

Data Structure

Project 2

학번: 2020202054

이름: 이종혁

학과: 컴퓨터정보공학부

수강과목: 데이터구조설계(월6/수5)

데이터구조실습(수7)

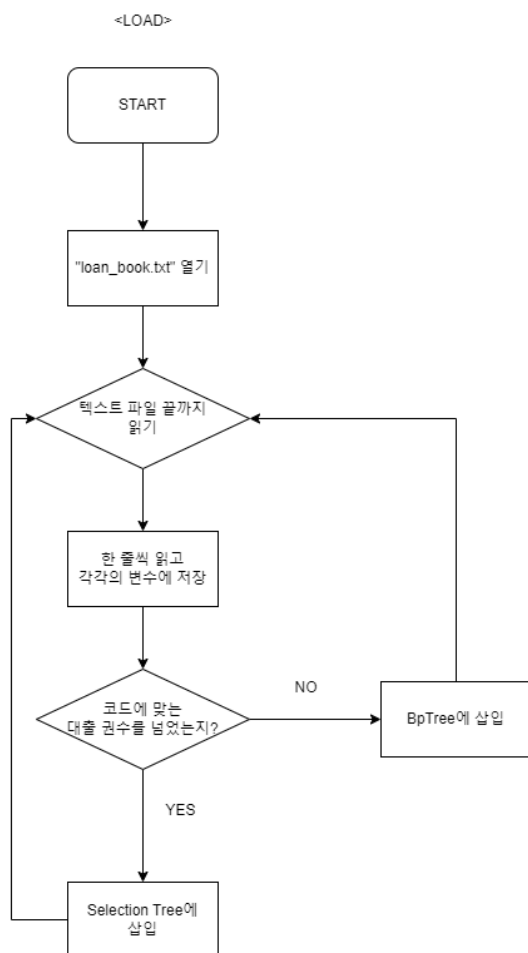
- Introduction

이번 프로젝트는 도서대출정보관리 프로그램을 구현해야 합니다. 크게는 총 B+Tree, Selection Tree, 그리고 Heap 까지 3 가지의 자료구조가 서로 데이터를 주고받는 구조로 이루어져 있습니다.

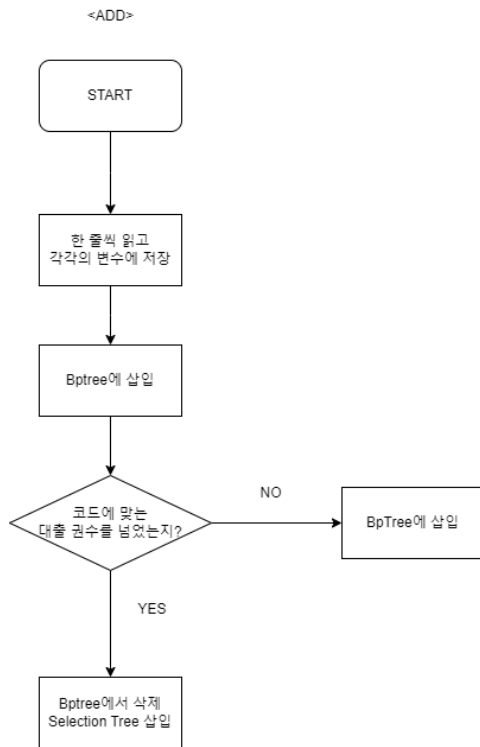
우선 LOAD 나 ADD 명령어를 통해, 도서의 정보를 추가합니다. 도서의 정보로는 이름/분류코드/저자/발행연도/대출권수로 이루어져 있고, 분류코드 별로 대출할 수 있는 도서의 수가 정해져 있습니다. 우선 대출이 현재 가능한 상태의 도서라면, B+Tree 에 존재합니다. 그리고 만약 더 이상 대출할 수 없는 상태가 된다면(ADD 로 계속해서 대출 수가 추가되거나, LOAD 시 초기에 이미 기준이 되었을 때) Selection Tree 에 삽입되게 됩니다. 추가로 Selection Tree 의 모든 Leaf Node 의 Run 은 각각 Heap 으로 나누어져 있고, Heap 인 Run 들은 모두 같은 분류 코드끼리 이루어집니다.

- Flowchart

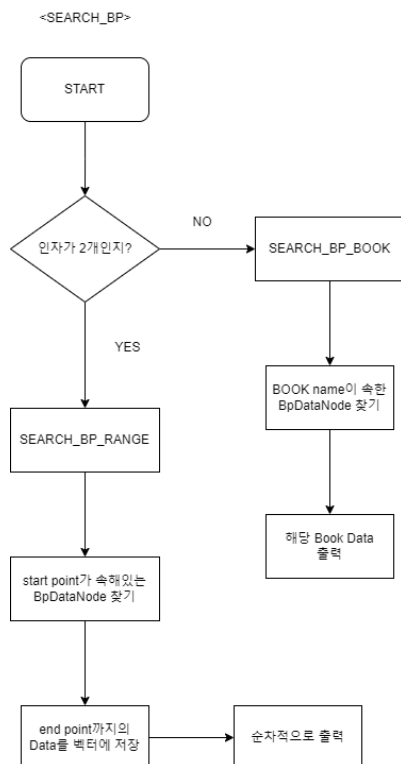
명령어별로 플로우차트를 그렸습니다.



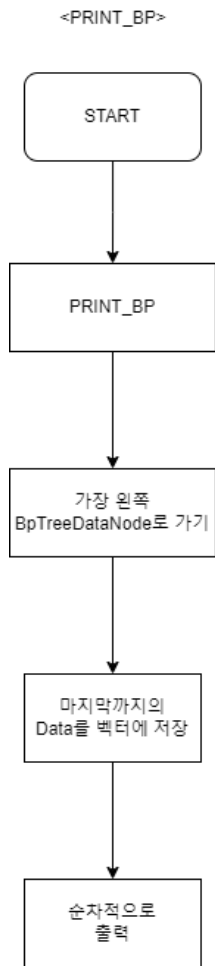
LOAD 시의 데이터를 읽고, 각각의 변수로 저장한 후에 대출불가 여부를 loan_count 로 판별 후, 대출 불가능한 경우에는 바로 Selection Tree 로 넣었습니다.



ADD 에 경우 LOAD 와 흡사하나, command 한 줄만 읽고 해당 데이터를 우선 BpTree 에 넣었습니다. 그 후 이미 존재하는 데이터인지 아닌지를 판별하였고, 이미 존재하는 경우 대출 불가 도서인지를 확인 후 Bptree 에서 삭제하고 Selection Tree 에 삽입했습니다.

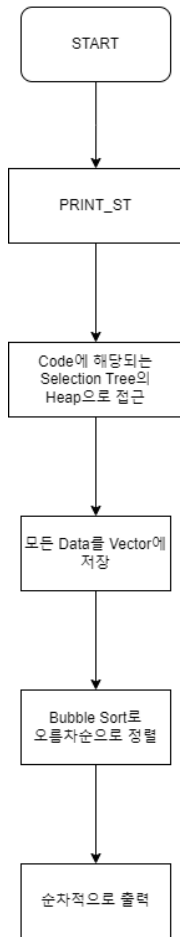


SEARCH_BP 는 인자가 몇 개 들어왔는지를 판별한 후에, 각각의 알맞은 함수를 호출하여 Bptree 의 데이터를 출력하였습니다.



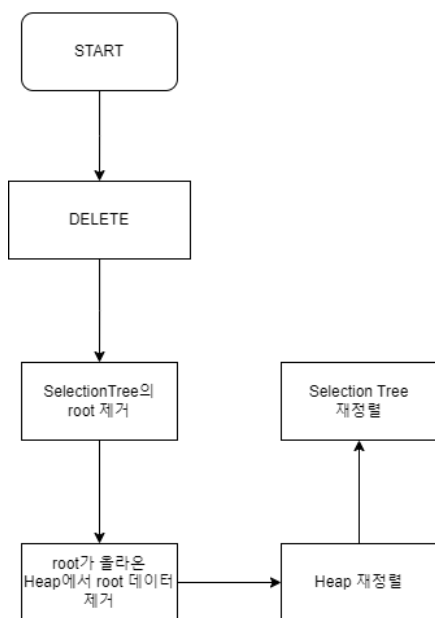
PRINT_BP 는 가장 처음에 있는 DataNode 로 접근해서, DataNode 끝까지 순차적으로 출력하였습니다.

<PRINT_ST>



PRINT_ST 에 경우, 해당하는 코드에 맞는 Selection Tree 의 Run 인 Heap 에 접근하여 출력하였습니다.

<DELETE>



DELETE 에 경우, SelectionTree 의 Root 에 해당하는 Run 으로 접근하여 해당 Heap 에서 그 데이터를 삭제하고 재정렬 후, Selection Tree 를 재정렬했습니다.

B: A21 Data

명시된 것처럼 K 는 각 노드를 연결하는 포인터, A 는 대소비교의 기준이 되는 Key 값, B 는 A 의 Data 입니다. 우선 Data Node 의 split 은 두 가지 경우로 나누었습니다. 만약 DataNode 의 부모가 없는 상태에서 Split 을 해야 한다면, 즉 BpTree 에 DataNode 만 존재하고 해당 노드가 root 라면 부모 Node 와 데이터를 나눠서 담을 Node 를 선언합니다.

여기서 보면, map 의 특성에 의해서 이미 Node 내의 Map Data 들은 크기 순으로 정렬되어 있기 때문에, 크기 상 가운데에 있는 것을 부모로 올린다고 생각하면 됩니다.

하지만 DataNode 의 부모는 무조건 IndexNode 이기 때문에, Key 값과 그 Key 값의 데이터로 이루어진 DataNode 와는 다르게 Key 값과 다른 Node 를 가리키는 포인터로 이루어져야 한다는 것을 알아야 합니다. 이러한 개념을 갖고 부모로 올려주면 됩니다.

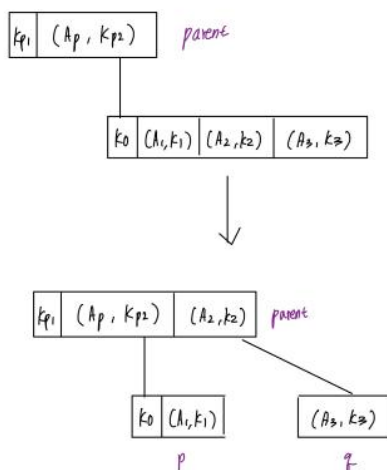
또한, Index Node 와 다르게 부모로 올라가면 자식에서는 지우는 것이 아니라, DataNode 는 모든 데이터에 대해서 포함하고 있기에, 새로 나눠 담을 노드인 q 에 2 번째와 3 번째 데이터를 복사해줍니다. 그리고 원래 기존에 있던 p 에는 1 번째 데이터만 남게 하면 됩니다.

그 후 부모와 자식의 관계 설정을 그림과 같이 해주고, DataNode 끼리도 Doubly Linked List 로 관계를 설정해주면 Split 이 끝납니다.(설정 간에 Split 된 DataNode 가 기존 DataNode 들의 사이에 자리를 잡을 수도 있기에, 이러한 경우도 고려해야 합니다.)

2 번의 경우, 부모 노드를 생성하는 것이 아니라, 기존에 존재하던 부모 Index Node 에 두번째 Data 를 넘겨주면 된다는 차이가 있습니다.

<Index Node Split>

<Index Node split>



K: 각 노드를 연결하는 Pointer

A: 기준이 되는 key 값

B: A의 Data

IndexNode 에 경우, DataNode 와 가장 큰 차이점이라면, Map 을 구성하는 Data 종류와 부모로 올려주는 방식에 있습니다.

우선 IndexNode 는 앞서 언급한 것과 같이, 기준이 되는 Key 값과 다른 Node 를 가리키는 포인터로 이루어져 있습니다.

또한 부모로 데이터를 올리면, 올라간 Data 는 더 이상 자식에 남지 않습니다. 그래서 DataNodeSplit 시 q 와, IndexNodeSplit 시 q 를 보면, 확연한 차이를 느낄 수 있습니다.

<Selection Tree 재정렬>

Selection Tree 는 쉽게 생각해서 토너먼트와 같습니다. 그래서 Leaf Node 에서부터 기준에 의해 정해진 Data 가 점점 올라와서 Root 까지 차지하는 것입니다. 그렇기 때문에, 만약 Selection Tree 에 재정렬이 필요한 경우, 가장 밑(Leaf Node)에서부터 서로의 형제들과 비교를 해서, root 까지 반복해서 승자를 결정해서 재정렬을 하면 되는 것입니다.

<Heap 재정렬 Up/Down>

Heap, 특히 이번 과제에서 사용된 Min Heap 에 경우, **부모가 자식보다 무조건 값이 작은** 관계를 가진 자료 구조입니다.

그렇기 때문에, Selection Tree 와 흡사하게 밑에서부터 위로 재정렬할 수도, 추가로 Delete 구현을 위해 위에서부터 밑으로 재정렬하는 경우도 있습니다. 하지만 Selection Tree 와 Heap 의 가장 큰 차이점은, Selection Tree 는 토너먼트 개념으로 올라가는 것이기 때문에 **값을 복사**하는 방식이라면, Heap 은 부모와 자식의 관계만을 생각하여 기준에 맞지 않은 경우 서로 **값을 교환**하는 방식입니다.

아래에서부터 위로 Heap 을 재정렬하는 것은 Insert 에서 쓰입니다. Heap 의 Insert 는 가장 마지막에 추가된 Node 의 다음 위치에 새로운 Node 가 들어가게 됩니다. 그 상태에서 가장 밑에 기준에 부합하는지 아닌지 모르는 Node 가 들어왔기 때문에, 반복문을 통해 아래에서부터 위로 재정렬을 하게 됩니다.

반대로 Delete 시에는 root 를 제거하고, 마지막에 추가되었던 Node 를 root 로 들여와서 재정렬을 해야 합니다. 그래서 그 때에는 위에서 아래로 기준이 맞을 때까지 Swap 하며 내려가게 됩니다.

여기서 중요한 것은, 아래에서 위로 올라갈 때에는 단순히 부모로만 가면 되기 때문에 상관없이 없었지만, Min Heap 에 조건에 맞게 하기 위해 항상 더 작은 자식과 비교를 하도록 조건을 추가하면 됩니다.

- Result Screen


```

1  abc 000 nd 2000 3
2  pciv 300 Nf 1985 1
3  knfu tmlf 600 Vh 2248 1
4  euwc okat 700 NkEjKj 2132 1
5  klim 500 lo Sd 2088 2
6  oecI 300 Vo 2033 1
7  qamI mjxz 100 PoZbTp 2252 1
8  gvdK 500 VdUxWl 1958 1
9  pdef 600 TcSk Yo 2273 0
10 ctdd 100 VpKr 2281 0
11 cadz jfhD 400 Lp El 2026 0
12 ydxb mxko 700 Sf 2187 2
13 vdze ykxv 100 Cz Cx Mq 1922 1
14 xmf 400 Ew Bt Fd 1959 4
15 udak 000 Bg 1984 1
16 cwdo 000 Fi Lm 2029 0
17 eey 300 Yu Aa Mv 2088 3
18 dbzl 000 Ji Iv 1942 0
19 fkse 700 Ws Om Vk 2138 2
20 vfix 100 Cm 1917 1
21 wsuf 000 Hv Al 2243 2
22 plof 000 Fx 1925 0
23 ojgo 700 Nt Bl Tr 1988 1
24 eked 300 dne 1922 1
25 msdg 000 Ke Xt 1944 0
26 kchk 000 Hg 2252 3
27 brdj 400 Al So Uw 1905 0
28 ahgi 200 Dk 2127 1
29 gaiv 300 Jh Zg Fo 2102 1
30 hnek 500 Uw Tf Rk 2228 1
31 czjn 700 Zq Le Zb 2153 1
32 like 100 Bi Pw 2333 1
33 vfna 500 Ue 2188 1
34 pjop 100 He 2173 1
35 lbbj 400 Hh 2343 1
36 escawci 500 Lv Ao 2237 0
37 wnyz ukte 000 Tu 2108 0
38 jj fo kdgi 600 lu Ng De 1957 0
39 askx 300 Sa Ir Gc 1929 1
40 azhn 500 De Wv Dk 1983 0
41 arrpqxy 000 Bc Kv Ti 1988 0
42 jebg 300 Sb Ov Mz 2331 1
43 txbt 000 Gm Ca 2237 1
44 hzyd 400 Ng Zk Zz 2343 1
45 loku 300 Tc 2190 4
46 ol vfmu 400 Ia 2279 1
47 ruz ulmb 100 Xp 2288 2
48 xbm 400 Qo 2258 1
49 yuqu 000 Qh 2072 0
50 tmb dogo 100 Kc Lp Ja 2271 0
51 boy hnni 200 Rf 1999 1
52 jmlo 300 Fr 2340 0

```

이 파일이 loan_book.txt입니다.

도서 분류 코드	대출 불가 도서
000	abc / kchk
100	
200	
300	loku
400	xmf
500	klim
600	
700	ybxb mxko / fkse

위 표는 LOAD 시 이미 대출불가하기에 SelectionTree에 옮겨진 데이터들입니다.

```
1 LOAD
2 PRINT_ST 000
3 PRINT_ST 100
4 PRINT_ST 200
5 PRINT_ST 300
6 PRINT_ST 400
7 PRINT_ST 500
8 PRINT_ST 600
9 PRINT_ST 700
10
11 ADD knfu tmlf 600 Vh 2246
12 ADD seey 300 Yu Aa Mv 2086
13 SEARCH_BP book
14 ADD book 200 risa 1990
15 ADD book 200 risa 1990
16 SEARCH_BP book
17 ADD book 200 risa 1990
18 SEARCH_BP book
19 SEARCH_BP a b
20 SEARCH_BP a e
21 SEARCH_BP A Z
22 SEARCH_BP a z
23 PRINT_BP
24 PRINT_ST 000
25 PRINT_ST 100
26 PRINT_ST 200
27 PRINT_ST 300
28 PRINT_ST 400
29 PRINT_ST 500
30 PRINT_ST 600
31 PRINT_ST 700
32 PRINT_ST 800
33 DELETE
34 PRINT_ST 000
35 PRINT_ST 100
36 PRINT_ST 200
37 PRINT_ST 300
38 PRINT_ST 400

===== LOAD =====
2 Success
3 =====
4
5 =====PRINT_ST=====
6 abc/000/nd/2000/3
7 kchk/000/Hg/2252/3
8 =====
9
10 =====ERROR=====
11 500
12 =====
13
14 =====ERROR=====
15 500
16 =====
17
18 =====PRINT_ST=====
19 loku/300/Tc/2190/4
20 =====
21
22 =====PRINT_ST=====
23 :xmwf/400/Ew Bt Fd/1959/4
24 =====
25
26 =====PRINT_ST=====
27 kl im/500/Io Sd/2066/2
28 =====
29
30 =====ERROR=====
31 500
32 =====
33
34 =====PRINT_ST=====
35 fkse/700/Ws Om Vk/2136/2
36 ydxb mxko/700/St/2187/2
37 =====
38
```

그래서 LOAD 성공 후, 각각의 Selection Tree에 대해 출력을 하면, 다음과 같이 나오게 됩니다.

(빨간 선의 위까지 결과입니다.)

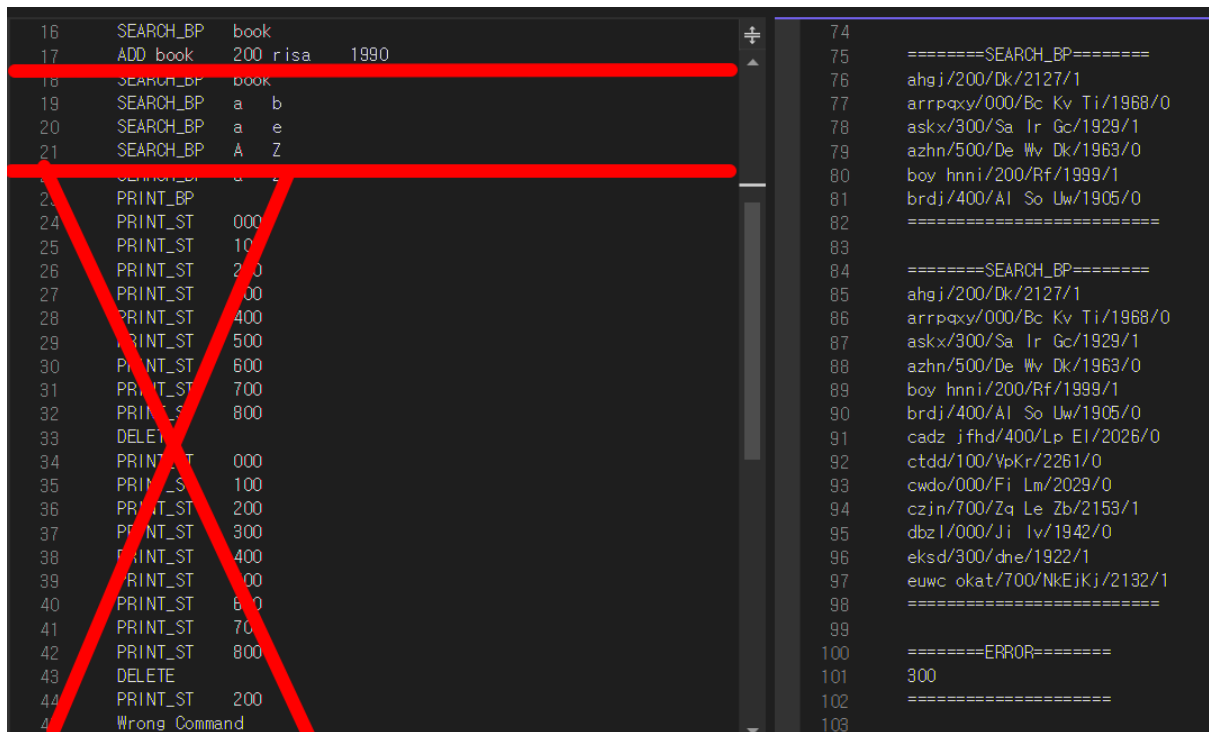
```
7 PRINT_ST 500
8 PRINT_ST 600
9 PRINT_ST 700
10 ADD book 200 risa 1990
11 ADD knfu tmlf 600 Vh 2246
12 ADD seey 300 Yu Aa Mv 2086
13 SEARCH_BP book
14 ADD book 200 risa 1990
15 ADD book 200 risa 1990
16 SEARCH_BP book
17 ADD book 200 risa 1990
18 SEARCH_BP book
19 SEARCH_BP a b
20 SEARCH_BP a e
21 SEARCH_BP A Z
22 SEARCH_BP a z
23 PRINT_BP
24 PRINT_ST 000
25 PRINT_ST 100
26 PRINT_ST 200
27 PRINT_ST 300
28 PRINT_ST 400
29 PRINT_ST 500
30 PRINT_ST 600
31 PRINT_ST 700
32 PRINT_ST 800
33 DELETE
34 PRINT_ST 000
35 PRINT_ST 100
36 PRINT_ST 200
37 PRINT_ST 300
38 PRINT_ST 400
39 PRINT_ST 500
40 PRINT_ST 600
41 PRINT_ST 700
42 PRINT_ST 800
43 DELETE

=====ADD=====
39
40 book/200/risa/1990
41 =====
42
43 =====ADD=====
44 knfu tmlf/600/Vh/2246
45 =====
46
47 =====ADD=====
48 seey/300/Yu Aa Mv/2086
49 =====
50
51 =====SEARCH_BP=====
52 book/200/risa/1990/0
53 =====
54
55 =====ADD=====
56 book/200/risa/1990
57 =====
58
59 =====ADD=====
60 book/200/risa/1990
61 =====
62
63 =====SEARCH_BP=====
64 book/200/risa/1990/2
65 =====
66
67 =====ADD=====
68 book/200/risa/1990
69 =====
70
71 =====ERROR=====
72 300
73 =====
74
```

ADD 부분을 확인해보면, book이라는 데이터가 최초로 들어왔고, 카운트가 아직 0이기에

SEARCH_BP시 결과가 나오게 됩니다. (카운트 0) 또한 seey와 knfu tmlf라는 두 가지 데이터는 LOAD시 추가되었는데, ADD를 한 번씩 함으로써 대출 권수가 추가되어 Selection Tree로 옮겨졌습니다.

book을 ADD를 두 번 더 하고 SEARCH_BP를 해보면 카운트도 2로 맞게 출력이 됩니다. 그 상태에서 한 번 더 ADD를 하면 카운트 기준이 되어서 BPtrree에서 삭제되었음을 확인할 수 있습니다.



```
16 SEARCH_BP book
17 ADD book 200 risa 1990
18 SEARCH_BP book
19 SEARCH_BP a b
20 SEARCH_BP a e
21 SEARCH_BP A Z
22 SEARCH_BP a z
23 PRINT_BP
24 PRINT_ST 000
25 PRINT_ST 100
26 PRINT_ST 200
27 PRINT_ST 300
28 PRINT_ST 400
29 PRINT_ST 500
30 PRINT_ST 600
31 PRINT_ST 700
32 PRINT_ST 800
33 DELETE
34 PRINT_ST 000
35 PRINT_ST 100
36 PRINT_ST 200
37 PRINT_ST 300
38 PRINT_ST 400
39 PRINT_ST 500
40 PRINT_ST 600
41 PRINT_ST 700
42 PRINT_ST 800
43 DELETE
44 PRINT_ST 200
45 Wrong Command

74
75 =====SEARCH_BP=====
76 ahgj/200/Dk/2127/1
77 arrpxy/000/Bc Kv Ti/1968/0
78 askx/300/Sa Ir Gc/1929/1
79 azhn/500/De Wv Dk/1963/0
80 boy hnni/200/Rf/1999/1
81 brdj/400/Al So Uw/1905/0
82 =====
83
84 =====SEARCH_BP=====
85 ahgj/200/Dk/2127/1
86 arrpxy/000/Bc Kv Ti/1968/0
87 askx/300/Sa Ir Gc/1929/1
88 azhn/500/De Wv Dk/1963/0
89 boy hnni/200/Rf/1999/1
90 brdj/400/Al So Uw/1905/0
91 cadz jfhd/400/Lp El/2026/0
92 ctdd/100/YoKr/2261/0
93 cwdo/000/Fi Lm/2029/0
94 czjn/700/Zq Le Zb/2153/1
95 dbzl/000/Ji Iv/1942/0
96 eksd/300/dne/1922/1
97 euwc okat/700/NkEjKj/2132/1
98 =====
99
100 =====ERROR=====
101 300
102 =====
103
```

이제 SEARCH_BP_RANGE를 테스트해보면, 입력된 범위에 대해서 잘 출력이 되고 있습니다. 또한 책 제목들은 소문자로만 이루어져 있기 때문에 A부터 Z까지의 출력은 아무것도 할 것이 없어, ERROR문이 출력됩니다.

```

103 -----SEARCH_BP-----
104
105 ahgj/200/Dk/2127/1
106 arrpqxy/000/Bc Kv Ti/1968/0
107 askx/300/Sa Ir Gc/1929/1
108 azhn/600/De Wv Dk/1963/0
109 boy hnni/200/Rf/1999/1
110 brdj/400/Al So Uw/1906/0
111 cadz jfhhd/400/Lp El/2026/0
112 ctdd/100/VpKr/2261/0
113 cwdo/000/Fi Lm/2029/0
114 czjn/700/Zq Le Zb/2153/1
115 dbzl/000/Ji Iv/1942/0
116 ekad/300/dne/1922/1
117 euwc okat/700/NkEjKj/2132/1
118 gaiv/300/Jh Zg Fo/2102/1
119 gvdK/600/VdUxWl/1956/1
120 hnek/600/Uw Tf Rk/2226/1
121 hzyd/400/Ng Zk Zz/2343/1
122 jebg/300/Sb Ov Mz/2331/1
123 jj fo kdgi/600/Iu Ng De/1957/0
124 jmlO/300/Fr/2340/0
125 lbbj/400/Hh/2343/1
126 like/100/Bi Pw/2333/1
127 msdg/000/Ke Xt/1944/0
128 oecI/300/Vo/2033/1
129 ojQo/700/Nt BI Tr/1988/1
130 ol vfmU/400/Ia/2279/1
131 pciv/300/Nf/1965/1
132 pdef/600/TcSk Yo/2273/0
133 pjop/100/Hc/2173/1
134 plof/000/Fx/1925/0
135 qamI mjxz/100/PoZbTp/2262/1
136 ruz ulmb/100/Xp/2288/2
137 sscawci/600/Lv Ao/2237/0
138 tmb dCgp/100/Kc Lp Ja/2271/0
139 txbt/000/Gm Ca/2237/1
140 udak/000/Bg/1984/1
141 vdze ykxv/100/Cz Cx Mq/1922/1
142 vfix/100/Cm/1917/1
143 vfna/600/Ue/2186/1
144 wnyz ukte/000/Tu/2106/0
145 wsuf/000/Hv Al/2243/2
146 xbmG/400/Qo/2258/1
147 yuqu/000/Qh/2072/0
148 -----

```

```

150 -----PRINT_BP-----
151
152 ahgj/200/Dk/2127/1
153 arrpqxy/000/Bc Kv Ti/1968/0
154 askx/300/Sa Ir Gc/1929/1
155 azhn/600/De Wv Dk/1963/0
156 boy hnni/200/Rf/1999/1
157 brdj/400/Al So Uw/1906/0
158 cadz jfhhd/400/Lp El/2026/0
159 ctdd/100/VpKr/2261/0
160 cwdo/000/Fi Lm/2029/0
161 czjn/700/Zq Le Zb/2153/1
162 dbzl/000/Ji Iv/1942/0
163 ekad/300/dne/1922/1
164 euwc okat/700/NkEjKj/2132/1
165 gaiv/300/Jh Zg Fo/2102/1
166 gvdK/600/VdUxWl/1956/1
167 hnek/600/Uw Tf Rk/2226/1
168 hzyd/400/Ng Zk Zz/2343/1
169 jebg/300/Sb Ov Mz/2331/1
170 jj fo kdgi/600/Iu Ng De/1957/0
171 jmlO/300/Fr/2340/0
172 lbbj/400/Hh/2343/1
173 like/100/Bi Pw/2333/1
174 msdg/000/Ke Xt/1944/0
175 oecI/300/Vo/2033/1
176 ojQo/700/Nt BI Tr/1988/1
177 ol vfmU/400/Ia/2279/1
178 pciv/300/Nf/1965/1
179 pdef/600/TcSk Yo/2273/0
180 pjop/100/Hc/2173/1
181 plof/000/Fx/1925/0
182 qamI mjxz/100/PoZbTp/2262/1
183 ruz ulmb/100/Xp/2288/2
184 sscawci/600/Lv Ao/2237/0
185 tmb dCgp/100/Kc Lp Ja/2271/0
186 txbt/000/Gm Ca/2237/1
187 udak/000/Bg/1984/1
188 vdze ykxv/100/Cz Cx Mq/1922/1
189 vfix/100/Cm/1917/1
190 vfna/600/Ue/2186/1
191 wnyz ukte/000/Tu/2106/0
192 wsuf/000/Hv Al/2243/2
193 xbmG/400/Qo/2258/1
194 yuqu/000/Qh/2072/0
195 -----

```

```

22 SEARCH_BP a z
23 PRINT_BP

```

모든 경우에 대해 출력하는 PRINT_BP와, SEARCH_BP로 전 범위를 출력해보았을 때, 동일한 출력이 되는 것을 확인할 수 있습니다.

```

196      =====PRINT_ST=====
197      abc/000/nd/2000/3
198      kchk/000/Hg/2252/3
199      =====
200
201      =====ERROR=====
202      500
203      =====
204
205      =====PRINT_ST=====
206      book/200/r isa/1990/3
207      =====
208
209      =====PRINT_ST=====
210      loku/300/Tc/2190/4
211      seey/300/Yu Aa Mv/2086/4
212      =====
213
214      =====PRINT_ST=====
215      xmwf/400/Ew Bt Fd/1959/4
216      =====
217
218      =====PRINT_ST=====
219      klim/500/lo Sd/2066/2
220      =====
221
222      =====PRINT_ST=====
223      knfu tmlf/600/Vh/2246/2
224      =====
225
226      =====PRINT_ST=====
227      fkse/700/Ws Om Vk/2136/2
228      ydxb mxko/700/Sf /2187/2
229      =====
230
231      =====ERROR=====
232      500
233      =====
234

```

```

24      PRINT_ST      000
25      PRINT_ST      100
26      PRINT_ST      200
27      PRINT_ST      300
28      PRINT_ST      400
29      PRINT_ST      500
30      PRINT_ST      600
31      PRINT_ST      700
32      PRINT_ST      800

```

도서 분류 코드	대출 불가 도서
000	abc / kchk
100	
200	book
300	loku / seey
400	xmwf
500	klim
600	knfu tmlf
700	ybxb mxko / fkse

이제 ADD와 LOAD를 마치고 다시 Selection Tree를 모두 출력해보았습니다.

파란색은 ADD로 인해 추가되었다고 예상한 책의 이름이고, 결과를 확인해보면 잘 추가

되었음을 확인할 수 있습니다. 또한 대출 가능 권수도 코드 별로 알맞게 적용되었습니다. 마지막으로 옳지 않은 코드인 800을 입력했을 때는 오류가 출력됩니다.

여기서 확인이 가능한 것은 사전 순으로 정렬되어 있기 때문에, Selection Tree의 root는 "abc"라는 이름을 가진 데이터입니다.

```

235      ===== DELETE =====
236      Success
237      =====
238
239      =====PRINT_ST=====
240      kchk/000/Hg/2252/3
241      =====
242
243      =====ERROR=====
244      500
245      =====
246
247      =====PRINT_ST=====
248      book/200/risa/1990/3
249      =====
250
251      =====PRINT_ST=====
252      loku/300/Tc/2190/4
253      seey/300/Yu Aa Mv/2086/4
254      =====
255
256      =====PRINT_ST=====
257      xmwf/400/Ew Bt Fd/1959/4
258      =====
259
260      =====PRINT_ST=====
261      klim/500/lo Sd/2066/2
262      =====
263
264      =====PRINT_ST=====
265      knfu tmlf/600/Vh/2246/2
266      =====
267
268      =====PRINT_ST=====
269      fkse/700/Ws Om Vk/2136/2
270      ydxb mxko/700/Sf/2187/2
271      =====
272
273      =====ERROR=====
274      500
275      ---
33      DELETE
34      PRINT_ST      000
35      PRINT_ST      100
36      PRINT_ST      200
37      PRINT_ST      300
38      PRINT_ST      400
39      PRINT_ST      500
40      PRINT_ST      600
41      PRINT_ST      700
42      PRINT_ST      800

```

도서 분류 코드	대출 불가 도서
000	abc / kchk
100	
200	book
300	loku / seey

400	xmwf
500	klim
600	knfu tmlf
700	ybx b mxko / fkse

DELETE가 성공함에 따라, abc라는 이름을 가진 root의 데이터가 삭제되었습니다. 또한 Wrong code 입력에 대해서도 에러문을 출력하는 것을 확인할 수 있습니다.

```

43  DELETE
44  PRINT_ST 200
45  Wrong Command
46  EXIT

276
277  ===== DELETE =====
278  Success
279  =====
280
281  =====ERROR=====
282  500
283  =====
284
285  =====ERROR=====
286  700
287  =====
288
289  ===== EXIT =====
290  Success
291  =====
292

```

다음 root인 book을 지우고, book이 속한 code 200을 출력해보면, 삭제되어 데이터가 삭제된 것을 확인할 수 있습니다. 마지막으로 Wrong Command에 대한 Error 문도 출력을 하며, EXIT 커맨드에 따라 종료했습니다.

- Consideration

이번 과제는 상당히 복잡했습니다. 왜냐하면, 서로 독립되어 데이터만 주고받는 독립된 세 가지의 자료구조가 아닌, Heap은 Selection Tree의 RUN으로써 구성이 되어 있기 때문입니다. 그래서 맨 처음에 자료구조 간의 관계성을 설정하는데 상당히 복잡하고 헷갈렸습니다. 하지만 실습 자료에서 이미 코드의 개수는 정해져 있기 때문에, 미리 Selection Tree를 만들면 좋을 것 같다고 적혀있어서, 그 이후에는 좀 편하게 연결을 했습니다.

또한 저는 Heap을 구현하는 과정에서, Insert시 마지막으로 삽입되었던 위치의 다음 위치를 잡는데 있어서 많이 헤맸습니다. 그래서 고민 끝에, Heap이나 SelectionTree를 주어진 스켈레톤 코드에 맞게 Linked List로 구현은 하나, 구현 시 각각의 노드를 접근하는 데 있어서 편의성을 위해, 벡터를 사용하여 각각의 노드의 주소만을 저장하였습니다. 그렇게 했더니 Linked List로 구현도 가능하면서, Node를 접근할 시에는 상당히 시간 복잡도가 낮아졌습니다.

마지막으로 이번 B+Tree 구현 시 Delete를 완벽하게 구현하지 못했습니다. 왜냐하면 Delete 시 노드끼리 Merge 하는 데 있어서 상당한 복잡함과 어려움을 겪어 시간이 부족했습니다. 하지만 출력 예시와는 반드시 동일하게 나와야 한다는 생각을 하고 있어, 직접 Delete는 못했지만, Delete를 해야 하는 BpTreeNode로 접근하여 해당 Node의 Count를 -1로 초기화하였습니다. 이렇게 되면, 나중에 동명의 데이터가 다시 ADD 된다면 update count만 하면 되어서 다시 0으로 초기화도

될 것이며, -1이라는 카운트를 기준으로 Print를 할지 말지를 결정할 수 있었기에, 출력 예시와 매우 흡사하게 출력이 될 수 있었습니다.

이렇게 지금까지 저의 데이터구조설계 2차 프로젝트에 관한 보고서였습니다.

감사합니다.