

PREPARE^{NEW}

CERTIFY

COMPETE

Search



cs21b084 ▾

All Contests > OOAIA-2023-Lab-4 > Databases

Databases

locked

Problem

Submissions

Leaderboard

Discussions

In this challenge, the objective is to implement a `Database` class which is templated on the table schema. A `Database` class will store a collection of **records**, where each record will consist of a **key** and a **value**, which themselves are collection of fields (also called the **schema**). For example, in a student database, the key could be the student roll number (of type `string`), while the value could consist of fields like name, DOB, stream, semester, etc. The key may also consist of multiple fields, for example, in a database storing details about BTP/DDP guides of students, the key would be student roll number and faculty name. The point is that for every key instance, there would be at most one record with that key in the database.

The `Database` class will be templated on classes `Key` and `Value`. You need to create a `Record` class which stores all details of a record (i.e. the `Key` and the `Value`). The `Database` class should then store all the records using the `DoublyLinkedList` class created in challenge-1. You should then implement the following methods in the `Database` class:

- `isPresent(Key & k)` returns `true` if the input key is present in the database, otherwise `false`. `selectRecord(Key & k)` returns the record containing the key (you can assume that `k` will be present).
- `selectRangeRecord(Key & rangeStart, Key & rangeEnd)` returns a vector containing all the records in the database whose key lies between `rangeStart` and `rangeEnd` (inclusive).
- `updateRecord(Key & k, Value & v)` updates the record in the database whose key is `k` with value `v`. If `k` is not present, there is no change in the database.
- `updateRangeRecord(Key & rangeStart, Key & rangeEnd, Value & v)` updates all records in the database whose key lies between `rangeStart` and `rangeEnd`, with value `v` (inclusive).
- `insertRecord(Key & k, Value & v)` inserts a new record with `k` and `v`.
- `deleteRecord(Key & k)` deletes the record with key `k`. `deleteRangeRecord(Key & rangeStart, Key & rangeEnd)` deletes all records with key between `rangeStart` and `rangeEnd` (inclusive).
- `getMinRecord()` and `getMaxRecord()` return the minimum and maximum record respectively (you can assume that the database won't be empty).
- `getAllRecords()` returns a vector containing all the records in the database.

To implement these operations efficiently, you should ensure that while adding/deleting records from the linked list, it remains sorted in increasing order according to `Key`. Now, while searching for a record with a specific `Key k`, you should implement a binary search-like procedure which should make at most *logn* calls to `atIndex` method of `DoublyLinkedList`, where *n* is the number of records in the database. To summarize, use the fact that the linked list is sorted to implement the methods as efficiently as possible. Note that you should not change the `DoublyLinkedList` implementation from challenge-1. Your implementation of `Database` class should call appropriate methods of the `DoublyLinkedList` class. You need not worry about the time complexity of the methods in the `DoublyLinkedList` implementation.

There will be a 30% penalty for inefficient implementations.

Input Format

Input is handled by the starter code. Every input is terminated by a command "50". Which is terminating command.

Constraints

None.

Output Format

Output is handled by the starter code.

Sample Input 0

```
1
2
STUDENT AE19S078 Pearce Jamuna 7.57
COURSE CS6122 PROBABILISTIC-AND-SMOOTHED-ALGORITHM EVEN BVVR
2
STUDENT AE19S078
2
FACULTY CS1
50
```

Sample Output 0

```
1
0
```

Explanation 0

Student with roll number AE19S078 is present in the student database and there is no entry in faculty database.

Sample Input 1

```
1
2
STUDENT AE19S078 Pearce Jamuna 7.57
COURSE CS6122 PROBABILISTIC-AND-SMOOTHED-ALGORITHM EVEN BVVR
3
COURSE CS6122
3
STUDENT AE19S078
50
```

Sample Output 1

```
CS6122 PROBABILISTIC-AND-SMOOTHED-ALGORITHM EVEN BVVR
AE19S078 Pearce Jamuna 7.57
```

Sample Input 2

```
1
5
COURSE CS6508 Natural_Language_Processing EVEN BVVR
COURSE CS6044 Speech_Technology ODD RAVI
COURSE CS6422 Advanced_Wireless_Communications_and_Networks EVEN AYON
COURSE CS6480 Reinforcement_Learning EVEN YADU
COURSE CS6196 Stochastic_Optimization ODD KKR
4
COURSE CS6196 CS6480
50
```

Sample Output 2

```

CS6196 Stochastic_Optimization ODD KKR
CS6422 Advanced_Wireless_Communications_and_Networks EVEN AYON
CS6480 Reinforcement_Learning EVEN YADU

```

Sample Input 3

```

1
5
COURSE CS6508 Natural_Language_Processing EVEN BVVR
COURSE CS6044 Speech_Technology ODD RAVI
COURSE CS6422 Advanced_Wireless_Communications_and_Networks EVEN AYON
COURSE CS6480 Reinforcement_Learning EVEN YADU
COURSE CS6196 Stochastic_Optimization ODD KKR
5
COURSE CS6196 Multi-armed_bandits EVEN BVVR
5
COURSE CS6521 Speech_Technology ODD RAVI
3
COURSE CS6196
2
COURSE CS6521
50

```

Sample Output 3

```

CS6196 Multi-armed_bandits EVEN BVVR
0

```

Sample Input 4

```

1
5
FACULTY CS15 Elvin Asst.Professor 10000
FACULTY CS10 Gaston Assoc.Professor 20000
FACULTY CS01 Earlie Professor 30000
FACULTY CS24 Bobbe Assoc.Professor 20000
FACULTY CS09 Darsey Professor 30000
6
FACULTY CS10 CS15 Glynda Professor 30000
4
FACULTY CS10 CS15
50

```

Sample Output 4

```

CS10 Glynda Professor 30000
CS15 Glynda Professor 30000

```

Sample Input 5

```

1
5
STUDENT PH23M044 Ives Alakananda 9.58
STUDENT PH23M099 Quillan Mandakini 8.17
STUDENT PH23S025 Ailee Mahanadi 6.5
STUDENT PH23S056 Luella Mahanadi 7.3
STUDENT PH23S057 Beatrisa Mandakini 8.93
5
STUDENT PH23S025 Beatrisa Mandakini 8.93
3
STUDENT PH23S025
50

```

Sample Output 5

PH23S025 Beatrisa Mandakini 8.93

Sample Input 6

```
1
5
FACULTY CS15 Elvin Asst.Professor 10000
FACULTY CS10 Gaston Assoc.Professor 20000
FACULTY CS01 Earlie Professor 30000
FACULTY CS24 Bobbe Assoc.Professor 20000
FACULTY CS09 Darsey Professor 30000
4
FACULTY CS10 CS24
7
FACULTY CS15
4
FACULTY CS10 CS24
50
```

Sample Output 6

```
CS10 Gaston Assoc.Professor 20000
CS15 Elvin Asst.Professor 10000
CS24 Bobbe Assoc.Professor 20000
CS10 Gaston Assoc.Professor 20000
CS24 Bobbe Assoc.Professor 20000
```

Sample Input 7

```
1
5
FACULTY CS15 Elvin Asst.Professor 10000
FACULTY CS10 Gaston Assoc.Professor 20000
FACULTY CS01 Earlie Professor 30000
FACULTY CS24 Bobbe Assoc.Professor 20000
FACULTY CS09 Darsey Professor 30000
4
FACULTY CS09 CS24
8
FACULTY CS10 CS15
2
FACULTY CS15
4
FACULTY CS09 CS24
50
```

Sample Output 7

```
CS09 Darsey Professor 30000
CS10 Gaston Assoc.Professor 20000
CS15 Elvin Asst.Professor 10000
CS24 Bobbe Assoc.Professor 20000
0
CS09 Darsey Professor 30000
CS24 Bobbe Assoc.Professor 20000
```

Sample Input 8

```
1
3
STUDENT AE23D016 Bobbe Mandakini 7.82
STUDENT ME19B05 Earlie Mandakini 6.93
```

```
STUDENT AE19S078 Pearce Jamuna 7.57
9
STUDENT
50
```

Sample Output 8

```
AE19S078 Pearce Jamuna 7.57
```

Sample Input 9

```
1
3
STUDENT AE23D016 Bobbe Mandakini 7.82
STUDENT ME19B05 Earlie Mandakini 6.93
STUDENT AE19S078 Pearce Jamuna 7.57
10
STUDENT
50
```

Sample Output 9

```
ME19B05 Earlie Mandakini 6.93
```

[f](#) [t](#) [in](#)

Submissions: 85

Max Score: 60

Difficulty: Medium

Rate This Challenge:

☆☆☆☆☆

[More](#)

C++20



```
1 #include <iostream>
2 #include <string>
3 #include <vector>
4
5 using namespace std;
6 template<class object>
7 int Bsearch(object* arr, int n, int m, object k)
8 {
9
10     while(n!=m)
11     {
12         int mid = (n+m)/2;
13         if (arr[mid]<k)
14         {
15             n = mid+1;
16         }
17         else if (arr[mid]>k)
18         {
19             m = mid-1;
20         }
21         else{
22             return mid;
23         }
24     }
25     if (arr[n]>k)
```

```

26     {
27         return n;
28     }
29     else{
30         return n+1;
31     }
32     return n;
33 }
34 template<class Object>
35 class DoublyLinkedList
36 {
37 private:
38     int lastindex;
39     struct Node
40     {
41     Object data;
42     Node *next;
43     Node *prev;
44     Node(const Object & d = Object(), Node * p = nullptr, Node * n = nullptr)
45         : data(d), prev(p), next(n) {}
46     };
47
48     Node *head; //sentinel node at the beginning
49     Node *rear; //sentinel node at the end
50
51
52 public:
53
54     /*TODO: Define a constructor for DoublyLinkedList here, allocating the sentinel nodes*/
55     DoublyLinkedList (){
56         head = new Node; rear = new Node;
57         head->prev = NULL;
58         rear->next = NULL;
59         head->next = rear;
60         rear->prev = head;
61         lastindex = -1;
62     }
63
64     class Iterator
65     {
66     private:
67         Node *current;
68     public:
69         Iterator() { }
70         Iterator(Node *inp) : current(inp) {}
71         Object & operator*() {return current->data;}
72         Iterator & operator ++()
73         {
74             this->current = this->current->next;
75             return *this;
76         }
77         Iterator & operator --()
78         {
79             this->current = this->current->prev;
80             return *this;
81         }
82         bool operator != (Iterator rhs)
83         {
84             if (this->current==rhs.current){return false;}
85             else {return true;}
86         }
87
88
89     /*TODO: You can add more methods here */
90
91     friend class DoublyLinkedList<Object>;

```

```

92 //friend class Database;
93 };
94 int lstid(){return lastindex;}
95 Iterator begin(){return Iterator(head->next);}
96 Iterator end(){return Iterator(rear);}
97 void insert(Iterator itr, Object o)
98 {
99     lastindex++;
100     Node* nn = new Node;
101     nn->data = o;
102     (itr.current)->prev->next = nn;
103     nn->prev = (itr.current)->prev;
104     (itr.current)->prev = nn;
105     nn->next = itr.current;
106 }
107 void erase(Iterator itr)
108 {
109     lastindex--;
110     (itr.current)->next->prev = itr.current->prev;
111     (itr.current->prev)->next = itr.current->next;
112 }
113 Iterator atIndex(int p)
114 { /* Implement here */
115     //cout<<p<<endl;
116     Iterator ff;
117     ff = Iterator(head->next);
118     for (int ii=0; ii<p; ii++)
119     {
120         ++ff;
121     }
122     return ff;
123 }
124 int indexOf(Iterator itr)
125 { /* Implement here */
126     auto ptr = (begin()).current;
127     for (int ii=0; ii<=lastindex; ii++)
128     {
129         if (ptr->data==*(itr))
130         {
131             return ii;
132         }
133         ptr=ptr->next;
134     }
135     return -1;
136 }
137 void display()
138 {
139     for (auto it = begin(); it != end(); ++it)
140         cout << *it << " ";
141     cout << endl;
142 }
143 };
144 //template<class Key, class Value>
145 template <class Key, class Value>
146 class Database{
147 public:
148     class Record {
149     private:
150         Key k;
151         Value v;
152     public:
153         Record(const Key &k = Key(), const Value &iv = Value()) : k(k), v(iv) {}
154         Key &getKey() { return k; }
155         Value &getValue() { return v; }
156         const bool & operator == (const Record & rhs)

```

```

158     {
159         Key t = rhs.k;
160         return !(this->k < t || t < this->k);
161     }
162     friend class Database<Key, Value>;
163 };
164
165 private:
166     DoublyLinkedList<Record> list;
167     /*use the DoublyLinkedList class created in challenge 1*/
168 public:
169     Database() {DoublyLinkedList<Record> list;}
170     /*TODO: Implement the methods here*/
171     void insertRecord(Key k, Value v)
172     {
173         int n = 0, m = list.lstid();
174         if (n>m)
175         {
176             list.insert(list.atIndex(0), Record(k, v));
177             return;
178         }
179         while(n<m)
180         {
181             int mid = (n+m)/2;
182             if ((*list.atIndex(mid)).k<k)
183             {
184                 n = mid+1;
185             }
186             else if (k<(*list.atIndex(mid)).k)
187             {
188                 m = mid-1;
189             }
190             else{
191                 return;
192             }
193         }
194         if (k<(*list.atIndex(n)).k)
195         {
196             list.insert(list.atIndex(n), Record(k, v));
197             return;
198         }
199         else{
200             list.insert(list.atIndex(n+1), Record(k, v));
201             return;
202         }
203     }
204     bool isPresent(Key k)
205     {
206         int hh=-1;
207         int n = 0, m = list.lstid();
208         while(n<m)
209         {
210             int mid = (n+m)/2;
211             if ((*list.atIndex(mid)).k<k)
212             {
213                 n = mid+1;
214             }
215             else if (k<(*list.atIndex(mid)).k)
216             {
217                 m = mid-1;
218             }
219             else{
220                 hh = mid;
221                 return true;
222             }
223         }

```



```

224     if (!((*(list.atIndex(n))).k<k)|| (k<*(list.atIndex(n)).k))) { hh = n; return true;}
225     return false;
226 }
227 Record selectRecord(Key k)
228 {
229     int n = 0, m = list.lstid();
230     while(n<m)
231     {
232         int mid = (n+m)/2;
233         if ((*(list.atIndex(mid))).k<k)
234         {
235             n = mid+1;
236         }
237         else if (k<*(list.atIndex(mid)).k)
238         {
239             m = mid-1;
240         }
241         else{
242             return *(list.atIndex(mid));
243         }
244     }
245     return *(list.atIndex(n));
246 }
247 vector<Record> selectRangeRecord(Key rs, Key re)
248 {
249
250     vector<Record> vec;
251     int hh=0, ll=0;
252     int n = 0, m = list.lstid();
253     while(n<m)
254     {
255         int mid = (n+m)/2;
256         if ((*(list.atIndex(mid))).k<rs)
257         {
258             n = mid+1;
259         }
260         else if (rs<*(list.atIndex(mid)).k)
261         {
262             m = mid-1;
263         }
264         else{
265             hh = mid;
266             break;
267         }
268     }
269     if (n==m) {hh = n;}
270     n = 0; m = list.lstid();
271     while(n<m)
272     {
273         int mid = (n+m)/2;
274         if ((*(list.atIndex(mid))).k<re)
275         {
276             n = mid+1;
277         }
278         else if (re<*(list.atIndex(mid)).k)
279         {
280             m = mid-1;
281         }
282         else{
283             ll = mid;
284             break;
285         }
286     }
287     if (n==m) {ll = n;}
288
289     auto yy = list.atIndex(hh);

```

```

290     for (int ii=hh; ii<=ll; ii++)
291     {
292         vec.push_back(*yy);
293         ++yy;
294     }
295     return vec;
296 }
297 void updateRecord(Key k, Value val)
298 {
299     int hh=0;
300     int n = 0, m = list.lstid();
301     while(n<m)
302     {
303         int mid = (n+m)/2;
304         if ((*list.atIndex(mid)).k<k)
305         {
306             n = mid+1;
307         }
308         else if (k<(*list.atIndex(mid)).k)
309         {
310             m = mid-1;
311         }
312         else{
313             (*list.atIndex(mid)).v = val;
314             break;
315         }
316     }
317     if (!(k<(*list.atIndex(n)).k)||((*list.atIndex(n)).k)<k) {(*list.atIndex(n)).v
= val;}
318     else
319     {
320         return;
321     }
322 }
323
324 }
325 void updateRangeRecord(Key rs, Key re, Value val)
326 {
327     int hh=0, ll=0;
328     int n = 0, m = list.lstid();
329     while(n<m)
330     {
331         int mid = (n+m)/2;
332         if ((*list.atIndex(mid)).k<rs)
333         {
334             n = mid+1;
335         }
336         else if (rs<(*list.atIndex(mid)).k)
337         {
338             m = mid-1;
339         }
340         else{
341             hh = mid;
342             break;
343         }
344     }
345     if (n==m) {hh = n;}
346     n = 0; m = list.lstid();
347     while(n<m)
348     {
349         int mid = (n+m)/2;
350         if ((*list.atIndex(mid)).k<re)
351         {
352             n = mid+1;
353         }
354         else if (re<(*list.atIndex(mid)).k)

```

```

355     {
356         m = mid-1;
357     }
358     else{
359         ll = mid;
360         break;
361     }
362 }
363 if (n==m) {ll = n;}
364 auto yy = list.atIndex(hh);
365 for (int ii=hh; ii<=ll; ii++)
366 {
367     (*yy).v = val;
368     ++yy;
369 }
370 }
371 void deleteRecord(Key k)
372 {
373     int hh=0;
374     int n = 0, m = list.lstid();
375     while(n<m)
376     {
377         int mid = (n+m)/2;
378         if ((*list.atIndex(mid))).k<k)
379         {
380             n = mid+1;
381         }
382         else if (k<(*list.atIndex(mid))).k)
383         {
384             m = mid-1;
385         }
386         else{
387             hh = mid;
388             break;
389         }
390     }
391     if (!(k<(*list.atIndex(n)).k)||((*list.atIndex(n)).k<k)) {hh = n;}
392     list.erase(list.atIndex(hh));
393 }
394 void deleteRangeRecord(Key rs, Key re)
395 {
396     int hh=0, ll=0;
397     int n = 0, m = list.lstid();
398     while(n<m)
399     {
400         int mid = (n+m)/2;
401         if ((*list.atIndex(mid))).k<rs)
402         {
403             n = mid+1;
404         }
405         else if (rs<(*list.atIndex(mid)).k)
406         {
407             m = mid-1;
408         }
409         else{
410             hh = mid;
411             break;
412         }
413     }
414     if (n==m) {hh = n;}
415     n = 0; m = list.lstid();
416     while(n<m)
417     {
418         int mid = (n+m)/2;
419         if ((*list.atIndex(mid))).k<re)
420         {

```

```

421         n = mid+1;
422     }
423     else if (re<*(list.atIndex(mid)).k)
424     {
425         m = mid-1;
426     }
427     else{
428         ll = mid;
429         break;
430     }
431 }
432 if (n==m) {ll = n;}
433 auto yy = list.atIndex(hh);
434 for (int ii=hh; ii<=ll; ii++)
435 {
436     list.erase(yy);
437     --yy;
438     ++yy;
439 }
440 }
441 Record getMinRecord()
442 {
443     return *(list.atIndex(0));
444 }
445 Record getMaxRecord()
446 {
447     return *(list.atIndex(list.lstid()));
448 }
449 vector<Record> getAllRecords()
450 {
451     vector<Record> v;
452     auto yy = (list.atIndex(0));
453     for (int ii=0; ii<=list.lstid(); ii++)
454     {
455         v.push_back(*yy);
456         ++yy;
457     }
458     return v;
459 }
460 };
461
462 class StudentsKey {
463 public:
464     string rollNo;
465     StudentsKey(const string &inp = "") : rollNo(inp) {}
466     bool operator<(StudentsKey &rhs) { return rollNo < rhs.rollNo; }
467     friend ostream & operator<<(ostream & out, StudentsKey &k);
468 };
469 ostream & operator<<(ostream & out, StudentsKey &k) {
470     out << k.rollNo;
471     return out;
472 }
473
474 class StudentsValue {
475 public:
476     string name;
477     string hostel;
478     float cgpa;
479     StudentsValue(const string &n = "", const string &h = "", float c = 10.0) : name(n),
480     hostel(h), cgpa(c) {}
481     friend ostream & operator<<(ostream & out, StudentsValue &v);
482 };
483 ostream & operator<<(std::ostream & out, StudentsValue &v) {
484     out << v.name << " " << v.hostel << " " << v.cgpa;
485     return out;

```

```

486 }
487
488 class FacultyKey {
489 public:
490     string empId;
491     FacultyKey(const string &inp = "") : empId(inp) {}
492     bool operator<(FacultyKey &rhs) { return empId < rhs.empId; }
493     friend ostream & operator<<(ostream & out, FacultyKey &k);
494 };
495
496 ostream & operator<<(ostream & out, FacultyKey &k) {
497     out << k.empId;
498     return out;
499 }
500
501 class FacultyValue {
502 public:
503     string name;
504     string designation;
505     float salary;
506     FacultyValue(const string &n = "", const string &d = "", float s = 10.0) : name(n),
507     designation(d), salary(s) {}
508     friend ostream & operator<<(ostream & out, FacultyValue &v);
509 };
510 ostream & operator<<(std::ostream & out, FacultyValue &v) {
511     out << v.name << " " << v.designation << " " << v.salary;
512     return out;
513 }
514
515 class CourseKey {
516 public:
517     string courseId;
518     CourseKey(const string &inp = "") : courseId(inp) {}
519     bool operator<(CourseKey &rhs) { return courseId < rhs.courseId; }
520     friend ostream & operator<<(ostream & out, CourseKey &k);
521 };
522
523 ostream & operator<<(ostream & out, CourseKey &k) {
524     out << k.courseId;
525     return out;
526 }
527
528 class CourseValue {
529 public:
530     string name;
531     string semester;
532     string facultyName;
533     CourseValue(const string &n = "", const string &s = "", const string &f = "") : name(n),
534     semester(s), facultyName(f) {}
535     friend ostream & operator<<(ostream & out, FacultyValue &v);
536 };
537 ostream & operator<<(std::ostream & out, CourseValue &v) {
538     out << v.name << " " << v.semester << " " << v.facultyName;
539     return out;
540 }
541
542 int main() {
543     int command;
544     bool b;
545     int BREAKING_COMMAND = 50;
546
547     Database<StudentsKey, StudentsValue> student_db;
548     Database<FacultyKey, FacultyValue> faculty_db;
549     Database<CourseKey, CourseValue> course_db;

```

```

550
551 while (true) {
552     cin >> command;
553     if (command == BREAKING_COMMAND) {
554         break;
555     }
556
557     if (command == 1) { /*insert record*/
558         int numberOfRecords = 0; /*number of records to be inserted*/
559         cin >> numberOfRecords;
560
561         while (numberOfRecords--> 0) {
562
563             string database;
564             cin >> database;
565
566             if (database == "STUDENT") {
567                 string rollNo, name, hostel;
568                 float cgpa;
569                 cin >> rollNo >> name >> hostel >> cgpa;
570
571                 StudentsKey k(rollNo);
572                 StudentsValue v(name, hostel, cgpa);
573                 student_db.insertRecord(k, v);
574             } else if (database == "FACULTY") {
575                 string empId, name, designation;
576                 float salary;
577                 cin >> empId >> name >> designation >> salary;
578
579                 FacultyKey k(empId);
580                 FacultyValue v(name, designation, salary);
581                 faculty_db.insertRecord(k, v);
582             } else {
583                 string courseId, name, semester, facultyName;
584                 cin >> courseId >> name >> semester >> facultyName;
585
586                 CourseKey k(courseId);
587                 CourseValue v(name, semester, facultyName);
588                 course_db.insertRecord(k, v);
589             }
590         }
591
592     } else if (command == 2) { /*check whether the key is present*/
593         string database;
594         cin >> database;
595
596         if (database == "STUDENT") {
597             string rollNo;
598             cin >> rollNo;
599
600             StudentsKey k(rollNo);
601             b = student_db.isPresent(k);
602             cout << b << endl;
603         } else if (database == "FACULTY") {
604             string empId;
605             cin >> empId;
606
607             FacultyKey k(empId);
608             b = faculty_db.isPresent(k);
609             cout << b << endl;
610         } else {
611             string courseId;
612             cin >> courseId;
613
614             CourseKey k(courseId);
615             b = course_db.isPresent(k);

```

```

616         cout << b << endl;
617     }
618
619     } else if (command == 3) {
620         string database; /*returns the record containing the key*/
621         cin >> database;
622
623         if (database == "STUDENT") {
624             string rollNo;
625             cin >> rollNo;
626
627             StudentsKey k(rollNo);
628             auto r = student_db.selectRecord(k);
629             cout << r.getKey() << " " << r.getValue() << endl;
630         } else if (database == "FACULTY") {
631             string empId;
632             cin >> empId;
633
634             FacultyKey k(empId);
635             auto r = faculty_db.selectRecord(k);
636             cout << r.getKey() << " " << r.getValue() << endl;
637         } else {
638             string courseId;
639             cin >> courseId;
640
641             CourseKey k(courseId);
642             auto r = course_db.selectRecord(k);
643             cout << r.getKey() << " " << r.getValue() << endl;
644         }
645
646     } else if (command == 4) { /*returns a vector containing all the records in the
database
647                               whose key lies between rangeStart and rangeEnd
(inclusive).*/
648         string database;
649         cin >> database;
650
651         if (database == "STUDENT") {
652             string rangeStart, rangeEnd;
653             cin >> rangeStart >> rangeEnd;
654
655             StudentsKey rs(rangeStart);
656             StudentsKey re(rangeEnd);
657             auto lst = student_db.selectRangeRecord(rs, re);
658             for (auto r : lst) {
659                 cout << r.getKey() << " " << r.getValue() << endl;
660             }
661         } else if (database == "FACULTY") {
662             string rangeStart, rangeEnd;
663             cin >> rangeStart >> rangeEnd;
664
665             FacultyKey rs(rangeStart);
666             FacultyKey re(rangeEnd);
667             auto lst = faculty_db.selectRangeRecord(rs, re);
668             for (auto r : lst) {
669                 cout << r.getKey() << " " << r.getValue() << endl;
670             }
671         } else {
672             string rangeStart, rangeEnd;
673             cin >> rangeStart >> rangeEnd;
674
675             CourseKey rs(rangeStart);
676             CourseKey re(rangeEnd);
677             auto lst = course_db.selectRangeRecord(rs, re);
678             for (auto r : lst) {
679                 cout << r.getKey() << " " << r.getValue() << endl;

```

```

680     }
681 }
682
683 } else if (command == 5) { /*updates the record in the database whose key is k with
value v.
684                                     If k is not present, there is no change in the database.*/
685     string database;
686     cin >> database;
687
688     if (database == "STUDENT") {
689         string rollNo, name, hostel;
690         float cgpa;
691         cin >> rollNo >> name >> hostel >> cgpa;
692
693         StudentsKey k(rollNo);
694         StudentsValue v(name, hostel, cgpa);
695         student_db.updateRecord(k, v);
696     } else if (database == "FACULTY") {
697         string empId, name, designation;
698         float salary;
699         cin >> empId >> name >> designation >> salary;
700
701         FacultyKey k(empId);
702         FacultyValue v(name, designation, salary);
703         faculty_db.updateRecord(k, v);
704     } else {
705         string courseId, name, semester, facultyName;
706         cin >> courseId >> name >> semester >> facultyName;
707
708         CourseKey k(courseId);
709         CourseValue v(name, semester, facultyName);
710         course_db.updateRecord(k, v);
711     }
712
713 } else if (command == 6) { /*updates all records in the database whose key lies between
714                             rangeStart and rangeEnd, with value v (inclusive).*/
715     string database;
716     cin >> database;
717
718     if (database == "STUDENT") {
719         string rangeStart, rangeEnd, name, hostel;
720         float cgpa;
721         cin >> rangeStart >> rangeEnd >> name >> hostel >> cgpa;
722
723         StudentsKey rs(rangeStart);
724         StudentsKey re(rangeEnd);
725         StudentsValue v(name, hostel, cgpa);
726         student_db.updateRangeRecord(rs, re, v);
727     } else if (database == "FACULTY") {
728         string rangeStart, rangeEnd, name, designation;
729         float salary;
730         cin >> rangeStart >> rangeEnd >> name >> designation >> salary;
731
732         FacultyKey rs(rangeStart);
733         FacultyKey re(rangeEnd);
734         FacultyValue v(name, designation, salary);
735         faculty_db.updateRangeRecord(rs, re, v);
736     } else {
737         string rangeStart, rangeEnd, name, semester, facultyName;
738         cin >> rangeStart >> rangeEnd >> name >> semester >> facultyName;
739
740         CourseKey rs(rangeStart);
741         CourseKey re(rangeEnd);
742         CourseValue v(name, semester, facultyName);
743         course_db.updateRangeRecord(rs, re, v);
744     }

```



```
745
746     } else if (command == 7) { /*deletes the record with key k.*/
747         string database;
748         cin >> database;
749
750         if (database == "STUDENT") {
751             string rollNo;
752             cin >> rollNo;
753
754             StudentsKey k(rollNo);
755             student_db.deleteRecord(k);
756         } else if (database == "FACULTY") {
757             string empId;
758             cin >> empId;
759
760             FacultyKey k(empId);
761             faculty_db.deleteRecord(k);
762         } else {
763             string courseId;
764             cin >> courseId;
765
766             CourseKey k(courseId);
767             course_db.deleteRecord(k);
768         }
769
770     } else if (command == 8) { /*deletes all records with key between rangeStart and
rangeEnd (inclusive).*/
771         string database;
772         cin >> database;
773
774         if (database == "STUDENT") {
775             string rangeStart, rangeEnd;
776             cin >> rangeStart >> rangeEnd;
777
778             StudentsKey rs(rangeStart);
779             StudentsKey re(rangeEnd);
780             student_db.deleteRangeRecord(rs, re);
781         } else if (database == "FACULTY") {
782             string rangeStart, rangeEnd;
783             cin >> rangeStart >> rangeEnd;
784
785             FacultyKey rs(rangeStart);
786             FacultyKey re(rangeEnd);
787             faculty_db.deleteRangeRecord(rs, re);
788         } else {
789             string rangeStart, rangeEnd;
790             cin >> rangeStart >> rangeEnd;
791
792             CourseKey rs(rangeStart);
793             CourseKey re(rangeEnd);
794             course_db.deleteRangeRecord(rs, re);
795         }
796
797     } else if (command == 9) { /*return the minimum record*/
798         string database;
799         cin >> database;
800
801         if (database == "STUDENT") {
802             auto r = student_db.getMinRecord();
803             cout << r.getKey() << " " << r.getValue() << endl;
804         } else if (database == "FACULTY") {
805             auto r = faculty_db.getMinRecord();
806             cout << r.getKey() << " " << r.getValue() << endl;
807         } else {
808             auto r = course_db.getMinRecord();
809             cout << r.getKey() << " " << r.getValue() << endl;
```

```
810     }
811
812 } else if (command == 10) { /*return the maximum record*/
813     string database;
814     cin >> database;
815
816     if (database == "STUDENT") {
817         auto r = student_db.getMaxRecord();
818         cout << r.getKey() << " " << r.getValue() << endl;
819     } else if (database == "FACULTY") {
820         auto r = faculty_db.getMaxRecord();
821         cout << r.getKey() << " " << r.getValue() << endl;
822     } else {
823         auto r = course_db.getMaxRecord();
824         cout << r.getKey() << " " << r.getValue() << endl;
825     }
826
827 } else if (command == 11) { /*returns a vector containing all the records in the
database.*/
828     string database;
829     cin >> database;
830
831     if (database == "STUDENT") {
832         auto lst = student_db.getAllRecords();
833         for (auto r : lst) {
834             cout << r.getKey() << " " << r.getValue() << endl;
835         }
836     } else if (database == "FACULTY") {
837         auto lst = faculty_db.getAllRecords();
838         for (auto r : lst) {
839             cout << r.getKey() << " " << r.getValue() << endl;
840         }
841     } else {
842         auto lst = course_db.getAllRecords();
843         for (auto r : lst) {
844             cout << r.getKey() << " " << r.getValue() << endl;
845         }
846     }
847
848 } else {
849     cout << "INVALID COMMAND!" << endl;
850     break;
851 }
852 }
853 }
```

Line: 1 Col: 1

 Upload Code as File ☐ Test against custom input

Run Code

Submit Code