Wet HW 2 (wet part)

Graded

Group

גיא פרידמן דניאלה כהן

View or edit group

Total Points

100 / 100 pts

Autograder Score 100.0 / 100.0

Passed Tests

STL

Compilation

Memory Leaks

- 0) Test 0 (2/2)
- 1) Test 1 (2/2)
- 2) Test 2 (2/2)
- 3) Test 3 (2/2)
- 4) Test 4 (2/2)
- 5) Test 5 (2/2)
- 6) Test 6 (2/2)
- 7) Test 7 (2/2)
- 8) Test 8 (2/2)
- 9) Test 9 (2/2)
- 10) Test 10 (2/2)
- 11) Test 11 (2/2)
- 12) Test 12 (2/2)
- 13) Test 13 (2/2)
- 14) Test 14 (2/2)
- 15) Test 15 (2/2)
- 16) Test 16 (2/2)
- 17) Test 17 (2/2) 18) Test 18 (2/2)
- 19) Test 19 (2/2)
- 13) 1636 13 (2/2)
- 20) Test 20 (2/2)
- 21) Test 21 (2/2)
- 22) Test 22 (2/2)
- 23) Test 23 (2/2)
- 24) Test 24 (2/2)
- 25) Test 25 (2/2)
- 26) Test 26 (2/2)
- 27) Test 27 (2/2)
- 28) Test 28 (2/2)
- 29) Test 29 (2/2)
- 30) Test 30 (2/2)
- 31) Test 31 (2/2)
- 32) Test 32 (2/2)
- 33) Test 33 (2/2)
- 34) Test 34 (2/2)
- 35) Test 35 (2/2)
- 36) Test 36 (2/2)
- 37) Test 37 (2/2)
- 38) Test 38 (2/2)
- 39) Test 39 (2/2)
- 40) Test 40 (2/2)
- 41) Test 41 (2/2)
- 42) Test 42 (2/2)

46) Test 46 (2/2) 47) Test 47 (2/2) 48) Test 48 (2/2) 49) Test 49 (2/2)		
Autograder Results		
Autograder Output		
Please ensure that you add your other group member to this submission. A tutorial can be found here: https://shorturl.at/ttSty Valgrind NO LEAKS Test #0 Passed Test #10 Passed Test #20 Passed Test #30 Passed Test #40 Passed		
STL		
Compilation		
Memory Leaks		
0) Test 0 (2/2)		
1) Test 1 (2/2)		
2) Test 2 (2/2)		
3) Test 3 (2/2)		
4) Test 4 (2/2)		
5) Test 5 (2/2)		
6) Test 6 (2/2)		
7) Test 7 (2/2)		

43) Test 43 (2/2) 44) Test 44 (2/2) 45) Test 45 (2/2)

8) Test 8 (2/2)
9) Test 9 (2/2)
10) Test 10 (2/2)
11) Test 11 (2/2)
12) Test 12 (2/2)
13) Test 13 (2/2)
14) Test 14 (2/2)
15) Test 15 (2/2)
16) Test 16 (2/2)
17) Test 17 (2/2)
18) Test 18 (2/2)
19) Test 19 (2/2)
20) Test 20 (2/2)
21) Test 21 (2/2)
22) Test 22 (2/2)
23) Test 23 (2/2)
24) Test 24 (2/2)
25) Test 25 (2/2)

26) Test 26 (2/2)
27) Test 27 (2/2)
28) Test 28 (2/2)
29) Test 29 (2/2)
30) Test 30 (2/2)
31) Test 31 (2/2)
32) Test 32 (2/2)
33) Test 33 (2/2)
34) Test 34 (2/2)
35) Test 35 (2/2)
36) Test 36 (2/2)
37) Test 37 (2/2)
38) Test 38 (2/2)
39) Test 39 (2/2)
40) Test 40 (2/2)
41) Test 41 (2/2)
42) Test 42 (2/2)
43) Test 43 (2/2)

44) Test 44 (2/2)	
45) Test 45 (2/2)	
46) Test 46 (2/2)	
47) Test 47 (2/2)	
48) Test 48 (2/2)	
49) Test 49 (2/2)	

Submitted Files

```
//
1
     // Created by Guy Friedman on 28/01/2025.
2
3
4
5
     #ifndef NEWTEAM_H
     #define NEWTEAM H
6
7
8
     #define SELECT_ID_BY_MAX_RECORD(record1,record2,id1,id2) (record1>=record2?id1:id2)
9
     #include <cassert>
10
11
12
     class NewTeam {
13
     protected:
14
      int id;
15
       int size;
16
       int record;
17
       NewTeam* root_by_mass;
       NewTeam* root_by_record; //for now, it is not used
18
19
     public:
20
21
       NewTeam() = delete;
22
23
       NewTeam(int id): id(id), size(1), record(0), root_by_mass(this), root_by_record(this) {}
24
       /*
25
26
       NewTeam(int id,int newRec, int newSize): NewTeam(id) { //todo maybe can delete
27
         this->record = newRec;
28
         this->size = newSize;
29
       }
       */
30
31
32
       ~NewTeam() {
33
         this->root by mass = nullptr;
34
         this->root_by_record = nullptr;
35
       }
36
37
       NewTeam* get_root_by_mass() {
38
         if (this->root_by_mass == this) {
39
           return this;
40
         }
41
         NewTeam* actual_root_by_mass = this->root_by_mass->get_root_by_mass();
42
         this->root_by_mass = actual_root_by_mass;
43
         return actual_root_by_mass;
44
       }
45
46
       int get_id() {
47
         return this->id;
48
       }
49
```

```
50
       int get_record() {
51
          //make sure that it's only called on the leader
          assert(this == this->root_by_mass);
52
53
54
         return this->record;
55
       }
56
       bool isTeam(int id) {
57
         //make sure that it's only called on the leader
58
59
          assert(this == this->root_by_mass);
         return this->id == id;
60
       }
61
62
63
       void unite_team(NewTeam* other) { //done - make sure in main that 'this' is team 1 and 'other' is
     team 2
         //make sure main has taken care of its responsibilities for union
64
         this->verify_main_for_union(other);
65
66
67
         //get new parameters after union
         int new_id = SELECT_ID_BY_MAX_RECORD(this->record, other->record, this->id, other->id);
68
     //SELECT_ID_BY_MAX_RECORD(record1,record2,id1,id2) (record1>=record2?id1:id2)
69
          int new size = this->size + other->size;
70
         int new_record = this->get_record() + other->get_record(); //
                                                                          int new_record = this->record +
     other->record;
71
72
         //regular unite
73
          this->unite_helper(other,0);
74
         //update merge details(int new id, int new size, int new record)
75
76
         //I don't care who is actually the root, I'll just update both to save me the hassle of checking
77
          this->update_merge_details(new_id,new_size,new_record);
78
          other->update_merge_details(new_id,new_size,new_record);
79
       }
80
       /**
81
82
        *just unite by size, don't care about id, record and other stuff,
        *unite team takes care of this
83
        */
84
       void unite_helper(NewTeam* other, int make_signature_different_to_prevent_confusion) {
85
          //make sure main has taken care of its responsibilities for union
86
87
         this->verify_main_for_union(other);
88
89
         //retrieve original team sizes
90
         int this_size = this->size;
          int other_size = other->size;
91
92
93
         //unify by moving small (not small 'mamash') tree root to point at big tree root
         if (this_size <= other_size) {</pre>
94
95
            this->root_by_mass = other; //other->root_by_mass;
         } else {
96
97
            other->root_by_mass = this; //this->root_by_mass;
98
         }
```

```
99
100
          //update sizes (even though it's updated in 'unite_team', make sure)
101
          int sum_size = this_size + other_size;
102
          this->size = sum_size;
103
          other->size = sum_size;
104
       }
105
106
107
       void verify_main_for_union(NewTeam* other) {
108
          //should only call to unite by roots - responsibility of the main
          assert(other == other->root_by_mass);
109
          assert(this == this->root_by_mass);
110
111
112
113
       void update_merge_details(int new_id, int new_size, int new_record) {
114
          this->id = new id;
115
          this->record = new_record;
116
         this->size = new size;
117
       }
118
119
       bool check active immediate(int idVerify) {
120
          return ((this->id == idVerify) && (this->check_active_immediate()));
121
       }
122
       bool check active immediate() {
123
          return this->root_by_mass == this || this->root_by_mass->id == this->id;
124
       }
125
126
       void winMatch() {
127
          assert(this->root_by_mass == this);
128
          ++this->record;
129
       }
130
131
       void loseMatch() {
          assert(this->root_by_mass == this);
132
133
          --this->record;
134
       }
135
136
137
138
139
140
141 };
142
143
144
145 #endif //NEWTEAM_H
146
```

```
//
1
     // Created by Guy Friedman on 28/01/2025.
2
3
4
5
     #ifndef NEWTEAMARR_H
     #define NEWTEAMARR H
6
7
     #include "NewTeam.h"
8
     #include "ChainHashArray.h"
9
10
11
     class NewTeamArr : public ChainHashArray<NewTeam> {
12
     public:
13
14
       NewTeamArr() : ChainHashArray<NewTeam>() {}
15
16
       ~NewTeamArr() = default;
17
18
       bool team outdated(int team id) {
19
         //@brief - get team root, compare root id to 'team id'
20
         //legacy function, instead of rewriting the code to remove it,
         //I just utilised existing code and logic.
21
22
         bool team_is_active = this->team_active(team_id);
23
         return !team_is_active;
24
       }
25
       bool team_active(int team_id) {
26
27
         //done - cri8 dis foonktzion
28
29
         //find root of team with 'team_id' key
         NewTeam* root team of team id = this->get root of(team id);
30
31
32
         //if team doesn't exist return false
33
         if (root team of team id == nullptr) {
           return false;
34
35
         }
36
37
         //get the id of the root
38
         int id_of_root = root_team_of_team_id->get_id();
39
40
         return id_of_root == team_id;
41
       }
42
43
       void unite_teams(int team1, int team2) {
44
         //@brief - team1->unite_team(other);
45
46
         //make sure that we got existing teams
47
         assert(this->team_active(team1));
         assert(this->team_active(team2));
48
49
```

```
50
          //get the roots of team1 and team2
51
          NewTeam* team_one = this->get_root_of(team1);
          NewTeam* team_two = this->get_root_of(team2);
52
53
          //make sure that we received valid teams
54
55
          assert(team_one->get_id() == team1);
          assert(team_two->get_id() == team2);
56
57
58
         //execute union by utilising existing code and logic
59
          team_one->unite_team(team_two);
60
       }
61
62
       NewTeam* get root of(int team id) {
63
         //@brief - get_root_by_mass() of team held by key 'team_id'
64
65
          //get team with key 'team id'
          NewTeam* temp_placeholder = this->find(team_id);
66
67
         //maybe there isnt any team in initial 'team_id'
68
         if (temp_placeholder == nullptr) {
69
70
            return nullptr;
71
         }
72
73
         //get root of team 'hana"l'
74
          NewTeam* root_team = temp_placeholder->get_root_by_mass();
75
76
          return root_team;
77
       }
78
79
       NewTeam* get_jockeys_actual_team(int id) {
80
          //@brief - get team with id, get root of team with id, return root
81
82
          //get root of team corresponding with key of 'id'
          NewTeam* temp_placeholder = this->get_root_of(id);
83
84
85
         //make sure no funny business is going on
          assert(temp_placeholder != nullptr);
86
87
          assert(this->team_active(temp_placeholder->get_id()));
88
89
          return temp_placeholder;
90
       }
91
92
       bool check_active_immediate(int team_id) {
93
          NewTeam* temp_placeholder = this->find(team_id);
         if (temp_placeholder == nullptr) {
94
95
            return false;
96
         }
97
          return temp_placeholder->check_active_immediate(team_id);
98
       }
99
100 };
101
```

104 #endif //NEWTEAMARR_H

```
//
1
     // Created by Guy Friedman on 28/01/2025.
2
3
4
5
     #ifndef RECORDARR_H
     #define RECORDARR H
6
7
     #include "Record.h"
     #include "ChainHashArray.h"
8
9
     #include "NewTeam.h" //todo maybe change this to a forward declaration instead of inclusion
10
11
12
     class RecordArr : public ChainHashArray<Record> {
13
       public:
14
15
       RecordArr() : ChainHashArray<Record>() {}
16
17
       ~RecordArr() = default;
18
19
       void add team to record(NewTeam* team, int team id, int record id) {
20
         //@brief - check if record exists, if so then add, if not create, store and then add
21
22
         //make sure input is valid
23
         assert(team_id > 0);
24
         assert(team != nullptr);
25
         //get relevant record
26
27
         Record* record = this->find(record id);
28
29
         //if the record doesn't already exist, create it and store it.
         if (record == nullptr) {
30
           record = new Record(record_id);
31
32
           this->insert(record_id,record);
33
         }
34
35
         //add team to relevant record
         record->insert(team_id, team);
36
37
       }
38
39
       void remove_team_from_record( int team_id, int record_id) {
40
         //@brief - use this->remove(), then check if the record is empty, if it is - then delete it
41
42
         //retrieve relevant record
43
         Record* record = this->find(record_id);
44
         //verify input - make sure nothing funny is happening
45
46
         assert(record != nullptr);
47
         //remove relevant team from relevant record
48
49
         record->remove(team_id);
```

```
50
51
          //if the record is empty now, delete it.
52
          if(record->isEmpty()) {
53
            this->deleteItem(record_id);
54
          }
55
        }
56
        bool can_unite_by_record(int record_id) {
57
          //@brief - find record of (record_id) and record of (-record_id),
58
59
          //@brief - check if they are singletons,
          //@brief - if either of them isn't singleton or doesn't exist - return false
60
          //@brief - if both are singletons - return true
61
62
63
          //make sure input is valid
          assert(record_id > 0);
64
65
          //retrieve relevant records
66
          Record* positive record = this->find(record id);
67
          Record* negative record = this->find(-record id);
68
69
70
          //if either of them doesn't exist, can not unite by record
71
          if (positive_record == nullptr | | negative_record == nullptr) {
72
            return false;
73
          }
74
          //if at least one of them is not a singleton, can not unite by record
75
76
          if (!((positive_record->isSingleton())&&(negative_record->isSingleton()))) {
77
            return false;
78
          }
79
80
          //if none of the previous checks failed
81
          return true;
82
       }
83
84
        int return_team_id_of_singleton_record(int record) {
          //@brief - get record, assert it is singleton, pop team, store team id, re-insert team, return saved id.
85
86
          //get proper record
87
          Record* record_singleton = this->find(record);
88
89
90
          //make sure record exists and is actually singleton
          assert(record_singleton != nullptr);
91
92
          assert(record_singleton->isSingleton());
93
          //get team from record
94
95
          NewTeam* team = record_singleton->pop();
96
97
          //save team id
98
          int team_id = team->get_id();
99
100
          //return team to its place in record_singleton
101
          record_singleton->insert(team_id, team);
```

```
//
1
2
     // Created by Guy Friedman on 27/01/2025.
3
4
5
     #ifndef JOCKEY_H
6
     #define JOCKEY_H
7
8
9
10
     class Jockey {
11
     protected:
12
       int id;
13
       int record;
14
       int team_id;
15
     public:
16
       Jockey() = delete;
17
       /*
18
       Jockey(int id) {
19
20
         this->id = id;
21
         this->record = 0;
22
         this->team_id = 0;
23
       }
       */
24
25
26
       Jockey(int id, int team_id) {
         this->id = id;
27
28
         this->record = 0;
29
         this->team_id = team_id;
30
       }
31
32
       virtual ~Jockey() = default;
33
34
       int getId() const {
35
         return this->id;
36
       }
37
       int getRecord() const {
38
         return this->record;
39
40
       }
41
42
       int getTeamId() const {
43
         return this->team_id;
44
       }
45
       void setId(int id) {
46
         this->id = id;
47
48
       }
49
```

```
void setRecord(int record) {
50
        this->record = record;
51
52
      }
53
      void setTeamId(int team_id) {
54
        this->team_id = team_id;
55
56
      }
57
58
      void winMatch() {
59
        ++this->record;
60
      }
61
62
      void loseMatch() {
63
        --this->record;
     }
64
65
   };
66
67
68
69
    #endif //JOCKEY_H
70
```

▼ ChainHashArray.h

```
≛ Download
```

```
//
1
     // Created by Guy Friedman on 24/01/2025.
2
3
4
5
     #ifndef CHAINHASHARRAY_H
     #define CHAINHASHARRAY H
6
7
     #include "DeQue.h"
8
     #include "Pair.h"
9
     #include <cassert>
10
     constexpr int MAX_FILL_RATIO_CHAIN_HASH_ARRAY = 10;
11
     constexpr int INITIAL SIZE CHAIN HASH ARRAY = 16;
12
     #define MAX(a,b) (a>b)?a:b
13
     #define MIN(a,b) (a<b)?a:b
14
     #define MAKE_DOUBLE(a) (a*2)
15
     #define HALF_OF(a) (a*0.5)
16
     #define QUARTER_OF(a) (a*0.25)
17
     #define EMPTY (0)
18
19
20
21
22
     template<class T>
23
     class ChainHashArray {
     private:
24
25
       DeQue<Pair<T>>* data_arr;
26
       int arr_size;
27
       int amount_of_items;
       int capacity;
28
29
     public:
30
31
32
       ChainHashArray(): capacity(EMPTY) {
         this->data arr = nullptr;
33
         this->arr_size = INITIAL_SIZE_CHAIN_HASH_ARRAY;
34
35
         this->amount_of_items = EMPTY;
         this->updateCapacity();
36
37
         this->initializeArray();
38
       }
39
40
       ~ChainHashArray() {
41
         delete[] data_arr;
42
       }
43
       void insert(int key, T* value) {
44
45
         this->insert(key, value, true);
46
       }
47
       T* find(int key) {
48
49
         int index = this->calcIndex(key);
```

```
50
          Pair<T> temp = Pair<T>(key);//new Pair<T>(key);
51
          Pair<T>* toFind = this->data_arr[index].find(temp);
52
          //delete temp;
53
          return to Find == nullptr ? nullptr : to Find->value;
54
       }
55
       T* remove(int key) {
56
          return this->remove(key,true);
57
58
       }
59
       void deleteItem(int key) {
60
          delete this->remove(key);
61
62
       }
63
64
       int size() {
65
          return this->amount_of_items;
66
       }
67
       T* popRandom() {
68
          if (this->amount_of_items == EMPTY) {
69
70
            return nullptr;
71
          }
72
          for (int i = 0; i < this->arr_size; i++) {
73
            DeQue<Pair<T>>& desiredSlot = this->data arr[i];
74
            if (desiredSlot.getSize() != EMPTY) {
              Pair<T>* temp = desiredSlot.pop();
75
76
              --this->amount of items;
              T* tempVal = temp->extract();
77
78
              delete temp;
79
              return tempVal;
80
            }
81
          }
82
          //NOTICE - no check for this->checkUpdateArr();
83
          return nullptr; //if we got here then there is a problem
84
       }
85
       void clear() {
86
          int items_amount = this->amount_of_items;
87
          for (int i = 0; i < items_amount; i++) {
88
89
            this->popRandom();
90
          }
91
          assert(this->amount_of_items == EMPTY);
92
          this->checkUpdateArr();
93
       }
94
95
     protected:
96
97
       void insert(int key, T* value, bool checkForUpdateSize) {
98
          int insertionIndex = this->calcIndex(key);
99
          DeQue<Pair<T>>& desiredSlot = this->data_arr[insertionIndex];
100
          Pair<T>* newItem = new Pair<T>(key, value);
101
          desiredSlot.append(newItem);
```

```
102
         ++this->amount_of_items;
         if (checkForUpdateSize) {
103
104
           this->checkUpdateArr();
105
         }
106
       }
107
108
       /**
109
        *the same functionality as regular insert, but without
110
        *checking for size updates, for inside methods and usages
111
        * @param key
112
        * @return
113
114
        */
       void insertImmediate(int key, T* value) {
115
116
         this->insert(key, value, false);
117
       }
118
119
       void insertImmediate(Pair<T>* newItem) {
120
          this->insertImmediate(newItem->key);
121
       }
122
       T* remove(int key, bool checkForUpdateSize) {
123
124
         int index = this->calcIndex(key);
125
          DeQue<Pair<T>>* toRemove = &(this->data arr[index]);
126
          Pair<T> toRemovePair = Pair<T>(key);
127
128
          Pair<T>* toFind = (toRemove->remove(toRemovePair));
129
         //delete toRemovePair;
         if (toFind == nullptr) {
130
131
           return nullptr;
132
         }
         T* value = toFind->extract();
133
134
         delete toFind;
         --this->amount of items;
135
         if (checkForUpdateSize) {
136
           this->checkUpdateArr();
137
138
         }
139
         return value;
140
       }
141
       /**
142
143
        *the same functionality as regular remove, but without
144
        *checking for size updates, for inside methods and usages
145
        * @param key
146
        * @return
147
        */
148
149
       T* removeImmediate(int key) {
150
          return this->remove(key, false);
151
       }
152
153
```

```
154
       void initializeArray() {
155
          //delete[] this->data arr;
156
          this->data_arr = new DeQue<Pair<T>>[this->arr_size];
157
          for (int i = 0; i < this->arr_size; ++i) {
158
            //this->data_arr[i] = *(new DeQue<Pair<T>>());
159
            DeQue<Pair<T>>& desiredSlot = this->data_arr[i];
160
            desiredSlot.verifyInitialisation();
161
         }
162
       }
163
164
        ChainHashArray(int size) : capacity(EMPTY) {
165
          this->data_arr = nullptr;
166
          this->arr size = MAX(INITIAL SIZE CHAIN HASH ARRAY, size);
167
          this->amount_of_items = EMPTY;
168
          this->updateCapacity();
169
          this->initializeArray();
170
       }
171
172
       void updateCapacity() {
173
          capacity = this->arr_size * MAX_FILL_RATIO_CHAIN_HASH_ARRAY;
174
       }
175
176
       int calcIndex(int key) {
177
          int sheerit = key % this->arr size;
178
          sheerit += this->arr size;
179
          int positive sheerit = sheerit % this->arr size;
180
          return positive sheerit;
181
       }
182
183
       void checkUpdateArr() {
          if (this->amount_of_items == this->capacity) {
184
185
            this->makeBigger();
186
            return;
187
          }
188
          if (this->amount_of_items <= QUARTER_OF(this->capacity)) {
            this->makeSmaller();
189
190
          }
191
       }
192
193
       void resize(int new_capacity) {
          ChainHashArray<T>* other = new ChainHashArray(new_capacity);
194
195
          for (int i = 0; i < this->arr_size; i++) {
196
            DeQue<Pair<T>>& temp = (this->data_arr)[i];
197
            int tempSize = temp.getSize();
            for (int j = 0; j < tempSize; j++) {
198
199
              Pair<T>* templtem = temp.pop();
200
              other->insertImmediate(tempItem->key, tempItem->extract());
201
              delete templtem;
202
            }
203
          }
204
          this->swapData(other);
205
          delete other;
```

```
206
       }
207
       void makeBigger() {
208
209
         this->resize(MAKE_DOUBLE(this->arr_size));
210
       }
211
212
       void makeSmaller() {
213
         if(this->arr_size <= INITIAL_SIZE_CHAIN_HASH_ARRAY) {</pre>
214
           return;
215
         }
216
         this->resize(HALF_OF(this->arr_size));
217
       }
218
219
       template <typename K>
220
       void swap(K& item_1, K& item_2) {
221
         K temp = item_1;
222
         item_1 = item_2;
223
         item_2 = temp;
224
       }
225
226
       void swapData(ChainHashArray* other) {
227
         assert(other != nullptr);
228
         swap<int>(other->arr_size, this->arr_size);
229
         swap<int>(other->amount_of_items, this->amount_of_items);
230
         swap<int>(other->capacity, this->capacity);
231
         swap<DeQue<Pair<T>>*>(other->data_arr, this->data_arr);
232
       }
233
234
235
236
237
238 };
239
240
241
242
     #endif //CHAINHASHARRAY_H
243
```

```
//
1
     // Created by Guy Friedman on 24/01/2025.
2
3
4
5
     #ifndef DEQUE_H
6
     #define DEQUE_H
7
8
     #include "DeQueNode.h"
9
     #define EMPTY (0)
10
11
12
     template <typename T>
13
14
     class DeQue {
15
     protected:
16
       DeQueNode<T>* head;
17
       DeQueNode<T>* tail;
18
       int size;
19
     public:
20
21
       DeQue(): head(new DeQueNode<T>()), tail(new DeQueNode<T>()), size(0) {
22
         //head->addInitial(this->tail);
23
         this->head->next = this->tail;
24
         this->tail->prev = this->head;
25
      }
26
27
       ~DeQue() {
28
         delete head;
29
      }
30
31
       int getSize() const {
32
         return size;
33
      }
34
35
       void append(T* item) {
36
         auto newNode = new DeQueNode<T>(item);
37
         this->tail->queueAdd(newNode);
38
         ++this->size;
39
      }
40
       void insert(T* item) {
41
42
         auto newNode = new DeQueNode<T>(item);
43
         this->head->stackAdd(newNode);
44
         ++this->size;
45
      }
46
47
       T* pop() {
48
         assert(this->head->hasNext());
49
         auto newNode = this->head->popNext();
```

```
T* tempVal = newNode->getData();
50
         newNode->nullify();
51
         delete newNode;
52
         --this->size;
53
         return tempVal;
54
55
       }
56
       T* find(T& value) {
57
58
         auto node = this->head->find(value);
59
         return (node == nullptr)?nullptr:node->getData();
60
       }
61
       T* remove(T& item) {
62
         DeQueNode<T>* temp = this->head->remove(item);
63
64
         if (temp == nullptr) {
65
           return nullptr;
66
         }
         T* newTemp = temp->extract();
67
         temp->verifyDeCouple();
68
         temp->nullify();
69
70
         delete temp;
71
         --this->size;
72
         return newTemp;
73
       }
74
       void verifyInitialisation() {
75
76
         if(this->size != EMPTY) {
77
           return;
78
         }
79
         this->head->next = this->tail;
         this->tail->prev = this->head;
80
81
       }
82
83
84
85
86
87
     };
88
89
90
91
     #endif //DEQUE_H
92
```

▼ DeQueNode.h

```
//
1
     // Created by Guy Friedman on 24/01/2025.
2
3
4
5
     #ifndef DEQUENODE_H
6
     #define DEQUENODE_H
7
8
9
     #include <cassert>
10
11
12
13
14
     template <typename T>
15
     class DeQue;
16
17
     template <typename T>
     class DeQueNode {
18
19
     private:
20
21
       // Declare the templated DeQue as a friend
22
       template <typename U>
23
       friend class DeQue;
24
25
       inline bool initialNode() const {
26
         return (this->next == nullptr) && (this->prev == nullptr);
27
       }
28
29
       inline bool noData() const {
30
         return this->data == nullptr;
31
       }
32
33
       inline bool canBecomeHead() {
34
         return (this->initialNode()) && (this->noData());
35
       }
36
37
       inline bool canBecomeTail() {
38
         return (this->initialNode()) && (this->noData());
39
       }
40
41
     protected:
42
43
       T* data;
       DeQueNode<T>* next;
44
45
       DeQueNode<T>* prev;
46
47
     public:
48
49
       DeQueNode() : data(nullptr), next(nullptr), prev(nullptr) {}
```

```
50
        DeQueNode(T* data) : data(data), next(nullptr), prev(nullptr) {}
51
52
53
        ~DeQueNode() {
54
          delete next;
55
          delete data;
56
       }
57
        void nullify() {
58
          this->next = nullptr;
59
60
          this->prev = nullptr;
61
          this->data = nullptr;
62
       }
63
       void deCouple() {
64
65
          this->next->prev = this->prev;
66
          this->prev->next = this->next;
67
          this->next = nullptr;
68
          this->prev = nullptr;
69
       }
70
71
        void verifyDeCouple() {
72
          if (this->next == nullptr && this->prev == nullptr) {
73
            return;
74
          }
75
          this->deCouple();
76
       }
77
78
        T* extractAndDelete() {
79
          this->deCouple();
80
          T* temp = this->data;
          this->data = nullptr;
81
82
          this->nullify();
83
          //delete this;
84
          return temp;
85
       }
86
        T* extract() {
87
          T* temp = this->data;
88
          this->data = nullptr;
89
90
          return temp;
91
       }
92
93
        inline bool hasNext() const { //fixme problem
94
95
          return !((this->next != nullptr) && (this->next->isTail()));
96
       }
97
98
        inline bool isTail() const {
99
          return (this->prev != nullptr) && (this->data == nullptr) && (this->next == nullptr);
100
        }
101
```

```
102
       inline bool isHead() const {
103
          return (this->next != nullptr) && (this->data == nullptr) && (this->prev == nullptr);
104
       }
105
106
       DeQueNode<T>* popNext() {
107
          assert(this->hasNext());
108
          assert(this->next != nullptr && this->next->next != nullptr);
109
          DeQueNode<T>* temp = this->next;
110
          temp->deCouple();
111
          return temp;
112
       }
113
114
       void gueueAdd(DeQueNode<T>* node) {
115
          assert(node != nullptr);
116
          assert(!this->isHead());
117
          this->prev->next = node;
118
          node->prev = this->prev;
119
          node->next = this;
120
          this->prev = node;
121
       }
122
123
       void stackAdd(DeQueNode<T>* node) {
124
          assert(node != nullptr);
125
          assert(!this->isTail());
126
          this->next->prev = node;
127
          node->next = this->next;
128
          node->prev = this;
129
          this->next = node;
130
       }
131
132
       void addInitial(DeQueNode<T>* node) {
133
          assert(node != nullptr);
134
         //make sure that 'this' node is in a state required to become head
135
          assert(this->canBecomeHead());
136
         //make sure that 'node' node is in a state required to become tail
          assert(node->canBecomeTail());
137
138
          this->next = node;
139
          node->prev = this;
140
       }
141
       T* getData() const {
142
143
          return this->data;
144
       }
145
146
       DeQueNode<T>* find(T& toFind) {
147
          if(this->isTail()) {
148
            return nullptr;
149
          }
150
          if (this->isHead()) {
151
            return this->next->find(toFind);
152
          }
153
          if (*(this->data) == toFind) {
```

```
154
            return this;
155
         }
156
        return this->next->find(toFind);
157
158
159
       DeQueNode<T>* remove(T& toFind) {
160
          if(this->isTail()) {
161
            return nullptr;
162
         }
163
         if (this->isHead()) {
164
            return this->next->remove(toFind);
165
166
         if (*(this->data) == toFind) {
167
            this->deCouple();
168
           return this;
169
         }
170
         return this->next->find(toFind);
171
       }
172
173
       //int getKey() {return this->key;}
174
175
176
177
178 };
179
180
181
182 #endif //DEQUENODE_H
183
```

```
//
1
     // Created by Guy Friedman on 24/01/2025.
2
3
4
5
     #ifndef PAIR_H
6
     #define PAIR_H
7
8
9
     constexpr int DEFAULT_KEY = 0;
10
11
     template <typename T>
12
     class Pair {
     public:
13
14
       int key;
15
       T* value;
16
17
       Pair(int key, T* value): Pair(key) {
18
         this->value = value;
19
       }
20
21
       Pair(int key): Pair() {
22
         this->key = key;
23
       }
24
25
       Pair() : key(DEFAULT_KEY), value(nullptr) {}
26
27
       ~Pair() {
28
         delete this->value;
29
       }
30
       T* extract() {
31
32
         auto value = this->value;
33
         this->value = nullptr;
34
         return value;
35
       }
36
37
       void nullify() {
         this->key = DEFAULT_KEY;
38
         this->value = nullptr;
39
40
       }
41
42
43
       bool operator ==(const Pair& other) const {
44
         return this->key == other.key;
45
       }
46
47
       // Overloaded operator== as a member function
48
       //bool operator==(int otherKey) const {return this->key == otherKey;}
49
```

```
50
       // Friend operator== (int == Pair<int>)
51
       friend bool operator==(int lhs, const Pair& rhs) {
         return rhs == lhs;
52
53
       }
54
55
56
57
58
    };
59
60
61
62
    #endif //PAIR_H
63
```

```
// You can edit anything you want in this file.
1
     // However you need to implement all public Plains function, as provided below as a template
2
3
4
     #include "plains25a2.h"
5
     #include "wet2util.h"
6
     #include "ChainHashArray.h"
7
     #include <new>
8
     #include "lockey.h"
9
     #include "NewTeamArr.h"
10
     #include "Record.h"
11
     #include "RecordArr.h"
12
     #include "Jockey.h"
13
14
     #define MIN(a,b) (a<b)?a:b
15
16
     #define ABS(a) ((a)<0?(-(a)):(a))
17
     Plains::Plains()
18
19
     {
20
21
       RecordArr* records;
22
       NewTeamArr* teams:
23
       ChainHashArray<Jockey>* jockeys;
24
       */
25
       this->records = new RecordArr();
26
       this->teams = new NewTeamArr();
27
       this->jockeys = new ChainHashArray<Jockey>();
28
29
     }
30
     Plains::~Plains()
31
32
33
       delete records:
       delete teams;
34
35
       delete jockeys;
36
     }
37
38
     StatusType Plains::add_team(int teamId)
39
     {
40
       try{
41
         if(teamId<=0) {</pre>
           return StatusType::INVALID_INPUT;
42
43
         }
44
45
         //check if team already exists (don't care if active or not)
46
         NewTeam* newTeam = this->teams->find(teamId);
47
         if (newTeam != nullptr) {
           return StatusType::FAILURE;
48
49
         }
```

```
50
51
          //after we checked that there is no and was no team for 'teamId', create it.
52
          newTeam = new NewTeam(teamId);
53
         //put team in teams
54
          this->teams->insert(teamId, newTeam);
55
56
         //the record for every beginner team
57
         int initial_record = 0;
58
59
         //add team to corresponding record
60
61
         this->records->add_team_to_record(newTeam,teamId,initial_record);
62
63
         return StatusType::SUCCESS;
       } catch (std::bad_alloc& e) {
64
65
          return StatusType::ALLOCATION_ERROR;
66
       }
     }
67
68
69
70
     StatusType Plains::add_jockey(int jockeyId, int teamId)
71
72
       try{
73
         if(jockeyId \le 0)  {
74
            return StatusType::INVALID_INPUT;
75
         }
76
77
         //make sure team is valid, without path modification
         if(!this->teams->check active immediate(teamId)) {
78
79
            return StatusType::FAILURE;
80
         }
81
         //make sure jockey doesn't already exist
82
         Jockey* jockey = this->jockeys->find(jockeyId);
83
84
         if(jockey != nullptr) {
            return StatusType::FAILURE;
85
86
         }
87
88
         //create jockey
         jockey = new Jockey(jockeyId,teamId);
89
90
91
         //store jockey in our system
          this->jockeys->insert(jockeyId,jockey);
92
93
94
         //todo increment size of team by 1
95
96
         return StatusType::SUCCESS;
97
       } catch (std::bad_alloc& e) {
98
          return StatusType::ALLOCATION_ERROR;
99
       }
100 }
101
```

```
102
103
     StatusType Plains::update match(int victoriouslockeyId, int losinglockeyId)
104 {
105
       try{
106
         //check input validity
107
         if(victorious)ockeyId<=0 | | losing|ockeyId <= 0 | | victorious|ockeyId == losing|ockeyId) {
            return StatusType::INVALID_INPUT;
108
109
         }
110
111
         //@brief - check jockeys exist, check jockeys are not in the same team, remove teams from their
     records
112
         //@brief - update score for jockeys and teams, put teams in their new records
113
114
         //check that winning jockey exists
115
         Jockey* winningJockey = this->jockeys->find(victoriousJockeyId);
116
         if (winning|ockey == nullptr) {
117
            return StatusType::FAILURE;
118
         }
119
120
         //check that losing jockey exists
121
         lockey* losinglockey = this->jockeys->find(losinglockeyId);
122
         if (losinglockey == nullptr) {
123
            return StatusType::FAILURE;
124
         }
125
         //get team of winning jockey
126
127
          NewTeam* winningTeam = this->teams->get_root_of(winningJockey->getTeamId());
128
         //get team of losing jockey
129
          NewTeam* losingTeam = this->teams->get root of(losinglockey->getTeamId());
130
131
132
         //make sure it is not the same group
133
         if (winningTeam==losingTeam) {
134
            return StatusType::FAILURE;
135
         }
136
137
         //make sure no funny business is going on
138
          assert(winningTeam != nullptr);
139
          assert(losingTeam != nullptr);
140
         //remove each team from their record
141
142
          this->records->remove_team_from_record(winningTeam->get_id(),winningTeam->get_record());
          this->records->remove_team_from_record(losingTeam->get_id(),losingTeam->get_record());
143
144
145
         //update scores
146
         winningJockey->winMatch();
147
         losingJockey->loseMatch();
          winningTeam->winMatch();
148
149
         losingTeam->loseMatch();
150
151
         //store teams in their new records
```

```
152
         this->records->add_team_to_record(winningTeam,winningTeam->get_id(),winningTeam-
     >get_record());
153
         this->records->add_team_to_record(losingTeam,losingTeam->get_id(),losingTeam->get_record());
154
155
         return StatusType::SUCCESS;
156
       } catch (std::bad_alloc& e) {
157
         return StatusType::ALLOCATION_ERROR;
158
       }
159 }
160
161
     StatusType Plains::merge_teams(int teamId1, int teamId2)
162
163
     {
164
       try{
165
         if(teamId1<=0||teamId2<=0||teamId1==teamId2) {
166
           return StatusType::INVALID INPUT;
167
         }
168
169
         //make sure team1 is active
170
         if (!this->teams->team_active(teamId1)) {
171
           return StatusType::FAILURE;
172
         }
173
174
         //make sure team2 is active
175
         if (!this->teams->team_active(teamId2)) {
176
           return StatusType::FAILURE;
177
         }
178
179
         //retrieve team1 and team2
180
         NewTeam* teamOne = this->teams->get root of(teamId1);
181
         NewTeam* teamTwo = this->teams->get_root_of(teamId2);
182
183
         //make sure no funny business is happening
184
         assert(teamOne->get id() == teamId1);
185
         assert(teamTwo->get_id() == teamId2);
186
187
         //if they are the same team - cant merge
188
         if (teamOne == teamTwo) {
189
           return StatusType::FAILURE;
         }
190
191
192
         //remove each team from their record
193
         this->records->remove_team_from_record(teamId1,teamOne->get_record());
194
         this->records->remove_team_from_record(teamId2,teamTwo->get_record());
195
196
         //use logic in TeamArr to handle merging of teams
197
         this->teams->unite_teams(teamId1,teamId2);
198
199
         //make sure all is going good and no funny business, should be ok if merge went ok in TeamArr.
200
         assert(teamOne->get_id() == teamTwo->get_id());
201
```

```
202
         //now ID will be updated in both teams, get the new root (would automatically select either
     teamOne or teamTwo, according to the merge)
203
          NewTeam* newRoot = this->teams->get_root_of(teamOne->get_id());
204
205
         //store merged team in the appropriate record
206
         this->records->add_team_to_record(newRoot, newRoot->get_id(), newRoot->get_record());
207
208
         return StatusType::SUCCESS;
       } catch(std::bad_alloc& e){
209
         return StatusType::ALLOCATION_ERROR;
210
211
       }
212 }
213
214
215
     StatusType Plains::unite_by_record(int record)
216
217
       try{
218
         if (record<=0) {
            return StatusType::INVALID_INPUT;
219
220
         }
221
222
         //make sure that we can unite by record
223
         if (!this->records->can_unite_by_record(record)) {
224
            return StatusType::FAILURE;
         }
225
226
227
         //get id of teams in singleton records
228
         int positive_team_id = this->records->return_team_id_of_singleton_record(record);
229
         int negative_team_id = this->records->return_team_id_of_singleton_record(-record);
230
231
         //utilise existing logic and code to handle merging execution
232
         return this->merge_teams(positive_team_id,negative_team_id);
233
234
       } catch(std::bad alloc& e){
235
          return StatusType::ALLOCATION_ERROR;
236
       }
237 }
238
239
240
     output_t<int> Plains::get_jockey_record(int jockeyId)
241
     {
242
       try{
243
         if (jockeyId<=0) {
244
            return StatusType::INVALID_INPUT;
245
246
         Jockey* jock = this->jockeys->find(jockeyId);
247
         if (jock==nullptr) {
248
            return StatusType::FAILURE;
249
         }
250
         return jock->getRecord();
251
       } catch(std::bad_alloc& e) {
252
          return StatusType::ALLOCATION_ERROR;
```

```
253
     }
254 }
255
256
     output_t<int> Plains::get_team_record(int teamId)
257
258 {
259
       try{
         if (teamId<=0) {
260
261
           return StatusType::INVALID_INPUT;
262
         }
263
         //make sure team is active without path modifications
264
265
         if (!this->team_active(teamId)) { //check_active_immediate
266
           return StatusType::FAILURE;
267
         }
268
         //get root of team
269
270
         NewTeam* newTeam = this->teams->get_root_of(teamId);
271
         //make sure no funny business is going on
272
273
         assert(newTeam->get_id() == teamId);
274
         //the record of our team
275
276
         int record = newTeam->get_record();
277
278
         //return it
279
         return record;
280
281
       } catch(std::bad_alloc& e) {
282
         return StatusType::ALLOCATION_ERROR;
283
       }
284 }
285
```

```
//
1
     // 234218 Data Structures 1.
2
3
     // Semester: 2025A (Winter).
    // Wet Exercise #1.
4
5
     // The following header file contains all methods we expect you to implement.
6
7
     // You MAY add private methods and fields of your own.
8
     // DO NOT erase or modify the signatures of the public methods.
9
     // DO NOT modify the preprocessors in this file.
     // DO NOT use the preprocessors in your other code files.
10
11
     //
12
     #ifndef PLAINS25WINTER_WET1_H_
13
14
     #define PLAINS25WINTER WET1 H
15
16
     #include "wet2util.h"
17
     #include "ChainHashArray.h"
     #include <new>
18
19
20
     #include "Jockey.h"
     #include "NewTeamArr.h"
21
22
     #include "Record.h"
     #include "RecordArr.h"
23
24
25
     class Plains {
     private:
26
27
       //
28
       // Here you may add anything you want
29
       //
30
       RecordArr* records:
31
       NewTeamArr* teams;
32
       ChainHashArray<Jockey>* jockeys;
33
     public:
34
35
       // <DO-NOT-MODIFY> {
36
       Plains();
37
38
       ~Plains();
39
40
       StatusType add_team(int teamId);
41
42
       StatusType add_jockey(int jockeyId, int teamId);
43
44
       StatusType update_match(int victorious|ockeyld, int losing|ockeyld);
45
46
       StatusType merge_teams(int teamId1, int teamId2);
47
48
       StatusType unite_by_record(int record);
49
```

```
50    output_t<int> get_jockey_record(int jockeyId);
51
52    output_t<int> get_team_record(int teamId);
53    // } </DO-NOT-MODIFY>
54    };
55
56    #endif // PLAINS25WINTER_WET1_H_
57
```

```
//
1
     // Created by Guy Friedman on 26/01/2025.
2
3
4
5
     #ifndef RECORD H
6
7
     #define RECORD_H
8
     #include "ChainHashArray.h"
9
     #include "NewTeam.h"
10
11
     typedef NewTeam what_to_hold;
12
     class Record {
13
14
     protected:
15
       ChainHashArray<what_to_hold>* held_items;
16
       int record_value;
17
     public:
       Record() = delete;
18
19
20
       Record(int record_value): record_value(record_value) {
21
         this->held_items = new ChainHashArray<what_to_hold>();
22
       }
23
24
       ~Record() {
25
         this->held_items->clear();
26
         delete this->held_items;
27
       }
28
29
       bool isEmpty() {
30
         return this->held_items->size() == 0;
31
       }
32
33
       bool isSingleton() {
         return this->held_items->size() == 1;
34
35
       }
36
37
       what_to_hold* pop() {
38
         return this->held_items->popRandom();
39
       }
40
41
       what_to_hold* remove(int key) {
42
         return this->held_items->remove(key);
43
       }
44
45
       void insert(int key, what_to_hold* value) {
46
         this->held_items->insert(key, value);
47
       }
48
49
       int get_records_val() {
```

```
return this->record_value;
50
51
       }
52
       int get_singleton_team_id() { //use is depreceated, already implemented in RecordArr
53
         //todo - pop only team (assert this is singleton), save team id, re insert team with team id, return
54
     team id
55
         assert(this->isSingleton());
56
57
         //fixme
58
         return 0;
59
       }
60
61
62
63
    };
64
65
66
67
     #endif //RECORD_H
68
69
```