Report on implementation of redis

Github - https://github.com/mkv4537/implementation-of-redis1

1. Why did i chose c++ language?

Ans :- i am proficient in c++, because standard library function of c++ gives a wide area to implement some concepts of sorting required. but mainly i focus on data structure and algorithm. I am open towards any language.

2. What are the further improvements that can be made to make it efficients?

-> i have written all code in a single file, later i can i use class, and with the help of low level design . make code more clear and understandable, and taht will be easy if i want to modifysome thing.

3. What data structures i have used and why?

-> for implementing different functionalities i have used different data structures

1. For set , get, setex, setnx implementation , i have used unordered_map , because we can add, remove, and get element in o(1) time complexity.

2. For implementation of zadd, zrange, and zrank, i have used skip list data structure and unordered_map, because we can search , add, remove element in o(log n), where n is size of element.

4. Does your implementation support multithreaded operation, if no why?

-> this code support singal threading, because we are not giving job and query simultaneously, we are giving one after other, but, for expiry i have detached the thread to run independently nd it will get deleted if time is equal to expire time, it is getting checked at every second.

Implemetation code ->

```
#include<bits/stdc++.h>
using namespace std;

struct Node{
    int data;
    string value;
    Node *next, *prev, *down, *up;
    Node(int val, string v){
        data = val;
        value = v;
        next = NULL;
        prev = NULL;
        down = NULL;
        up = NULL;
    }
};
unordered_map<string, pair<string, long long>>get_string;

unordered_map<long long, unordered_set<string>>timeToLive;

unordered_map<string, Node*>sorted_set;
unordered_map<string, unordered_map<string, long long>>iszset;
bool joinable();
void detach();
void join();
void fn()
{
        int curr_time = time(NULL);
        if (timeToLive.find(curr_time) != timeToLive.end()){
            unordered_set<string> s = timeToLive[curr_time];
            for (auto it : s){
                get_string.erase(it);

            }
            timeToLive.erase(curr_time);
        }


}
```

```
void timer_start(std::function<void(void)> func, unsigned int interval)
{
    std::thread([func, interval]() {
        while (true)
        {
            func();
            std::this_thread::sleep_for(std::chrono::milliseconds(interval));
        }
    }).detach();
}


void expire()
{

  timer_start(fn, 1000);
  while(true);
}
void rkeysAll(){
    for (auto it : get_string){
        cout<<it.first<<endl;
    }
    if (get_string.size() == 0)
        cout<<"empty list or set"<<endl;
}
// setting calue
string rset(string s1, string s2){

    if (get_string.find(s1) != get_string.end() && get_string[s1].second != -1){
        timeToLive[get_string[s1].second].erase(s1);
    }
    get_string[s1] = {s2, -1};
    return "OK";
}
// appending value
string rappend (string s1, string s2){
    pair<string, long long>p = get_string[s1];
    get_string[s1] = {p.first + s2, p.second};
    return to_string(get_string[s1].first.size());
}
// modifying value
string rchange (string s1, int value){
    if (get_string.find(s1) == get_string.end())
```

```
        return "does not exist";
    long long num = 0, sign = 1;
    string curr = get_string[s1].first;
    if (curr[0] == '-'){
     curr = curr.substr(1);
     sign = -1;
    }
    for (char c : curr){
        if (c >= '0' && c <= '9'){
          num = num * 10 + c - '0';
        }else{
            return "it is not number";
        }
    }
    get_string[s1] = {to_string(value + sign * num),get_string[s1].second};
    return to_string(sign * num + value);
}
// string length
string rstrlen (string s){
    if (get_string.find(s) != get_string.end()){
        return to_string(get_string[s].first.size());
    }
    return "0";
}
// setting value with expiry date
string rsetex(string s1, int ttl, string s2){

    if (get_string.find(s1) != get_string.end() && get_string[s1].second != -1){
        timeToLive[get_string[s1].second].erase(s1);
    }
    int curr_time = time(NULL);
    get_string[s1] = {s2, ttl + curr_time};
    timeToLive[ttl + curr_time].insert(s1);
    return "OK";
}
// expire the value as it goest out of time
string rttl(string s){
    int curr_time = time(NULL);
    if (get_string.find(s) != get_string.end()){
        if (get_string[s].second == -1){
            return "untill you will delete";
        }else{
            return to_string(curr_time - get_string[s].second);
```

```
        }
    }
    return "value does not exist";
}
// doesnot add the value, if already present
string rsetnx(string s1, string s2){

    if (get_string.find(s1) != get_string.end()){
        return "0";
    }

    get_string[s1] = {s2, -1};
    return "1";
}
// getting the value, get mothod
string rget (string s1){

    if (get_string.find (s1) != get_string.end()){
        return get_string[s1].first;
    }
    return "Nil";
}
// geeting pointer for zadd
Node* getnewlevel(){
        Node *head = new Node(INT_MIN, "nil"); Node *tail = new Node(INT_MAX, "nil");
        head->next = tail;
        tail->prev = head;
        return head;
}
// zadd method
string rzadd(string s){


    int count = 0;
    if (sorted_set.find(s) == sorted_set.end()){
        sorted_set[s] = getnewlevel();
        iszset[s] = unordered_map<string, long long>();
    }
    while (1){

        count++;
```

```
Node *head = sorted_set[s];
int score; string value;
cin>>score;
if (score == -1)
    break;
cin>>value;
if (iszset[s].find(value) != iszset[s].end()) continue;
iszset[s][value] = score;
Node *curr = head;

while (curr->down){
    while (curr->next->data <= score){
        curr = curr->next;
    }
    curr = curr->down;
}

while (curr->next->data <= score){
    curr = curr->next;
}

Node *temp = new Node(score, value), *temp1 = curr->next;
temp1->prev->next = temp;
temp->prev = temp1->prev;
temp->next = temp1;
temp1->prev = temp;

while (temp->prev->up == NULL && temp->next->up == NULL){
    Node *temp2 = new Node(temp->data, temp->value);
    curr = temp;
    while (curr->prev && curr->prev->up == NULL){
        curr = curr->prev;
    }
    if (curr->prev == NULL){
        curr->up = getnewlevel();
        curr->up->down = curr;
        Node *curr1 = temp;
        while (curr1->next){
            curr1 = curr1->next;
        }
        curr1->up = curr->up->next;
        curr1->up->down = curr1->up;
        head = curr->up;
```

```
            sorted_set[s] = head;
            break;
        }
        else{
            Node *temp3 = curr->prev->up->next;
            curr->prev->up->next = temp2;
            temp2->prev = curr->prev->up;
            temp3->prev = temp2;
            temp2->next = temp3;
        }
        temp->up = temp2;
        temp2->down = temp;
        temp = temp2;
    }
    }
    return to_string(count-1);
}
// zrange method
void rzrange(string s, int start, int end){

    if (sorted_set.find(s) == sorted_set.end()) {
        cout<<"Key does not exist";
        return;
    }
    Node *curr = sorted_set[s];


    if (end == -1) end = INT_MAX - 1;
    while (curr->down) {

        while (curr->next->data < start){
            curr = curr->next;
        }
        curr = curr->down;

    }
    curr = curr->next;
    while (curr->data <= end){
        cout<<curr->value<<endl;
        curr = curr->next;
    }
    curr = sorted_set[s];
    while (curr->down) curr = curr->down;
```

```
    while(curr){
        cout<<curr->data <<" "<< curr->value<<" ";
        curr = curr->next;
    }
}
// zrank function
string rzrank(string s, string val){
    if (iszset.find(s) == iszset.end()) return "set does not exist";
    if (iszset[s].find(val) == iszset[s].end()) return "value does not exist in set";
    int score = iszset[s][val];
    Node *curr = sorted_set[s];
    while (curr->down) {

        while (curr->next->data < score){
            curr = curr->next;
        }
        curr = curr->down;

    }
    int level = 0;
    while (curr->data <= score && curr->value != val){
        curr = curr->next;
    }
    if (curr->data > score){
        return "value does not exist";
    }
    int ans = 0;
    while (curr){
        while (curr->up){
            level++;
            curr = curr->up;
        }
        if (curr->prev)
        ans += pow(2, level);
        curr = curr->prev;

    }
    return to_string(ans);
    return "kd";
}
// delete method
string rdel (string s1){
```

```
    if (get_string.find(s1) != get_string.end()){
        if (get_string[s1].second != -1){
            timeToLive[get_string[s1].second].erase(s1);
        }
        get_string.erase(s1);
        return "1";
    }
    return "0";
}
// clearing map
string rflushAll (){
    get_string.clear();
    timeToLive.clear();
    iszset.clear();
    sorted_set.clear();
    return "ok";
}
// for use switch method
int get_function(string s){
    if (s == "set")
        return 1;
    if (s == "get")
        return 2;
    if (s == "del")
        return 3;
    if (s == "key_*")
        return 4;
    if (s == "flush_all")
        return 5;
    if (s == "setex" || s == "psetex")
        return 6;
    if (s == "ttl")
        return 7;
    if (s == "setnx")
        return 8;
    if (s == "strlen")
        return 9;
    if (s == "mset")
        return 10;
    if (s == "incrby" || s == "decrby" || s == "incr" || s== "decr")
        return 11;
    if (s == "append")
        return 12;
```

```
    if (s == "zadd")
        return 13;
    if (s == "zrange")
        return 14;
    if (s == "zrank")
        return 15;
    return 0;
}

int main(){

     // thread for independent
     thread expiry(expire);

     expiry.detach();
     if (!expiry.joinable()){
     while (1){
        string s, s1, s2;
        cout<<"Enter command"<<endl;
        cin>>s;
        int value = get_function (s), val = 1;
        switch (value){
            case 1:
                //string s1, s2;
                cin>>s1>>s2;
                cout<<rset(s1, s2)<<endl;
                break;
            case 2:
                //string s1;
                cin>>s1;
                cout<<rget(s1)<<endl;
                break;
            case 3:
                //string s1;
                cin>>s1;
                cout<<rdel(s1)<<endl;
                break;
            case 4:
                rkeysAll();
                break;
            case 5:
                cout<<rflushAll()<<endl;
                break;
```

```
case 6:
    int delay;
    cin>>s1>>delay>>s2;
    if (s == "psetex")
        delay = delay/1000;
    cout<<rsetex(s1, delay, s2)<<endl;
    break;
case 7:
    cin>>s1;
    cout<<rttl(s1)<<endl;
    break;
case 8:
    cin>>s1>>s2;
    cout<<rsetnx(s1, s2)<<endl;
    break;
case 9:
    cin>>s1;
    cout<<rstrlen(s1)<<endl;
    break;
case 10:
    while (1){
        cin>>s1>>s2;
        if (s1 == "-1")
            break;
        cout<<rset(s1, s2)<<endl;
    }
    break;
case 11:
    cin>>s1;
    if (s == "decr"){
        val = -1;
    }
    if (s == "decrby"){
        cin>>val;
        val = -val;
    }
    if (s == "incrby")
        cin>>val;
    cout<<rchange (s1, val)<<endl;
    break;
case 12:
    cin>>s1>>s2;
    cout<<rappend (s1, s2)<<endl;
```

```
            break;
        case 13:
            cin>>s1;
            cout<<rzadd (s1)<<endl;
            break;
        case 14:
            int end;
            cin>>s1>>val>>end;
            rzrange (s1, val, end);
            break;
        case 15:
            cin>>s1>>s2;
            cout<<rzrank (s1, s2)<<endl;
            break;
        default:
            cout<<"not valid"<<endl;
      }
   }
    }else{
       expiry.join();
    }

   return 0;
}
```