



DS Programming Ex.2

과목	자료구조
제출 일자	2019.06.07
담당 교수	강현철 교수님
학 과	컴퓨터공학과
학 번	20172129
이 름	박예슬

1. 알고리즘 개요 설명

1. 입력 받은 matrix를 binary search tree로 저장한다.
2. 이때, 저장되는 링크드리스트는 typedef형으로 따로 자료구조를 지정한다. (2번에서 설명)
3. 저장하면서 matrix에 중복되는 인자가 있으면 count값을 더해준다.
4. 각 row별로 BTS가 완성이 되면, loser tree를 만들어 주기위해 row별 우선순위가 가장 높은 노드를 탐색한다. 이때 우선순위는 과제에서 주어진 tie breaking 규칙을 가장 만족하는 것을 최우선 순위로 둔다.
5. count가 가장 높은 노드가 최우선 순위이므로, 먼저 inorder traverse로 BTS를 한 번 돌면서 가장 높은 count를 탐색한다.
6. 가장 높은 count를 탐색했으면, 그 다음은 count가 가장 높으면서, key값이 가장 큰 수를 출력해야 한다. BST는 가장 큰 수가 가장 오른쪽 끝에 있으므로 가장 오른쪽 끝부터 차례로 count값이 가장 큰 노드를 찾아준다. 이때, 제일 먼저 발견되는 노드가 최우선 순위 노드이다.
7. 각 row(BTS)마다 최우선 순위 노드를 탐지했으면, 그 노드의 값을 loser tree의 leaf node에 저장해준다.
8. loser tree를 만들기 위해 우선 winner tree를 생성 후, loser tree를 만들 것이다.
9. tree의 level 수는 $\lfloor \log_2 k + 1 \rfloor$ 이므로 그 값을 'int timeComplex'에 저장해준다.
10. 저장된 timeComplex번 2의 제곱을 해 준 값이 가장 왼쪽 leaf node의 번호가 된다. 이를 이용해서 7번에서 실행하고자 했던, leaf node에 최우선 순위 노드를 저장해 줄 수 있다.
11. 이때 winner tree와 loser tree의 뼈대가 되는 자료구조는 배열로 저장을 한다.
12. 먼저, leaf node부터 2개씩 비교해주면서 부모 노드에 승자를 저장해준다. 최하위 레벨의 가장 왼쪽 노드의 번호가 $\text{pow}(2, \text{timeComplex})$ 이다. 다음 레벨로 넘어가려면 timeComplex를 1씩 빼 주면 된다.
13. 최종적으로 winner tree가 완성이 되면 배열의 1번 노드부터 마지막 노드까지 값이 저장된다. 이때, 아직은 loser tree를 구현하기 전이므로 0번 노드엔 값이 저장되지 않는다.
14. 완성된 winner tree를 가지고 먼저 1번 노드를 0번 노드에 저장해준다. 그리고 이번엔 반대로 위의 레벨부터 아래로 내려오면서 패자에 대한 포인터를 위치시킨다.
15. loser tree의 0번 노드를 출력해주고, 0번 노드가 속한 run의 최우선 순위를 다시 탐색한 뒤, 토너먼트를 재구성한다.
16. 모든 row의 값이 다 나올 때까지 반복한다.

2. 사용한 자료구조 및 함수

<사용한 자료구조>

tree를 linked representation으로 구현하기 위한 구조체	<pre>typedef struct node *TreePointer; typedef struct node { int run; int value; int count; TreePointer leftChild, rightChild; }node;</pre>
tree를 배열 representation으로 구현하기 위한 구조체	<pre>typedef struct tree { int run; int value; int count; }TreeArr;</pre>

<사용한 함수>

Binary Search Tree를 만들 때, node 삽입하는 함수	<pre>TreePointer BST_insert(TreePointer root, int value, int i) { if (root == NULL) { //초기 root값 설정 root = (node*)malloc(sizeof(node)); root->leftChild = root->rightChild = NULL; root->value = value; root->run = i; root->count = 1; return root; } else { if (root->value > value) root->leftChild = BST_insert(root->leftChild, value, i); else if (root->value < value) root->rightChild = BST_insert(root->rightChild, value, i); else (root->count)++; } return root; }</pre>
만들어진 BST에서 최대 count값 찾기	<pre>void MAX_count_search(TreePointer root, int* num) { if (root) { MAX_count_search(root->leftChild, num); if (root->count > *num) { *num = root->count; } MAX_count_search(root->rightChild, num); } }</pre>

<p>BST에서 가장 큰 count값이 주어졌을 때, 가장 최우선순위의 node를 찾는 함수</p>	<pre> void MAX_run(TreePointer root, int* num, TreePointer* temp) { if (root) { MAX_run(root->rightChild, num, temp); if (root->count == *num) { //num은 최대 count수 이므로 num++해서 다시 if문을 충족할 수 없도록 (*num)++; *temp = root; } MAX_run(root->leftChild, num, temp); } } </pre>
<p>BTS에서 run의 최우선 순위 레코드에 해당하는 노드를 찾아주는 함수 통합</p>	<pre> //count 크고, key값 큰 순으로 우선순위 나열하기 TreePointer traverse(TreePointer root) { //num번째 run을 찾는다. TreePointer temp = NULL; TreePointer *RUN = &temp; int num = 0; //MAX값 저장 int *ptr = &num; MAX_count_search(root, ptr); MAX_run(root, ptr, RUN); return temp; } </pre>
<p>두 개의 배열 노드가 주어졌을 때, 승자 노드를 return</p>	<pre> TreeArr winner(TreeArr root1, TreeArr root2) { if (root1.count > root2.count) return root1; else if (root1.count < root2.count) return root2; else { //count수가 같을 때 if (root1.value > root2.value) return root1; else if (root1.value < root2.value) return root2; else { //count수 ,value값 둘 다 같을 때 if (root1.count > root2.count) return root1; else return root2; } } } </pre>
<p>두 개의 배열 노드가 주어졌을 때, 패자 노드를 return</p>	<pre> TreeArr loser(TreeArr root1, TreeArr root2) { if (root1.count > root2.count) return root2; else if (root1.count < root2.count) return root1; else { //count수가 같을 때 if (root1.value > root2.value) return root2; else if (root1.value < root2.value) return root1; else { //count수 ,value값 둘 다 같을 때 if (root1.count > root2.count) return root2; else return root1; } } } </pre>

승자 트리를 생성하는 함수	<pre> void make_winnertree(TreeArr* ptr, int tree_level) { int i, j, leftmost; //가장 왼쪽 ==> 1,2,4(2^2),8(2^3)... 즉 (2^tree_level) for (i = tree_level; i > 0; i--) { leftmost = (int)pow(2, i); for (j = 0; j < leftmost; j += 2) { ptr[(leftmost + j) / 2] = winner(ptr[leftmost + j], ptr[leftmost + j + 1]); } } } </pre>
패자 트리를 생성하는 함수	<pre> void make_losertree(TreeArr* ptr, int tree_level) { ptr[0] = ptr[1]; int i, j, leftmost; //가장 왼쪽 ==> 1,2,4(2^2),8(2^3)... 즉 (2^tree_level) for (i = 1; i <= tree_level; i++) { leftmost = (int)pow(2, i); for (j = 0; j < leftmost - 1; j += 2) { ptr[(leftmost + j) / 2] = loser(ptr[leftmost + j], ptr[leftmost + j + 1]); } } } </pre>
출력된 레코드 값을 제거해주는 함수	<pre> void make_clean(TreePointer root) { root->count = 0; root->run = 0; root->value = 0; } </pre>

<main 함수>

timeComplex : loser tree의 level 수를 계산하기 위한 변수 loserLength : loser tree의 leaf node의 시작 번호 (가장 왼쪽 leaf node 시작 번호) run은 나중에 loser tree의 최종 승자가 소속된 row를 저장해, 다시 레코드를 뽑기 위한 변수
<pre> int main() { int timeComplex = (int)sqrt(numRow) + 1; int loserLength = pow(2, timeComplex); int i, run = 0; TreePointer BST[numRow]; //조건: row당 1개의 BST를 생성 TreePointer temp; TreeArr* tptr = malloc(sizeof(TreeArr)*loserLength * 2); </pre>
Matrix로부터 row별로 BST를 생성한다.
<pre> for (i = 0; i < numRow; i++) { root = NULL; for (int j = 0; j < numCol; j++) root = BST_insert(root, matrix[i][j], i); //TreePointer에 각 row별 BST저장 BST[i] = root; } </pre>

만들어진 BST들을 각각 최우선순위 노드를 뽑아 loser tree의 leaf node에 저장시켜 초기화한다.

```
//루저트리를 만들기 위해 leaf 노드에 각 행의 우선순위 값을 불러와 초기화한다.
for (i = 0; i < numRows; i++) { //numRow개의 run에서 모두 가져옴.
temp = traverse(BST[i]); //각 row의 최고 우선순위를 가져옴
tptr[loserLength + i].value = temp->value; //leafnode의 시작은 loserLength + i번 이다.
tptr[loserLength + i].run = temp->run;
tptr[loserLength + i].count = temp->count;
make_clean(temp);
} // 초기 run값을 loser tree 배열에 불러오기
```

출력된 0번 노드 값을 지우고, 0번 노드가 속한 run(row)에서 다시 새로운 레코드를 leaf node에 출력하여 loser tree를 재구성하는 것을 반복한다. 만약, 출력된 0번 노드의 count값이 0이라면 모두 출력되고 남은 레코드가 없는 것이므로 반복문을 종료한다.

```
printf("(row,value,count)Wn-----Wn");
while (1) {
    make_winnertree(tptr, timeComplex);
    make_loser tree(tptr, timeComplex);

    if (tptr[0].count == 0) break;
    printf("(%d, %d, %d)Wn", tptr[0].run, tptr[0].value, tptr[0].count);
    run = tptr[0].run; //출력되어 다시 꺼내 와야 될 run의 번호
    tptr[0].run = 0; tptr[0].value = 0; tptr[0].count = 0;

    temp = traverse(BST[run]);
    tptr[loserLength + run].value = temp->value;
    tptr[loserLength + run].run = temp->run;
    tptr[loserLength + run].count = temp->count;
    make_clean(temp);
}
}
```

3. Test Data 및 실행 화면

<Test 1>

① NumRow 값 : 5

② NumCol 값 : 10

③ Matrix 행렬 :

```
matrix[numRow][numCol] = {  
{ 200,90,200,9,80,80,200,90,7,90 },  
{ 100,30,30,51,160,160,160,51,160,59 },  
{ 500,100,7000,100,900,600,100,100,650,100 },  
{ 1000,300,41,300,41,41,41,900,900,950 },  
{ 90,81,81,95,81,83,81,90,81,90 }  
};
```

④ 실행 화면 :

C:\WINDOWS\system32\cmd.exe

(row,value,count)

```
-----  
(2, 100, 5)  
(4, 81, 5)  
(1, 160, 4)  
(3, 41, 4)  
(0, 200, 3)  
(4, 90, 3)  
(0, 90, 3)  
(3, 900, 2)  
(3, 300, 2)  
(0, 80, 2)  
(1, 51, 2)  
(1, 30, 2)  
(2, 7000, 1)  
(3, 1000, 1)  
(3, 950, 1)  
(2, 900, 1)  
(2, 650, 1)  
(2, 600, 1)  
(2, 500, 1)  
(1, 100, 1)  
(4, 95, 1)  
(4, 83, 1)  
(1, 59, 1)  
(0, 9, 1)  
(0, 7, 1)
```

계속하려면 아무 키나 누르십시오 . . .

<Test 2>


① NumRow 값 : 3

② NumCol 값 : 8

③ Matrix 행렬 :

```
matrix[numRow][numCol] = {  
{ 10,16,10,9,15,15,10,16},  
{ 9,30,9,51,9,160,9,51},  
{ 500,100,20,100,900,600,20,100},  
};
```

④ 실행 화면 :

 선택 C:\WINDOWS\system32\cmd.exe

(row,value,count)

```
-----  
(1, 9, 4)  
(2, 100, 3)  
(0, 10, 3)  
(1, 51, 2)  
(2, 20, 2)  
(0, 16, 2)  
(0, 15, 2)  
(2, 900, 1)  
(2, 600, 1)  
(2, 500, 1)  
(1, 160, 1)  
(1, 30, 1)  
(0, 9, 1)  
계속하려면 아무 키나 누르십시오 . . .
```