Optimal Binary Search Tree 보고서

201504280
컴퓨터공학과 신윤호

1. 구현 내용

1) 초기화

주어진 형식의 데이터를 읽어 p, q테이블을 초기화하도록 했다. p, q테이블은 각각 dataTableP, dataTableQ라는 이름의 ArrayList로 구현했다. 모든 데이터를 읽어오면 키 노드들의 개수를 알 수 있다. 이를 size라는 변수에 저장하고 size를 토대로 w(i,j), e(i,j), root(i,j) 테이블을 생성했다.

각 테이블의 역할은 다음과 같다. 먼저 e(i,j)는 k_i부터 k_j까지의 키 노드를 사용한다고 했을 때 발생 빈도의 최소값을 저장한다. w(i,j)는 k_i부터 k_j 까지 키 노드를 사용한 트리에서, 각 k에 대한 p의 총합을 나타낸다. w 테이블을 이용함으로써 e(i,j)를 계산할 때 마다 w(i,j)를 처음부터 다시 계산할 필요가 없어진다. 마지막으로 root(i,j)는 k_i부터 k_j를 포함하는 서브트리의 루트를 저장하는데 사용된다.

2) findOptimalTree 메소드

Optimal BST를 찾아내는 메소드다. 먼저 for문을 이용해 e와 w를 초기화해 주었다. 주어진 데이터가 다음과 같을 경우 두 배열은 아래와 같이 초기화된다.

input

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

e table

0.05	0.00	0.00	0.00	0.00	0.00
0.00	0.10	0.00	0.00	0.00	0.00
0.00	0.00	0.05	0.00	0.00	0.00
0.00	0.00	0.00	0.05	0.00	0.00
0.00	0.00	0.00	0.00	0.05	0.00
0.00	0.00	0.00	0.00	0.00	0.10

w table

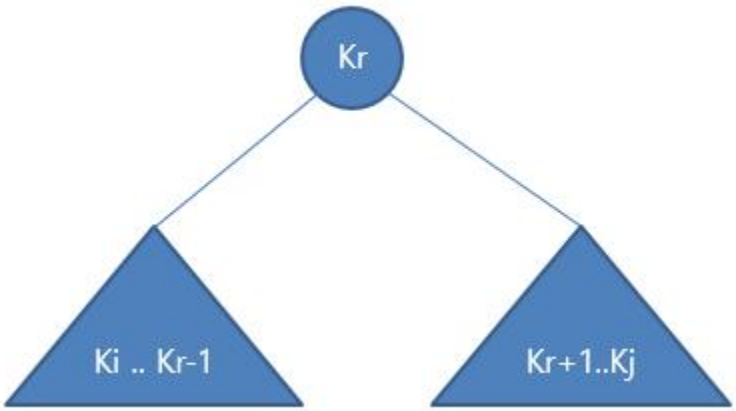
0.05	0.00	0.00	0.00	0.00	0.00
0.00	0.10	0.00	0.00	0.00	0.00
0.00	0.00	0.05	0.00	0.00	0.00
0.00	0.00	0.00	0.05	0.00	0.00
0.00	0.00	0.00	0.00	0.05	0.00
0.00	0.00	0.00	0.00	0.00	0.10

그 후, 삼중 반복문을 돌며 알고리즘이 수행된다. 외부의 두 반복문에 따라 $1 \leq i \leq j \leq n$에 대해 e[i][j], w[i][j]를 계산한다. 첫 번째 루프에서는 $i = 1$에서부터 $i = n$까지, e[i][j], w[i][j]를 계산하고, 두 번째 루프에서는 $i = 1$부터 $i = n - 1$까지 e[i][i+1], w[i][i+1]을 계산하는 식이다. 아래와 같은 과정으로 계산이 수행되는 것이다.

e table

0.05	0.00	0.00	0.00	0.00	0.00
0.00	0.10	0.00	0.00	0.00	0.00
0.00	0.00	0.05	0.00	0.00	0.00
0.00	0.00	0.00	0.05	0.00	0.00
0.00	0.00	0.00	0.00	0.05	0.00
0.00	0.00	0.00	0.00	0.00	0.10

또한 각 루프의 가장 내부 for문은 어떤 k_i부터 k_j중 어떤 키 노드가 루트가 될 것인지를 결정한다. 이진 탐색 트리의 특성을 고려했을 때, k_i~k_j 사이의 임의의 키 노드 k_r을 루트로 사용하는 상황은 아래와 같다.



즉, 이진 탐색 트리이므로 k_r을 기준으로 왼쪽 서브트리는 k_r보다 작은 값으로, 오른쪽 서브트리는 k_r보다 큰 값으로 구성될 것이다. 최적 이진 검색 트리 문제에서 각각의 키들은 오름차순으로 정렬되어 있다고 가정하고 있으므로, 이 경우 왼쪽 서브트리는 k_i~k_{r-1}의 노드로, 오른쪽 서브트리는 k_{r+1}~k_j로 구성되어 있을 것이다. 위 상황을 가정해 k_r을 루트로 사용할 때의 비용을 구하는 식은 과제에서도 주어졌듯 e[i,j]=e[i,r-1]+e[r+1,j]+w(i,j)이다. 가장 내부의 반복문은 k_i부터 k_j사이의 모든 노드들에 대해 해당 노드가 k_r이 되는 경우를 가정하고, 각각의 비용들을 계산해 그중 비용이 최소가 되는 루트를 k_r로 확정한 뒤, root테이블에 기록하는 역할을 수행한다.

최종적으로 만들어지는 각 테이블의 모습은 다음과 같으며, 이를 토대로 Optimal BST를 그려보면 아래와 같다.

e table :

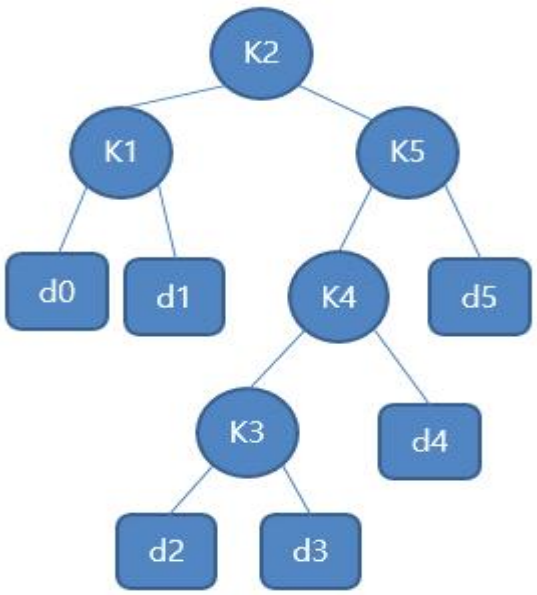
0.05	0.45	0.90	1.25	1.75	2.75
0.00	0.10	0.40	0.70	1.20	2.00
0.00	0.00	0.05	0.25	0.60	1.30
0.00	0.00	0.00	0.05	0.30	0.90
0.00	0.00	0.00	0.00	0.05	0.50
0.00	0.00	0.00	0.00	0.00	0.10

w table :

0.05	0.30	0.45	0.55	0.70	1.00
0.00	0.10	0.25	0.35	0.50	0.80
0.00	0.00	0.05	0.15	0.30	0.60
0.00	0.00	0.00	0.05	0.20	0.50
0.00	0.00	0.00	0.00	0.05	0.35
0.00	0.00	0.00	0.00	0.00	0.10

root table :

0	1	1	2	2	2
0	0	2	2	2	4
0	0	0	3	4	5
0	0	0	0	4	5
0	0	0	0	0	5



2. 결과 화면

```
Optimal BST cost : 2.75

e table :
0.05        0.45        0.90        1.25        1.75        2.75
0.00        0.10        0.40        0.70        1.20        2.00
0.00        0.00        0.05        0.25        0.60        1.30
0.00        0.00        0.00        0.05        0.30        0.90
0.00        0.00        0.00        0.00        0.05        0.50
0.00        0.00        0.00        0.00        0.00        0.10

w table :
0.05        0.30        0.45        0.55        0.70        1.00
0.00        0.10        0.25        0.35        0.50        0.80
0.00        0.00        0.05        0.15        0.30        0.60
0.00        0.00        0.00        0.05        0.20        0.50
0.00        0.00        0.00        0.00        0.05        0.35
0.00        0.00        0.00        0.00        0.00        0.10

root table :
0       1       1       2       2       2
0       0       2       2       2       4
0       0       0       3       4       5
0       0       0       0       4       5
0       0       0       0       0       5

Process finished with exit code 0
```