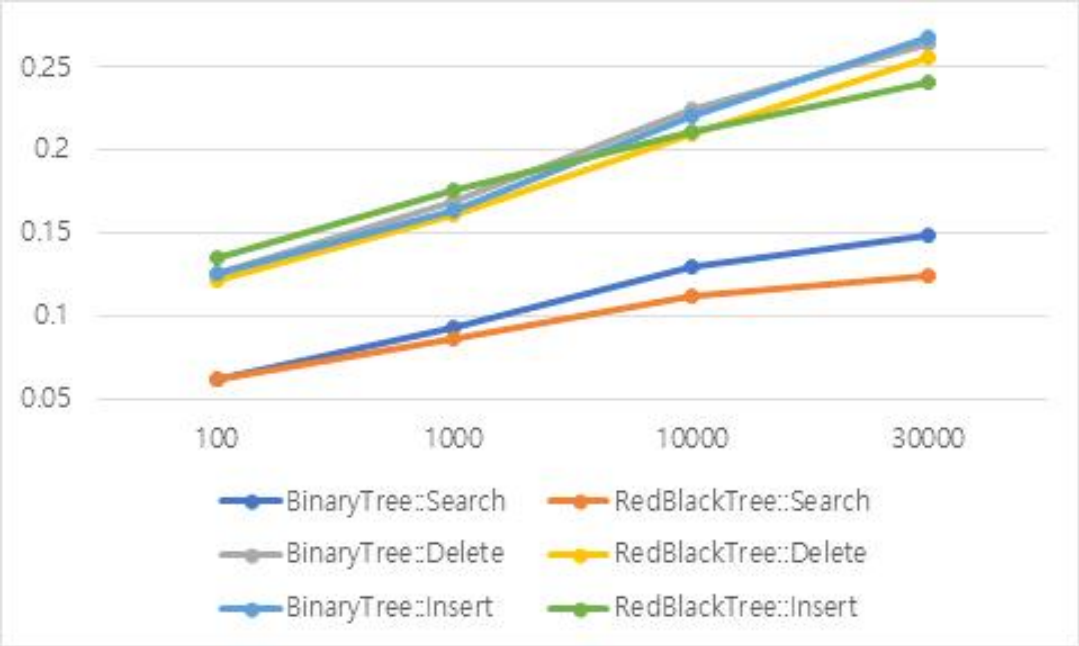

그림 3 insert, delete, search graph (binary vs redblack)

2.2.1 검색

100개의 데이터를 삽입해 놓고 검색을 수행한 경우에는 큰 차이가 없으나 데이터의 개수가 늘어날수록 그 성능 차이가 벌어지는 것을 볼 수 있다. BinaryTree의 경우 삽입, 삭제하는 것이 전부지만 RedBlackTree의 경우 BinaryTree에서 하는 삽입, 삭제 후에 RedBlackTree의 규칙을 토대로 트리의 밸런싱을 잡기 때문에 트리가 한쪽만 깊어지게 되는 불균형한 구조가 되는 것을 막아준다. 그러므로 검색시 RedBlackTree의 성능이 좋을 수 밖에 없고 이 성능차는

트리에 많은 데이터가 들어 있을수록 Binary Tree의 경우 depth의 편차가 더 심해지기 때문에 RedBlackTree와 성능차가 가속화 될 수밖에 없다.

2.2.2 삭제

삭제의 성능을 비교해보자. 삭제의 경우 BinaryTree의 경우 밸런싱 작업을 안하기 때문에 성능이 좋다고 생각할 수 있지만, 검색의 경우(삭제를 위한) 위에서 봤듯이 RedBlackTree가 더 빠르다. 두 포인트가 어느 정도로 상쇄가 될지, 어느 포인트가 더 성능에 가중치가 있는지는 그래프를 보면 알 수 있다. 내가 진행한 30000의 데이터 까지는 RedBlackTree가 일관되게 우세한 것을 볼 수 있다.

2.2.3 삽입

삽입의 성능을 비교해보자. 삽입의 경우도 삭제와 비슷하게 상쇄되는 포인트가 있다. 해당 데이터가 있는지 찾으며 내려가고 찾지 못하면 맨 아래까지 가서 삽입을 하기 때문에 해당 부분은 검색에 강점이 있고 depth가 균형을 이루어 비교적 바닥(leaf)까지 도달하는데 더 빠른 RedBlackTree가 강점이 있다. 반면 BinaryTree의 경우는 삽입 후 밸런싱 작업을 하지 않으니 비용을 아낄 수 있다. 그래프를 보면 알겠지만 초반에는 밸런싱작업을 하지 않는 BinaryTree가 더 빠른 것을 볼 수 있지만, 15000쯤 부터는 삽입 자체가 빠른 RedBlackTree가 강점을 보인다. 데이터가 많아질수록 더욱 가속화되는 것을 볼 수 있다.

제 4 장 결론

검색의 경우 Tree가 균형을 이루고있는 RedBlackTree가 더 빠르다. 삽입 삭제의 경우 검색과 달리 서로 상쇄 포인트가 있기 때문에 테스트 무조건 RedBlackTree가 빠르다고 단언할 수는 없다. 일단 내 테스트 환경에서 삭제의 경우 RedBlackTree가 우세했고 삽입의 경우 초반에는 BinaryTree가 우세하다가 15000쯤부터 역전되어 RedBlackTree가 우세한 것을 볼 수 있다.