

4. Seletion Tree

- Loser Tree, Winner Tree

학번 : 201002513

이름 : 최 혁수

☐ 실습 1. Loser Tree를 이용한 Selection Tree 구현

- 개요 & 알고리즘
- Source Code
 - Loser Tree
- 실행결과 분석 & 구현상의 오류 및 한계

☐ 실습 2. Winner Tree를 이용한 Selection Tree 구현

- 개요 & 알고리즘
- Source Code
 - Winner Tree
- 실행결과 분석 & 구현상의 오류 및 한계

☐ 소감문 & 한계성

☐ All Source Code

- Winner Tree
- Loser Tree

실습 1. Loser Tree

- 개요 & 알고리즘
- Source Code
 - Loser Tree
- 실행결과 분석 & 구현상의 오류 및 한계

Loser Tree

□ 개요

K개의 런(Run)으로 나뉜 n개의 원소들을 순차순서(Ordered sequence)로 합병(Merging)하는 선택트리(Selection tree)의 방법 중 패자트리(Loser tree)의 개념적 이해와 구현방법을 모색한다.

□ 접근방법

- 8개의 Run[]을 오름차순 정렬된 난수 저장
- 단말노드 구성 -> 부모노드 -> 조상노드의 값을 토너먼트(Tournament)의 패자들로 구성
- 최종 승자(Smallest data)는 루트노드(Root node)에 저장
- 루트노드(Root node)에 저장된 런(Run)의 Index 정보에 따라 런의 재구성
- 루트노드(Root node)의 값(Key)를 출력 및 재구성된 런에 의한 재귀적 시행

□ 패자트리 알고리즘

트리(Tree)의 단말노드는 각 런(Run)의 최소 키(Key)값의 원소를 나타내며 두 자식노드들이 부모노드에서 토너먼트(tournament)를 수행하여 패자(big data)는 부모노드, 승자(small data)는 상위 부모노드에서 다시 토너먼트를 취하는 형식 즉, 루트노드(Root node)로 올라간 승자는 순차순서에 의한 출력

Step 1. K개의 런(Run)에 의한 패자의 집합 = 단말노드(terminal node) 구성

Step 2. 두 승자의 토너먼트(Tournament) : 패자(Parent node), 승자(->Tournament)

Step 3. 최종승자(Root node)가 결정

Step 4. 최종승자의 출력과 최종승자(Root node)에 저장된 런(Run) index 정보에 의한 런(Run)의 재구성 및 Step 1. 재귀적인 수행

LoserTree Source Code(1/5)

□ head.h

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define TREE_SIZE 8 // 트리 및 런 사이즈
#define MAX_KEY 10000 // 런에 들어갈수 있는 키의 최대값

typedef struct _treeNode{ // Selection Tree의 노드
int key; // node의 key
int idx; // node의 키값이 위치하는 run의 번호
} treeNode;

typedef struct _SelectionTree{ // Selection Tree 구조체
treeNode tree[TREE_SIZE]; // Tree를 나타낼 배열
} SelectionTree;

typedef struct _SetOfRuns{ // run들의 구조체
int run[TREE_SIZE][TREE_SIZE];
//int topOfRuns[TREE_SIZE]; // 사용 안한 함수
} SetOfRuns;

// Run을 초기화한다.
// run에 임의의 값을 채운다. 각 런마다 sort된 값을 채운다.
void initRuns(SetOfRuns* runs);

// SelectionTree를 초기화 한다.
// 각 런의 첫번째 노드를 가지고 SelectionTree를 구성한다.
void initSelectionTree(SelectionTree* sTree, SetOfRuns* runs);

//SelectionTree를 이용하여 run에서 가장 작은 Key값 하나를 리턴한다.
//동시에 SelectionTree를 재구성한다.
//만약 run에 남아있는 노드가 없다면 MAX_KEY를 리턴한다.
int getWinner(SelectionTree* sTree, SetOfRuns* runs);

//SelectionTree를 출력한다.
void printTree(SelectionTree* sTree);

//현재 run에 남아있는 노드의 키값을 적당한 형태로 출력한다.
void printRuns(SetOfRuns* runs);
```

LoserTree Source Code(2/5)

❑ void initRuns(SetOfRuns*)

```
/** runs 초기화 : 난수에 대한 오름차순 */
void initRuns(SetOfRuns* runs){
    int i, j, k;
    int temp;
    srand((unsigned)time(NULL));

    for(i=0; i<TREE_SIZE; i++)
    for(j=0; j<TREE_SIZE; j++)
    runs->run[i][j] = rand()%10000; // MAX_KEY 보다 작은 난수를 무작위로 run[] 저장

    /** 버블 정렬에 의한 run[] 오름차순 정렬 */
    for(k=0; k<TREE_SIZE; k++)
    for(i=0; i<TREE_SIZE; i++)
    for(j=i+1; j<TREE_SIZE; j++){
        if(runs->run[k][i] > runs->run[k][j]){
            temp = runs->run[k][i];
            runs->run[k][i] = runs->run[k][j];
            runs->run[k][j] = temp;
        }
    }
}
```

❑ int getWinner(SelectionTree* , SetOfRuns*)

```
int getWinner(SelectionTree* sTree, SetOfRuns* runs){
    int i;
    int temp = sTree->tree[0].key; // initSelectionTree()의 Root 노드를 임시저장
    for(i=0; i<TREE_SIZE-1; i++){ // run[] 재구성
        runs->run[sTree->tree[0].idx][i] = runs->run[sTree->tree[0].idx][i+1];
    }
    // 최상위 Root 노드에 저장된 idx를 이용해 run[idx][7] 값에 MAX_KEY 저장.
    runs->run[sTree->tree[0].idx][TREE_SIZE-1] = MAX_KEY;
    initSelectionTree(sTree, runs); // 재구성된 run[]을 이용한 Loser Tree 재구성
    return temp; // 임시 저장된 temp값 return(출력)
}
```

LoserTree Source Code(3/5)

❑ void initSelectionTree(SelektionTree* ,SetOfRuns*)

```
/* run[]에 의한 Loser Tree 구성 */
void initSelectionTree(SelectionTree* sTree, SetOfRuns* runs)
{
    int i, j;
    int level3=4;    // 단말노드 첫 idx : 4
    int temp[2];     // sTree->tree[0], tree[1] 임시저장 값
    int f_num=0;
    // sTree->tree[i].key & idx = 0 & -1 로 초기화
    for(i=0; i<TREE_SIZE; i++){
        sTree->tree[i].key = 0;
        sTree->tree[i].idx = -1;
    }
    // run[]에 의한 Loser Tree 구성
    for(i=0; i<TREE_SIZE; i+=2){ // i+=2 : 2개의 run[]의 비교로 1개의 tree[]를 구성
        if(runs->run[i][0] > runs->run[i+1][0]){ // 첫 단말노드 구성
            sTree->tree[level3].key = runs->run[i][0]; // 작은 값을 저장
            sTree->tree[level3].idx = i;
            // 부모노드가 채워져있으면서 부모노드가 자식노드보다 작다면
            if(sTree->tree[level3/2].key !=0 && sTree->tree[level3/2].key < runs->run[i+1][0]){
                // 자식노드를 부모노드로 보낸다.
                sTree->tree[level3/2].key = runs->run[i+1][0];
                sTree->tree[level3/2].idx = i+1;
                level3++;
            }
            // 부모노드가 크다면 level3++로 skip
        } else if(sTree->tree[level3/2].key != 0 && sTree->tree[level3/2].key > runs->run[i+1][0]){
            level3++;
        }
        else{ // 부모노드가 비어있을때 자식노드값으로 초기화
            sTree->tree[level3/2].key = runs->run[i+1][0];
            sTree->tree[level3/2].idx = i+1;
            level3++;
        }
    }
}
```

~ 계속

LoserTree Source Code(4/5)

❑ void initSelectionTree(SelecionTree* ,SetOfRuns*)

```
else{ // == if(runs->run[i][0] < runs->run[i+1][0]) 이면서, 위와 같은 조건을 취한다.
    sTree->tree[level3].key = runs->run[i+1][0];
    sTree->tree[level3].idx = i+1;
    if(sTree->tree[level3/2].key != 0 && sTree->tree[level3/2].key < runs->run[i][0]){
        sTree->tree[level3/2].key = runs->run[i][0];
        sTree->tree[level3/2].idx = i;
        level3++;
    }
    else if(sTree->tree[level3/2].key != 0 && sTree->tree[level3/2].key > runs->run[i][0]){
        level3++;
    }
    else{
        sTree->tree[level3/2].key = runs->run[i][0];
        sTree->tree[level3/2].idx = i;
        level3++;
    }
}

// 쓰이지 않은 run[]의 idx 값을 temp[2]에 저장
// temp[2]에 저장된 2개의 정보는, tree[0] or tree[1] 에 채워진다.(최소값 2개)
for(i=0; i<TREE_SIZE; i++){
    for(j=0; j<TREE_SIZE; j++){
        if(i==sTree->tree[j].idx) break;
        else if(j==7) temp[f_num++] = i;
    }
    // temp[2]의 2개의 idx값을 통해, 두개의 run[]을 비교하여 tree[0] or tree[1]에 저장.(idx or .key)
    if(runs->run[temp[0]][0] > runs->run[temp[1]][0]){
        sTree->tree[0].key = runs->run[temp[1]][0];
        sTree->tree[0].idx = temp[1];
        sTree->tree[1].key = runs->run[temp[0]][0];
        sTree->tree[1].idx = temp[0];
    }
    else{
        sTree->tree[0].key = runs->run[temp[0]][0];
        sTree->tree[0].idx = temp[0];
        sTree->tree[1].key = runs->run[temp[1]][0];
        sTree->tree[1].idx = temp[1];
    }
}
```


LoserTree Source Code(5/5)

❑ void printTree(SelectionTree*)

```
/** Expression Tree **/
void printTree(SelectionTree* sTree){
    printf("          %d          \n",sTree->tree[0].key);
    printf("          %d          \n",sTree->tree[1].key);
    printf("      %d          %d          \n",sTree->tree[2].key,sTree->tree[3].key);
    printf("%d          %d          %d          %d          \n",sTree->tree[4].key,sTree->tree[5].key,sTree->tree[6].key,sTree->tree[7].key);
}
```

❑ int getWinner(SelectionTree* , SetOfRuns*)

```
/** Expression Run **/
void printRuns(SetOfRuns* runs){
    int i, j;
    for(i=0; i<TREE_SIZE; i++){
        for(j=0; j<TREE_SIZE; j++){
            printf("%4d  ", runs->run[j][i]);
        }
        printf("\n");
    }
    printf("\n");
}
```

❑ 실행결과 1

```
//실행 결과 1번
while( (i = getWinner(&sTree, &runs)) != MAX_KEY)
{
    printf("%4d ", i);
    if(j%10==9)
        printf("\n");
    j++;
}
printf("\n");
```

❑ 실행결과 2

```
//실행 결과 2번 : 32번 getWinner 출력 후 Tree & Runs 출력
for(i=0; i<32; i++){
    printf("%4d ",getWinner(&sTree, &runs));
    if(i%10==9) printf("\n");
}
printf("\n");
printf("===== SelectionTree View =====\n");
printTree(&sTree);
printRuns(&runs);
```

실행결과 분석 & 구현상의 오류 및 한계

□ 실행 결과화면 1

```

C:\Windows\system32\cmd.exe
===== SelectionTree View =====
                                125
                                428
                                372
                                1464
                                1027
                                391
                                2144
                                675
125 1027 391 372 2144 1464 675 428
270 2412 1525 3934 2431 2958 3218 1087
874 2576 2950 4701 3431 3865 3540 1096
1077 3873 3670 6333 3597 3866 4511 4529
3462 5631 5125 6436 6434 6291 6750 5013
3759 6169 6533 7211 6494 6830 7239 5527
8394 8014 7752 8056 7098 7954 8690 7171
9972 9412 9647 9446 7543 8310 8987 8493

=====
125 270 372 391 428 675 874 1027 1077 1087
1096 1464 1525 2144 2412 2431 2576 2950 2958 3218
3431 3462 3540 3597 3670 3759 3865 3866 3873 3934
4511 4529 4701 5013 5125 5527 5631 6169 6291 6333
6434 6436 6494 6533 6750 6830 7098 7171 7211 7239
7543 7752 7954 8014 8056 8310 8394 8493 8690 8987
9412 9446 9647 9972
계속하려면 아무 키나 누르십시오 . . .

```

□ 실행 결과화면 2

```

C:\Windows\system32\cmd.exe
===== SelectionTree View =====
                                3
                                461
                                1813
                                244
                                2407
                                1607
                                422
                                1575
2407 1813 1607 461 3 422 244 1575
3355 2170 3583 1383 154 1931 1213 2433
5072 3676 3818 2028 737 3586 2162 3848
6749 4011 6623 3231 1346 4158 2512 5499
7886 7056 8088 4042 1942 5815 6920 7350
8940 7801 8433 4829 2046 5834 7274 7802
9568 8391 9090 5456 2791 6612 7621 9717
9812 8755 9588 6565 4381 7453 9785 9724

=====
3 154 244 422 461 737 1213 1346 1383 1575
1607 1813 1931 1942 2028 2046 2162 2170 2407 2433
2512 2791 3231 3355 3583 3586 3676 3818 3848 4011
4042 4158
===== SelectionTree View =====
                                4381
                                4829
                                5072
                                5499
                                7056
                                6623
                                5815
                                6920
5072 7056 6623 4829 4381 5815 6920 5499
6749 7801 8088 5456 10000 5834 7274 7350
7886 8391 8433 6565 10000 6612 7621 7802
8940 8755 9090 10000 10000 7453 9785 9717
9568 10000 9588 10000 10000 10000 10000 9724
9812 10000 10000 10000 10000 10000 10000 10000
10000 10000 10000 10000 10000 10000 10000 10000
10000 10000 10000 10000 10000 10000 10000 10000
계속하려면 아무 키나 누르십시오 . . .

```

실행결과 분석 & 구현상의 오류 및 한계

□ 구현상의 오류 및 한계

패자트리(Loser Tree)의 근간이 되는 `initSelectionTree()` 구현시 최소의 비교문을 통한 트리를 구성하려 하였으나 반대로 상당한 비교문을 통해 구현되었기 때문에 그만큼의 메모리가 소모되고 있다. 또한 `Tree[0]`, `Tree[1]`을 한번의 for문을 통한 input이 제한되어 따로 두 값을 비교를 통해 토너먼트로 승자와 패자를 가리고 있다는 점도 구현상의 오류로 볼 수 있다.

루트노드(`tree[0]`)의 값을 출력할시 재정립된 런과 이를 통한 트리도 재구성되고 있는데 이때 `MAX_KEY`값을 런의 끝자락에 저장하고 있는 형태를 띄기 때문에 실습과제로 나온 PDF파일의 출력결과2와는 다르게 `MAX`의 값(10000)이 보여지고 있다.

PDF와 같이 출력된 값이 해당 런에서 안보이게끔 나오는 방법을 찾지 못했기 때문에 이는 구현상의 한계로 볼 수 있다.

실습 2. Winner Tree

- 개요 & 알고리즘
- Source Code
 - Winner Tree
- 실행결과 분석 & 구현상의 오류 및 한계

Winner Tree

□ 개요

K개의 런(Run)으로 나뉜 n개의 원소들을 순차순서(Ordered sequence)로 합병(Merging)하는 선택트리(Selection tree)의 방법 중 승자트리(Winner tree)의 개념적 이해와 구현방법을 모색한다.

□ 접근방법

- 8개의 Run[]을 오름차순 정렬된 난수 저장
- 단말노드 구성 -> 자식노드간의 토너먼트로 승 Or 패자 결정
- 최종 승자(Smallest data)는 루트노드(Root node)에 저장
- 루트노드(Root node)에 저장된 런(Run)의 Index 정보에 따라 런의 재구성
- 루트노드(Root node)의 값(Key)를 출력 및 재구성된 런에 의한 재귀적 시행

□ 승자트리 알고리즘

트리(Tree)의 단말노드는 각 런(Run)의 최소 키(Key)값의 원소를 나타내며 패자트리(Loser Tree)와는 다르게 부모노드(Parent node)가 아닌 자식노드(Children Tree)끼리의 토너먼트(비교우위)를 통해 승자는 계속해서 상위로 올려보내는 형식을 취하고 있다.

즉, 루트노드(Root node)로 올라간 승자는 순차순서에 의한 출력

Step 1. K개의 런(Run)에 의한 패자의 집합 = 단말노드(terminal node) 구성

Step 2. 자식노드(Children node)끼리의 토너먼트(tournament) 수행 -> 상위레벨로 이동

Step 3. 최종승자(Root node)가 결정

Step 4. 최종승자의 출력과 최종승자(Root node)에 저장된 런(Run) index

정보에 의한 런(Run)의 재구성 및 Step 1. 재귀적인 수행

WinnerTree Source Code(1/1)

❑ void initSelectionTree(SeletionTree* ,SetOfRuns*)

```
/* run[] 에 의한 Winner Tree 구성 */
void initSelectionTree(SelectionTree* sTree, SetOfRuns* runs){
    int i;
    int level3 = 4; // 단말노드 첫 idx : 4
    // sTree->tree[i].key & idx = 0 & -1 로 초기화
    for(i=0; i<TREE_SIZE; i++){
        sTree->tree[i].key = 0;
        sTree->tree[i].idx = -1;
    }
    // run[]에 의한 Winner Tree 구성
    // 단말노드(Terminal node)를 초기화
    for(i=0; i<TREE_SIZE; i+=2){
        if(runs->run[i][0] < runs->run[i+1][0]){
            sTree->tree[level3].key = runs->run[i][0];
            sTree->tree[level3].idx = i;
            level3++;
        }
        else{
            sTree->tree[level3].key = runs->run[i+1][0];
            sTree->tree[level3].idx = i+1;
            level3++;
        }
    }
    // 단말노드(Terminal node), 즉 자식노드(children)간의 비교를 통해 부모노드 초기화
    for(i=4; i<TREE_SIZE; i+=2){
        if(sTree->tree[i].key < sTree->tree[i+1].key){
            sTree->tree[i/2].key = sTree->tree[i].key;
            sTree->tree[i/2].idx = sTree->tree[i].idx;
        }
        else{
            sTree->tree[i/2].key = sTree->tree[i+1].key;
            sTree->tree[i/2].idx = sTree->tree[i+1].idx;
        }
    }
    // 자식노드(children)간의 비교를 통해 부모노드 초기화
    if(sTree->tree[2].key < sTree->tree[3].key){
        sTree->tree[1].key = sTree->tree[2].key;
        sTree->tree[1].idx = sTree->tree[2].idx;
    }
    else{
        sTree->tree[1].key = sTree->tree[3].key;
        sTree->tree[1].idx = sTree->tree[3].idx;
    }
    // 헤더파일에 제시된 tree[]의 갯수를 채우기 위해 tree[0]를 쓰고있다.
    sTree->tree[0].key = sTree->tree[1].key;
    sTree->tree[0].idx = sTree->tree[1].idx;
}
```


실행결과 분석 & 구현상의 오류 및 한계

□ 구현상의 오류 및 한계

패자트리와 같이 비교문의 반복횟수를 효율적으로 줄이지 못하여 트리의 Level별로 자식노드의 비교문을 통한 초기화를 진행하고 있다.

이는 Tree의 Level이 높아지면, 즉 Tree의 구성 노드가 많아질수록 코드의 비교문과 반복문의 횟수가 증가해야 된다는 결론이 나오고 이는 구현상 많은 데이터를 가지는 k개의 런을 구성하기 위한 winner tree를 구성하는것에는 부적합한 구현상의 한계로 볼 수 있다.

소감문 & 한계성

소감문 & 한계성

□ 소감문

선택트리(Seletion Tree)의 K개의 런(Run)으로 데이터를 분할하여 다시 합병(Merging)하는 방식의 승자 & 패자트리를 구현하면서 분할된 데이터 또한 여러 정렬방법에 의해 합병될수 있음을 알수있었고, 이번 구현의 가장 큰 성과는 트리(Tree)를 배열(Array)방식으로 구현했을 때 이론적으로 배운 시각적 그림이 아닌 컴퓨터가 받아들이는 배열구성법에 대해 알수있었으며 이것을 비교문을 통해 자식노드(Children node)에서 부모노드(Parent node)로의 접근이 어떻게 구성되는가를 정립할수 있었다.

또한 승자 & 패자트리를 통해 같은 정렬법임에도 불구하고 비교문의 횟수의 차이와 성능상의 차이를 알수있었으며 이는 효율적인 프로그래밍에 도움이 될것이라 생각된다.

□ 한계성

앞서 제시한 승자 및 패자트리는 최적화 되지않은 비효율적인 방식이라 볼 수 있다. 이는 적은 레벨의 트리임에도 부모-자식 노드의 연결을 위해 수많은 비교문을 이용하고 있으며 이를 기억하기 위한 메모리의 사용은 결국 비효율적으로 다가왔으며 가능한 재귀적인 구현을 통해 접근이 가능한지 살펴볼 필요가 있는 코드다.

* PDF로 제시한 코드중, 헤더파일에 `_SetOfRuns`의 멤버변수인 `topOfRuns[TREE_SIZE]`는 사용하지 않았습니다.

All Source Code

-Loser Tree

Loser Tree(1/5)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define TREE_SIZE 8 // 트리 및 런 사이즈
#define MAX_KEY 10000 // 런에 들어갈수 있는 키의 최대값
typedef struct _treeNode // Selection Tree의 노드
{
    int key; // node의 key
    int idx; // node의 키값이 위치하는 run의 번호
} treeNode;
typedef struct _SelectionTree // Selection Tree 구조체
{
    treeNode tree[TREE_SIZE]; // Tree를 나타낼 배열
} SelectionTree;
typedef struct _SetOfRuns // run들의 구조체
{
    int run[TREE_SIZE][TREE_SIZE];
    //int topOfRuns[TREE_SIZE];  사용 안한 함수
} SetOfRuns;
// Run을 초기화한다.
// run에 임의의 값을 채운다. 각 런마다 sort된 값을 채운다.
void initRuns(SetOfRuns* runs);
// SelectionTree를 초기화 한다.
// 각 런의 첫번째 노드를 가지고 SelectionTree를 구성한다.
void initSelectionTree(SelectionTree* sTree, SetOfRuns* runs);
//SelectionTree를 이용하여 run에서 가장 작은 Key값 하나를 리턴한다.
//동시에 SelectionTree를 재구성한다.
//만약 run에 남아있는 노드가 없다면 MAX_KEY를 리턴한다.
int getWinner(SelectionTree* sTree, SetOfRuns* runs);
//SelectionTree를 출력한다.
void printTree(SelectionTree* sTree);
//현재 run에 남아있는 노드의 키값을 적당한 형태로 출력한다.
void printRuns(SetOfRuns* runs);
```

Loser Tree(2/5)

```
void initRuns(SetOfRuns* runs){
    int i, j, k;
    int temp;
    srand((unsigned)time(NULL));

    for(i=0; i<TREE_SIZE; i++)
    for(j=0; j<TREE_SIZE; j++)
    runs->run[i][j] = rand()%10000; // MAX_KEY 보다 작은 난수를 무작위로 run[] 저장

    /** 버블 정렬에 의한 run[] 오름차순 정렬 **/
    for(k=0; k<TREE_SIZE; k++)
    for(i=0; i<TREE_SIZE; i++)
    for(j=i+1; j<TREE_SIZE; j++){
        if(runs->run[k][i] > runs->run[k][j]){
            temp = runs->run[k][i];
            runs->run[k][i] = runs->run[k][j];
            runs->run[k][j] = temp;
        }
    }
}

/** 최상위 tree[0].key 값을 return(출력) 및 Loser Tree 재구성 **/
int getWinner(SelectionTree* sTree, SetOfRuns* runs){
    int i;
    int temp = sTree->tree[0].key; // initSelectionTree()의 Root 노드를 임시저장
    for(i=0; i<TREE_SIZE-1; i++){ // run[] 재구성
        runs->run[sTree->tree[0].idx][i] = runs->run[sTree->tree[0].idx][i+1];
    }
    // 최상위 Root 노드에 저장된 idx를 이용해 run[idx][7] 값에 MAX_KEY 저장.
    runs->run[sTree->tree[0].idx][TREE_SIZE-1] = MAX_KEY;
    initSelectionTree(sTree, runs); // 재구성된 run[]을 이용한 Loser Tree 재구성
    return temp; // 임시 저장된 temp값 return(출력)
}

/** Expression Tree **/
void printTree(SelectionTree* sTree){
    printf("                %d                %d\n",sTree->tree[0].key);
    printf("                %d                %d\n",sTree->tree[1].key);
    printf("            %d                %d\n",sTree->tree[2].key,sTree->tree[3].key);
    printf("        %d                %d                %d                %d\n",sTree->tree[4].key,sTree->tree[5].key,sTree->tree[6].key,sTree->tree[7].key);
}
```

Loser Tree(3/5)

/* run[] 에 의한 Loser Tree 구성 */

void initSelectionTree(SelectionTree* sTree, SetOfRuns* runs)

```
{
    int i, j;
    int level3=4;    // 단말노드 첫 idx : 4
    int temp[2];     // sTree->tree[0], tree[1] 임시저장 값
    int f_num=0;
    // sTree->tree[i].key & idx = 0 & -1 로 초기화
    for(i=0; i<TREE_SIZE; i++){
        sTree->tree[i].key = 0;
        sTree->tree[i].idx = -1;
    }
    // run[]에 의한 Loser Tree 구성
    for(i=0; i<TREE_SIZE; i+=2){ // i+=2 : 2개의 run[]의 비교로 1개의 tree[]를 구성
        if(runs->run[i][0] > runs->run[i+1][0]){ // 첫 단말노드 구성
            sTree->tree[level3].key = runs->run[i][0]; // 작은 값을 저장
            sTree->tree[level3].idx = i;
            // 부모노드가 채워져있으면서 부모노드가 자식노드보다 작다면
            if(sTree->tree[level3/2].key !=0 && sTree->tree[level3/2].key <
runs->run[i+1][0]){
                sTree->tree[level3/2].key = runs->run[i+1][0]; // 자식노드를 부모노드로
보낸다.
                sTree->tree[level3/2].idx = i+1;
                level3++;
            }
            // 부모노드가 크다면 level3++로 skip
            else if(sTree->tree[level3/2].key != 0 && sTree->tree[level3/2].key >
runs->run[i+1][0]){
                level3++;
            }
            else{ // 부모노드가 비어있을때 자식노드값으로 초기화
                sTree->tree[level3/2].key = runs->run[i+1][0];
                sTree->tree[level3/2].idx = i+1;
                level3++;
            }
        }
    }
}
```

~ 계속

Loser Tree(4/5)

```
else{ // == if(runs->run[i][0] < runs->run[i+1][0]) 이면서, 위와 같은 조건을 취한다.
    sTree->tree[level3].key = runs->run[i+1][0];
    sTree->tree[level3].idx = i+1;
    if(sTree->tree[level3/2].key != 0 && sTree->tree[level3/2].key < runs->run[i][0]){
        sTree->tree[level3/2].key = runs->run[i][0];
        sTree->tree[level3/2].idx = i;
        level3++;
    }
    else if(sTree->tree[level3/2].key != 0 && sTree->tree[level3/2].key >
runs->run[i][0]){
        level3++;
    }
    else{
        sTree->tree[level3/2].key = runs->run[i][0];
        sTree->tree[level3/2].idx = i;
        level3++;
    }
}

// 쓰이지 않은 run[]의 idx 값을 temp[2]에 저장
// temp[2]에 저장된 2개의 정보는, tree[0] or tree[1] 에 채워진다.(최소값 2개)
for(i=0; i<TREE_SIZE; i++){
    for(j=0; j<TREE_SIZE; j++){
        if(i==sTree->tree[j].idx) break;
        else if(j==7) temp[f_num++]=i;
    }
    // temp[2]의 2개의 idx값을 통해, 두개의 run[]을 비교하여 tree[0] or tree[1]에 저장.(idx or .key)
    if(runs->run[temp[0]][0] > runs->run[temp[1]][0]){
        sTree->tree[0].key = runs->run[temp[1]][0];
        sTree->tree[0].idx = temp[1];
        sTree->tree[1].key = runs->run[temp[0]][0];
        sTree->tree[1].idx = temp[0];
    }
    else{
        sTree->tree[0].key = runs->run[temp[0]][0];
        sTree->tree[0].idx = temp[0];
        sTree->tree[1].key = runs->run[temp[1]][0];
        sTree->tree[1].idx = temp[1];
    }
}

void printRuns(SetOfRuns* runs){
    int i, j;
    for(i=0; i<TREE_SIZE; i++){
        for(j=0; j<TREE_SIZE; j++){
            printf("%d\t\t", runs->run[j][i]);
        }
        printf("\n"); }
    printf("\n"); }
```

Loser Tree(5/5)

```
void main()
{
    SelectionTree sTree;
    SetOfRuns runs;
    int i,j;
    // Runs, SelectionTree의 초기화
    initRuns(&runs);
    initSelectionTree(&sTree, &runs);
    // 초기 SelectionTree/Run 상태 출력
    printf("===== SelectionTree View =====Wn");
    printTree(&sTree);
    printRuns(&runs);
    // 정렬된 순서로 run의 레코드를 모두 출력
    printf("=====Wn");
    j=0;
    // 실행결과 1번
    /*
    while( (i = getWinner(&sTree, &runs)) != MAX_KEY)
    {
        printf("%4d ", i);
        if(j%10==9)
            printf("Wn");

        j++;
    }
    printf("Wn");
    */
    //실행 결과 2번 : 32번 getWinner 출력 후 Tree & Runs 출력

    for(i=0; i<32; i++){
        printf("%4d ",getWinner(&sTree, &runs));
        if(i%10==9) printf("Wn");
    }
    printf("Wn");
    printf("===== SelectionTree View =====Wn");
    printTree(&sTree);
    printRuns(&runs);
}
```


All Source Code

-Winner Tree

* void initSelectionTree() 외 Loser Tree와 동일

Winner Tree(1/1)

```
/* run[] 에 의한 Winner Tree 구성 */
void initSelectionTree(SelectionTree* sTree, SetOfRuns* runs){
    int i;
    int level3 = 4; // 단말노드 첫 idx : 4
    // sTree->tree[i].key & idx = 0 & -1 로 초기화
    for(i=0; i<TREE_SIZE; i++){
        sTree->tree[i].key = 0;
        sTree->tree[i].idx = -1;
    }
    // run[]에 의한 Winner Tree 구성
    // 단말노드(Terminal node)를 초기화
    for(i=0; i<TREE_SIZE; i+=2){
        if(runs->run[i][0] < runs->run[i+1][0]){
            sTree->tree[level3].key = runs->run[i][0];
            sTree->tree[level3].idx = i;
            level3++;
        }
        else{
            sTree->tree[level3].key = runs->run[i+1][0];
            sTree->tree[level3].idx = i+1;
            level3++;
        }
    }
    // 단말노드(Terminal node), 즉 자식노드(children)간의 비교를 통해 부모노드 초기화
    for(i=4; i<TREE_SIZE; i+=2){
        if(sTree->tree[i].key < sTree->tree[i+1].key){
            sTree->tree[i/2].key = sTree->tree[i].key;
            sTree->tree[i/2].idx = sTree->tree[i].idx;
        }
        else{
            sTree->tree[i/2].key = sTree->tree[i+1].key;
            sTree->tree[i/2].idx = sTree->tree[i+1].idx;
        }
    }
    // 자식노드(children)간의 비교를 통해 부모노드 초기화
    if(sTree->tree[2].key < sTree->tree[3].key){
        sTree->tree[1].key = sTree->tree[2].key;
        sTree->tree[1].idx = sTree->tree[2].idx;
    }
    else{
        sTree->tree[1].key = sTree->tree[3].key;
        sTree->tree[1].idx = sTree->tree[3].idx;
    }
    // 헤더파일에 제시된 tree[]의 갯수를 채우기 위해 tree[0]를 쓰고있다.
    sTree->tree[0].key = sTree->tree[1].key;
    sTree->tree[0].idx = sTree->tree[1].idx;
}
```