

목차

1. Node 클래스 소개
2. main함수 소개
3. 각각의 함수 소개
4. 실행

1. Node Class Introduction

-pair class

-int key-pair의 key저장

-int value-pair의 value저장(Leaf에는 값 저장, nonLeaf에는 0저장)

-Node leftChildNode-왼쪽 아래 자식을 저장

-Node parent-Node의 부모를 저장

-Node r-Leaf는 right sibling을 저장, nonLeaf는 rightchild를 저장

-Node l-Leaf노드에서 탐색, 삭제의 편의성을 위해 자신 옆의 left sibling을 저장(Leaf에만 해당)

-List<pair> p-pair 배열을 저장

-boolean isLeaf-Leaf와 nonLeaf를 구별.

-void sorting()-Node 클래스의 원소들을 오름차순으로 정렬

2. main Function Introduction

String cmd=args[0]

String indexFile = args[1]

cmd를 switch문으로 경우를 나누어서

-c 인 경우에는 args[2]를 차수 m에 저장하고 createFile(indexFile)을 호출해서 index.dat파일에 차수를 쓰고 저장합니다.

-i인 경우에는 input.csv파일을 한줄씩 읽어나가면서 indexfile에 key, value형태로 저장합니다. insert가 들어올 때 마다 트리를 그리면서 하는 것은 너무나도 시간이 오래 걸리기 때문에 singleKeySearch나, rangedSearch가 들어오는 순간에만 트리를 그리려고, insert명령이여도 트리를 그리지 않고 indexfile에 저장합니다.

-d인 경우에는 delete.csv파일을 한줄씩 읽어나가면서 indexfile에 key 한줄 형태로 저장합니다. delete가 들어올 때 마다 트리를 그리면서 하는 것은 너무나도 시간이 오래 걸리기 때문에 singleKeySearch나, rangedSearch가 들어오는 순간에만 트리를 그리려고, delete명령이여도 트리를 그리지 않고 indexfile에 저장합니다.

-s인 경우에는 실제로 트리를 그려서 search를 해야 하기 때문에 indexfile에서 한줄씩 읽어 나가면서 ','를 포함하는 경우에는 insert명령이기 때문에 insert를 해주고, ','가 없다면 delete

명령이기 때문에 delete를 해줍니다. 파일을 다 읽었다면 그제서야 singleKeySearch를 해서 nonleafNode에서 지나가는 key값들을 출력하고 마지막엔 그 key값의 value를 출력합니다.

-r인 경우에도 트리를 실제로 그려야 하기 때문에 -s에서 소개했던 방법대로 ,의 포함 유무로 insert,delete를 실행합니다. 그리고 파일을 다 읽었다면 rangedSearch함수를 실행해서 범위 안의 key, value값을 출력합니다.

3. Functions Introduction

-public static void createFile(String file)

=>처음 command line에서 -c index.dat 3 할때처럼 index.dat를 초기화할 때 사용, 매개 변수 file을 FileWriter에 넣어degree를 file에 저장한다.

-public static Node singleKeySearch(int findKey)

=>말 그대로 B+tree에서 매개변수인 findKey를 찾는다. nonLeafNode들을 통과하면서 키 값들을 저장해놓을 배열keys를 만들고, Node cur을 만들어서 cur.isLeaf==false일 때 동안 findKey와 cur의 key들을 비교해가면서 cur을 이동시킨다. cur.isLeaf==true가 되었을 때 while문에서 나와서 해당 노드에서 순회해가면서 findKey와 같을 때를 찾아간다. 찾으면 keys를 모두 출력, 못찾으면 NOT FOUND를 출력한다.

-public static Node findLocation(int key)

=>매개변수인 key를 가지고 있는 leafNode를 찾는 함수이다. Node cur =root로 시작해서 cur.isLeaf==false 일 때 동안 cur을 적절한 노드로 이동시키고 cur.isLeaf==true일 때 반복문을 탈출한다.

-public static void rangedSearch(int start, int end)

=>매개변수인 start와 end가 있는 LeafNode를 findLocation함수로 찾는다. 이때 저장해놓은 Node변수가 left, right. 만약 left==right라면 찾고자 하는 범위가 한 노드 안에 있는 것을 의미하므로 그 노드를 순회하면서 범위 사이에 있는 key와 value를 출력한다. left!=right이라면 cur!=right일 때 동안 cur을 left부터 이동시키며 출력한다.

-public static int whereToInsert(Node node, int key)

=>매개변수인 node에서 key를 어느 위치에 삽입해야 하는지를 구하는 함수이다.

key가 node의 끝 key보다 크다면 node.p.size()를 리턴, 그게 아니라면 for문을 순회하면서 key<node.p.get(i).key일 때를 찾고 그 위치를 반환한다.

-public static void Node leafNodeSplit(Node overflowNode)

=>overflow가 생긴 매개변수 overflowNode를 매개변수로 받아 두 개로 나누는 함수이다.

int idx는 overflowNode에서 나눠야 하는 지점

int change_key는 overflowNode에서 split하고 부모 노드로 올라갈 key

Node parent는 overflowNode의 부모, 만약 없다면 새롭게 만들어야 한다.

int locat 는 change_key를 부모노드에서 삽입해야 하는 인덱스

새롭게 newNode를 만든다. 이 newNode가 overflowNode에서 절반 정도의 pair의 반아갈 새롭게 생길 leafNode이다. newNode에 overflowNode의 pair들을 저장하고 overflowNode에서 삭제한다. 만약 overflowNode의 부모노드가 없다면 새롭게 노드를 만들고 키를 저장하고 parent와 newNode끼리 가장 부모 자식 관계를 설정해준다. overflowNode의 부모노드가 있다면 wheretoinstert(parent,change_key)를 통해 locat에 그 위치를 저장한다.

```
-public static void Node nonLeafNodeSplit(Node overflowNode)
```

=>overflowNode가 nonLeafNode여서 nonLeafNodeSplit을 해주는 함수이다.

변수 설명

idx 는 overflownode의 절반을 알려주는 인덱스

change_key는 overflowNode에서 그 부모로 올라갈 인덱스의 key

newRightPointer는 overflowNode에서 그 부모로 올라갈 인덱스의 leftChildNode

Node newNode가 오른쪽에 생길 새로운 노드

함수 설명

먼저 idx+1부터 overflowNode.size()까지 newNode에 원소들을 삽입해주고 leftchildnode의 부모도 newNode로 설정해준다. 그 후 원래 overflowNode에서 나간 원소들을 삭제해준다.

overflowNode의 부모가 null이라면 parent=new Node()로 새로 만들고 root로 설정해준다.

parent에 change_key를 넣고 overflowNode를 자식으로 설정하고 parent.r에 newNode를 넣어준다. overflowNode의 부모가 null이 아니라면 locat = wheretoinstert(parent,change_key)를 통해 부모 노드에서 어디인덱스에 삽입해야 하는지 구한다.

만약 locat==0이라면 0번째 노드에 overflowNode를 leftChildNode로 가지게끔 삽입하고 1번째 노드의 leftChildNode가 newNode가 된다. 만약 locat==parent.p.size()라면 끝에 삽입하고 parent.r == newNode가 된다. 그 두 가지 경우가 아니라면 parent에 overflowNode를 왼쪽 자식으로 가지게 삽입하고 locat+1의 leftChildNode에 newNode를 자식으로 가지게끔 설정해준다. 그 후 newNode와 overflow간의 부모 자식 관계를 설정해준다.

```
-public static void insertion(int key, int value)
```

=>key와 value를 tree에 삽입하는 함수이다. 만약 root==null이라면 tree가 만들어지지 않았다는 뜻이므로 트리를 만들고 root에 삽입한다. root!=null이라면 Node locat = findLocation(key)를 통해 key가 들어갈 Node locat를 구한다. locat에 넣었을 때 overflowNode가 난다면 leafNodeSplit을 통해 찢고, 그 부모를 locat에 넣어준다. 그리고 while문을 돌면서 그 부모가 overflow가 안 날때까지 확인하면서 nonLeafNodeSplit을 진행한다.

```
-public static void printTree(Node root)
```

=>코드 디버깅에 사용한 함수이다. root부터 BFS 형식으로 바로 아래 레벨의 노드들을 queue에 저장한 후 차례대로 pop하면서 출력한다.

```
-public static Node islnNonLeaf(int key)
```

=>nonleafNode중에서 key를 가지고 있는 node를 리턴하는 함수이다. Node cur = root 로 시작해서 cur.isLeaf == false 일 때 동안 순회하며 돌고 key == cur.p.get(i).key일 때 cur을

return 하고 만약 `cur.isLeaf == true`가 되었다면 `nonleaf`에는 없었다는 말이므로 `null`을 리턴한다.

-public static int findIdx(Node cur, int key)

=>cur에서 key가 몇 번째 인덱스에 위치하는지 탐색하는 함수이다.

for문을 돌면서 `cur.p.get(i).key==key`일 때의 인덱스를 리턴한다.

-public static int findParentIdx(Node parent, Node child)

=>child 노드가 parent 노드에서 몇 번째 인덱스의 `leftNodechild` 또는 `r`인지 리턴하는 함수이다. 만약 `parent.r==child` 라면 `parent.p.size()`를 리턴하고 그것이 아니라면 for loop을 돌면서 `parent.p.get(i).leftChildNode==child` 일때의 인덱스를 리턴한다.

-public static void printLeftLeafNode(int key)

-디버깅 할 때 leafNode간의 왼쪽 자식이 잘 이어졌나 확인하는 함수이다. `findLocation`으로 key값을 가지고 있는 leafNode를 찾고 그 leafNode부터 l을 따라 쪽 따라간다.

-public static void borrowFromRight(Node cur, int idx)

=>delete할 때 오른쪽 노드로부터 원소를 빌려오는 함수이다. `changeKey`와 `changeValue`에 `cur.r`의 key와 value를 저장하고 cur에서 idx에 있는 원소를 삭제한다. 그 후 cur에 다시 그 `changeKey`와 `changeValue`를 저장하고 `cur.r`에서 첫 번째 원소를 삭제한다.

public static void borrowFromLeft(Node cur, int idx)

=>delete할 때 왼쪽 노드로부터 원소를 빌려오는 함수이다. `changeKey`와 `changeValue`에 `cur.l`의 마지막 원소들을 저장하고 cur에서 idx에 있는 원소를 삭제하고 cur에 왼쪽 노드에서 빌려온 원소를 저장하고 cur의 원소가 정렬되어있지 않기 때문에 `cur.sorting()`을 한다. 그 후 `cur.l`에서 마지막 원소를 삭제한다.

-public static boolean checkUnderflow(Node node)

=>삭제 전의 node의 사이즈가 $\text{ceil}(m/2)-1$ 보다 작거나 같으면 삭제시 바로 underflow가 발생하므로 그것을 체크해주는 함수이다.

-public static boolean checkUnderflowafterDelete(Node node)

=> 삭제 후 node의 사이즈가 $\text{ceil}(m/2)-1$ 보다 작으면 underflow인 것을 체크해주는 함수이다. `checkUnderflow`함수와 다른점은 이 함수는 삭제 후를 체크하는 것이고 `checkUnderflow`는 삭제 후 underflow가 발생할지 미리 체크하는 함수이다.

-public static Node findLeftSibling(Node node)

=>nonLeaf노드 중에서 node와 부모가 같으면서 바로 옆 `leftsiblingnode`를 찾는 함수이다. Node parent 는 node의 parent로 지정 후 `parent.p.get(0).leftChildNode`, `parent.r == node` 인지 확인하고 아니라면 for loop을 돌면서 같은지 확인후 그 인덱스 -1 의 `leftChildNode`를 리턴한다.

-public static Node findRightSibling(Node node)

=>nonLeaf노드 중에서 node와 부모가 같으면서 바로 옆 rightsiblingnode를 찾는 함수이다. parent를 node.parent로 지정 후 paren.r ==node이면 오른쪽을 구할 수 없기 때문에 null을 리턴하고 그게 아니라면 for loop을 돌면서 찾은 인덱스 +1의 leftChildNode를 리턴한다.

-public static void nonLeafNodeMerge(Node cur, Node side, Node parent, String dir)

-삭제 도중 nonLeafNode 간에서 서로 merge해야 할 때 사용하는 함수이다.

매개변수 설명

Node cur - underflow가 난 node이다.

Node side - node와 합쳐질 node이다.

Node parent - node와 side의 부모이다.

String dir - 어느쪽 방향과 merge할지 알려주는 함수이다.

변수 설명

int parentIdx - parent node에서 cur이 몇 번째 인지 findParentIdx 함수를 통해 찾는다.

함수 설명

먼저 dir이 "Left"이면 왼쪽 sibling node와 합친다는 의미이다. k라는 변수에 parent노드에서 cur옆의 key를 받아온다. 그리고 side에 넣어주고 parent node에서 지워준다. side노드에 cur의 원소들을 넣어주고 부모 자식 관계를 설정해준다. 만약 parent의 size가 0또는 parentIdx가 parent의 size보다 1클 때 (지웠기 때문에) parent의 r을 side로 설정해준다. 그게 아니라면 parent의 parentIdx-1번째 leftChild를 side로 설정해준다. 만약 parent가 root거나 parent의 size가 0이라면 parent가 없어지고 side가 root가 된다. 그렇지 않는다면 parent에서도 underflow가 났는지 checkUnderflowafterDelete함수를 통해 확인하고 findLeftSibling, findRightSibling함수를 통해 null이 아닌 노드를 다시 nonLeafMerge함수에 재귀적으로 넣어준다. dir이 "Right"인 경우, "Left"인 경우와 거의 일맥상통한데, "Left"일 때는 cur을 left에 넣어줬다면, 이번엔 right을 cur에 넣어줍니다. 또한 그렇기 때문에 parent의 size가 0이거나 parentIdx가 parent.p.size()와 같은지 확인합니다.

-public static void deletion(int key)

변수 설명

Node locat = findLocation(key) key가 있는 leafNode 위치를 찾는다.

boolean underflow = checkUnderflow(locat) locat이 underflow의 유무를 확인한다.

int idx = findIdx(locat, key) locat에 key가 몇 번째 인덱스에 있는지 저장한다.

함수 설명

만약 locat==root라면 그냥 지우면 된다. 아니라면 underflow여부에 따라 갈린다. underflow가 아니라면 그저 지우면 되고 underflow라면 1. 오른쪽 노드에서 값을 빌릴수도 있고 2. 왼쪽 노드에서 값을 빌릴 수도 있고 3. 둘다 못 빌린다면 합쳐야 한다. 1. 오른쪽 노드에서 값을 빌릴 때 에는 그 오른쪽 노드가 null이 아니어야 하고 underflow도 나지 않아야 하며, locat과 locat.r이 부모가 같아야 한다. borrowFromRight함수에서 값을 빌리고, 원래 있던 값을 지운다. Node nonLeafNodeChange2 에는 오른쪽에서 빌린 값이 locat의 마지막 값으로 되어있기 때문에 그 값을 넣어서 nonleaf에서의 그 값을 locat.r의 첫 번째 값으로 바꿔야한다. 그래서 isInNonLeaf함수로 찾고 교체해준다. 또한 지우는 키 값이 0이라

면 그 값도 nonLeaf에서 바꿔줘야 하기 때문에 isNonLeaf함수로 노드를 찾고, 교체해준다.

2. 왼쪽에서 빌릴 때에도 locat.이 null이 아니고, underflow가 나지 않으며, 같은 부모를 공유한다면, borrowFromLeft함수를 통해 값을 빌려오고 만약 지우려했던 키가 노드의 첫 번째였다면 그 값도 nonLeaf에서 찾아 바꿔줘야 한다. 그리고 인덱스가 0이 아니면, 왼쪽에서 들어오는 키가 노드의 첫 번째로 오기 때문에 nonLeaf에서 원래노드의 첫 번째 키를 찾아 교체해줘야 한다.

3. 둘다 못 빌릴 때에는 MERGE를 해야 한다.오른쪽에서 빌릴수 있는지, 왼쪽에서 빌릴 수 있는지를 확인하고 각각 키를 합쳐준다음, 부모 노드가 underflow났는지 확인하고 nonLeafNodeMerge를 불러서 부모 노드의 양 옆 노드 중 알맞은 노드와 merge할 수 있게 한다.

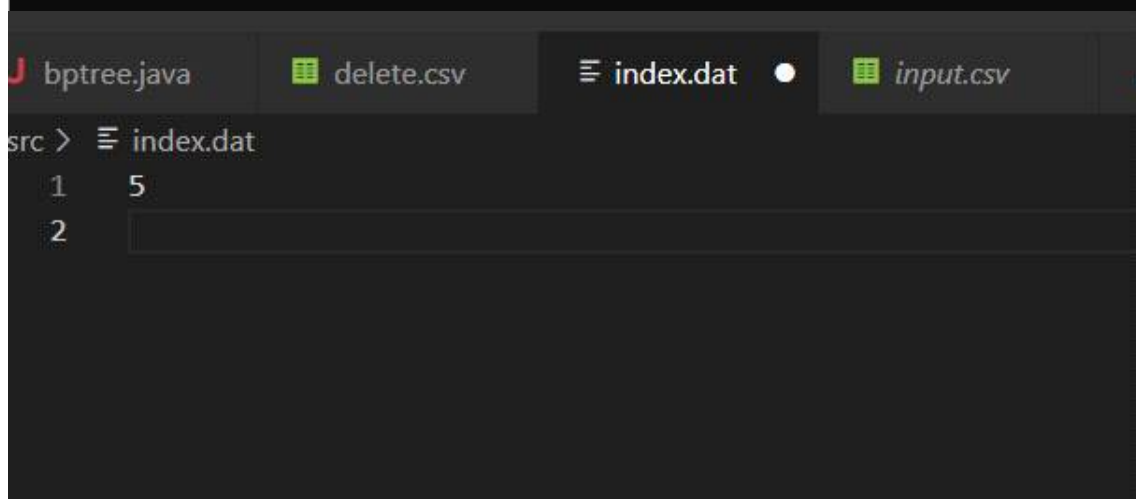
4. How to execute the file

-먼저 javac bptree.java로 컴파일을 해줍니다.

```
C:\workspace\vscode\bptree\src>javac bptree.java
C:\workspace\vscode\bptree\src>
```

-그 다음 java bptree -c index.dat 5로 degree값을 index.dat파일에 넣어줍니다.

```
C:\workspace\vscode\bptree\src>java bptree -c index.dat 5
C:\workspace\vscode\bptree\src>
```



<실행결과>

input.csv파일입니다.

-java bptree -i index.dat input.csv를 통해 input.csv를 index.dat에 넣어줍니다.

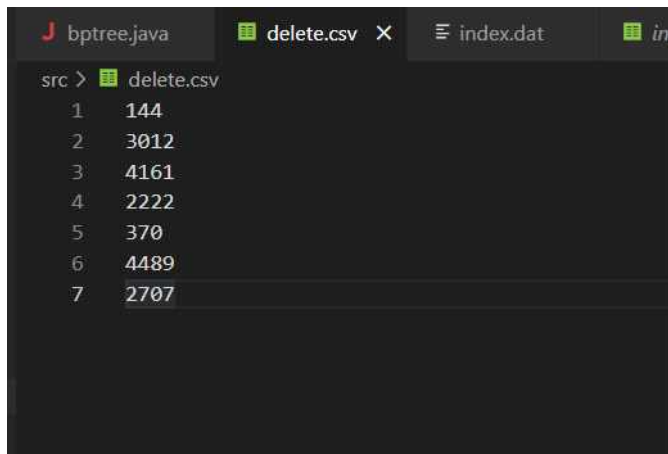
```
src > input.csv
1 1133,1133
2 7281,7281
3 2237,2237
4 7834,7834
5 1895,1895
6 144,144
7 3012,3012
8 8029,8029
9 4161,4161
10 2222,2222
11 9315,9315
12 9555,9555
13 6271,6271
14 5001,5001
15 9382,9382
16 5488,5488
17 7091,7091
18 8629,8629
19 8895,8895
20 9960,9960
21 6210,6210
22 728,728

C:\workspace\vscode\bptree\src>java bptree -i index.dat input.csv
C:\workspace\vscode\bptree\src>

src > index.dat
1 5
2 1133,1133
3 7281,7281
4 2237,2237
5 7834,7834
6 1895,1895
7 144,144
8 3012,3012
9 8029,8029
10 4161,4161
11 2222,2222
12 9315,9315
13 9555,9555
14 6271,6271
15 5001,5001
16 9382,9382
17 5488,5488
18 7091,7091
19 8629,8629
20 8895,8895
21 9960,9960
22 6210,6210
```

<실행결과>

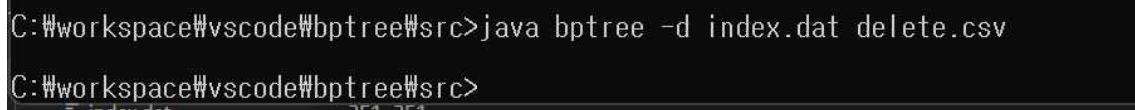
delete.csv 파일입니다.



The screenshot shows a VS Code editor with four tabs: bptree.java, delete.csv, index.dat, and input.csv. The delete.csv tab is active, showing a list of 7 numbers: 144, 3012, 4161, 2222, 370, 4489, and 2707. The cursor is at the end of the 7th line.

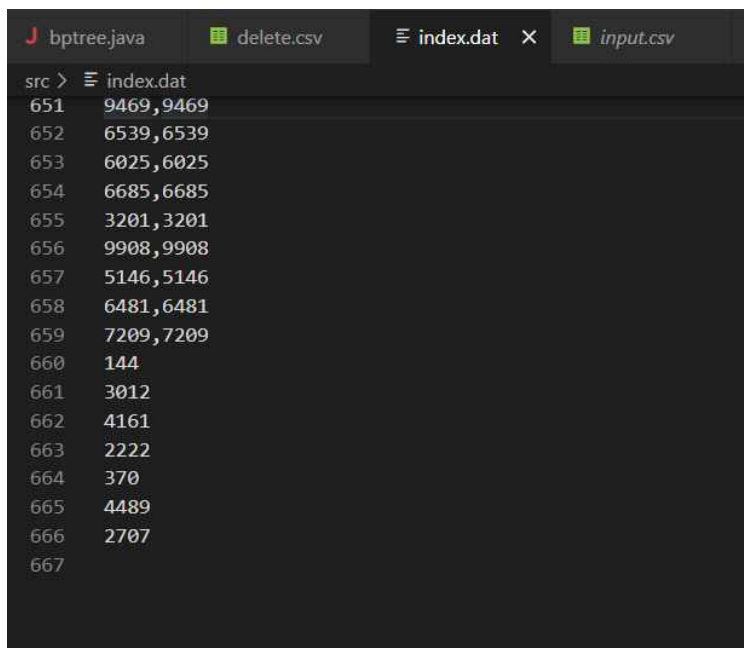
```
src > delete.csv
1 144
2 3012
3 4161
4 2222
5 370
6 4489
7 2707
```

-java bptree -d index.dat delete.csv를 통해 delete.csv 파일을 index.dat에 넣어줍니다.



The screenshot shows a terminal window with the command 'java bptree -d index.dat delete.csv' being executed. The output shows the file 'index.dat' being updated with the contents of 'delete.csv'.

```
C:\workspace\vscode\bptree\src>java bptree -d index.dat delete.csv
C:\workspace\vscode\bptree\src>
```



The screenshot shows a VS Code editor with four tabs: bptree.java, delete.csv, index.dat, and input.csv. The index.dat tab is active, showing a list of 17 lines of data. Each line contains a line number followed by a key-value pair. The key values are: 9469, 6539, 6025, 6685, 3201, 9908, 5146, 6481, 7209, 144, 3012, 4161, 2222, 370, 4489, and 2707.

```
src > index.dat
651 9469,9469
652 6539,6539
653 6025,6025
654 6685,6685
655 3201,3201
656 9908,9908
657 5146,5146
658 6481,6481
659 7209,7209
660 144
661 3012
662 4161
663 2222
664 370
665 4489
666 2707
667
```

<실행결과>

-java bptree -s index.dat 1133을 했을 때 결과입니다. key값들이 출력되고 한줄 띄고 value 값이 출력됩니다.


```
C:\workspace\vscode\bptree\src>java bptree -s index.dat 1133  
2173 895 1189 1133  
1133  
C:\workspace\vscode\bptree\src>
```

-java bptree -s index.dat 2707 했을 때 결과입니다. 2707은 delete.csv 파일에 있었기 때문에 NOT FOUND라고 잘 뜹니다.

```
C:\workspace\vscode\bptree\src>java bptree -s index.dat 2707  
2173 3012 2519 2634  
NOT FOUND  
C:\workspace\vscode\bptree\src>
```

-java bptree -r index.dat 200 300 했을 때 결과입니다. 사이에 있는 key와 value값이 잘 출력되는 것을 알 수 있습니다.

```
C:\workspace\vscode\bptree\src>java bptree -r index.dat 200 300  
228,228  
242,242  
259,259  
270,270  
277,277  
289,289  
294,294  
299,299  
C:\workspace\vscode\bptree\src>
```