Homework #2

Name: Purnabhishek Sripathi

Email: ps747@nau.edu

Userid: 6274051

Problem #1 (of 1)

1. A default constructor (that zeroes everything out)

```
99 class Queries_AR
100
101 public:
102
           vector<array<char,32>> queries; // Vector for query storage, with each query being a 32-character array
103
           long long int size; //Declaring the variable to store the size
104
105
           Queries_AR() // Default Constructor
106
107
108
          Queries_AR(char *path) //Custom Constructor that takes file path as input and retrieves queries from a text document
109
110
111
               long long int fileSize = get_size(path); // getting the size of the file
cout<<"queries fileSize:"<<fileSize<<endl; // printing the size of the file</pre>
112
113
114
               // fflush(stdout);
               queries.reserve(fileSize/45); //Reserving the space in the vector by file size & expected entry length
115
               // cout<<"check1 \n";
116
117
               readDataSet(path);
118
```

2. At least one custom constructor (e.g. one taking a file path or ifstream as input)

```
99
     class Queries_AR
100
     public:
101
102
          vector<array<char,32>> queries; // Vector for query storage, with each query being a 32-character array
103
          long long int size; //Declaring the variable to store the size \,
104
          Queries_AR() // Default Constructor
105
106
107
108
          Queries_AR(char *path) //Custom Constructor that takes file path as input and retrieves queries from a text document
109
111
               long long int fileSize = get_size(path); // getting the size of the file
cout<<"queries fileSize:"<<fileSize<<endl; // printing the size of the file</pre>
112
114
               // fflush(stdout);
               queries.reserve(fileSize/45); //Reserving the space in the vector by file size & expected entry length
115
               // cout<<"check1 \n";</pre>
116
117
               readDataSet(path);
118
```

3. A function to read the query dataset file.

```
109
          Queries_AR(char *path) //Custom Constructor that takes file path as input and retrieves queries from a text document
110
111
              long long int fileSize = get_size(path); // getting the size of the file
cout<<"queries fileSize:"<<fileSize<<endl; // printing the size of the file</pre>
112
113
              // fflush(stdout);
114
115
              queries.reserve(fileSize/45); //Reserving the space in the vector by file size & expected entry length
              // cout<<"check1 \n";
116
117
              readDataSet(path);
118
119
          void readDataSet(char* path){ // Function to read the query dataset file
120
121
              ifstream queryFile(path); // opening the file with given path
122
              int fileSize;
123
              string currLine; //Declaring the string currLine to store each line read from the file
              if (!queryFile.is_open())
124
              cout << "err: unable to open the file"; //print the error if the file cannot be opened
125
              bool val=false; //Declaring the boolean variable to alternate reading lines
126
              // cout<<"check2 \n"
127
              while (getline(queryFile, currLine)){ // while loop to read the lines from the file
129
130
                  if(val)
131
                  continue:
132
                  array<char,32> arr; // array to store the current query
133
                  fileSize=33;
                  while(--fileSize){
134
135
                      arr[32-fileSize]=currLine[32-fileSize]; //Copying the chars from currLine to array
136
137
                  queries.emplace_back(arr); //Adding query to vector of queries
138
              queries.shrink_to_fit(); //Shrink the vector to fit the actual number of elements
140
              // cout<<"check3 \n";
141
              fflush(stdout);
              cout << "\nqueries read successfully: " << queries.size();// Printing the no:of queries read</pre>
142
143
              queryFile.close();// Closing the file
144
```

4. A search function designed to find a sequence fragment within class's data.

```
145
          bool linear search(char *c){ // Linear search for a query
146
              for(int i=0;i<queries.size();++i){ // Iterating through the queries</pre>
147
148
                  temp=-1:
                  while((++temp)<32&&queries[i][temp]==c[temp]); // Comparing the each character of the query
149
150
                  if(temp==32)
                  return true; // Returning true if all characters match
151
152
              return false; // Returning false if no match is found
153
154
          bool binary_search(char *c){ // Binary search for a query in the vector
155
              long long int start=0,end=queries.size();
156
157
              int temp; //declaring the temp variable
158
              while(start<=end){
159
                  long long int mid= (end-start)/2+start; // calculating the middle index
                  temp=-1; // initializing the temp to -1
160
                  while((++temp)<32&&queries[mid][temp]==c[temp]); // while loop to compare each character of the query
161
162
                  if(temp==32) // if statement to return true if all characters match
163
                  return true;
164
                  else if(queries[mid][temp]<c[temp]) // Move to the right half
165
                  start=mid+1;
166
                  else // Move to the left half
167
                  end=mid-1;
168
169
              return false; //return false if no match is found
```

5. A function to sort the fragments of the Queries_AR object.

```
170
171
          void sortQueries(){ // function to sort the fragments of the Queries AR object.
              // cout<<"before sort"<<endl;</pre>
172
173
              // for(int i=0;i<10;++i){
                      cout<<string(queries[i].begin(),queries[i].end())<<endl;</pre>
174
              // }
175
176
              //sort(queries.begin(),queries.end());
177
              // cout<<"after sort"<<endl;</pre>
178
              // for(int i=0;i<10;++i){
179
                      cout<<string(queries[i].begin(),queries[i].end())<<endl;</pre>
              //
              // }
180
              merge sort(0, queries.size() - 1, queries); // merge sort method calling
181
182
183
          ~Queries_AR(){} // queries_ar destructor
184
185
```

6. A destructor

```
~Queries_AR(){} // <u>queries_ar destructor</u>
184
     };
185
     class GNOME
186
187
     {
188
         public:
189
          char *data = nullptr; // data pointer to store the gnome data
          long long int size = 0; // size variable to store the total size of the data
190
          GNOME(char *path) // Constructor that reads gnome data from a file
191
192
              unsigned long long fileSize = get_size(path); // Getting size of the file and storing it in filesize
193
194
              cout<<"\ngnome size"<< fileSize<<endl; // Printing the size of the file</pre>
195
              fflush(stdout); // Flush the output buffer
196
              data = (char *)malloc(fileSize); // Allocating memory for the gnome data
197
              ifstream queryFile(path); // Open the file with the given path
198
              string currLine; // declaring string to store the each line read from the file
              currLine.reserve(100); // reversing the space for 100 characters
199
200
              if (!queryFile.is open()) // condition to check if the file failed to open
                  cout << "err: unable to open the file"; // print error message
201
              while (getline(queryFile, currLine)) //reading the file line by line until end of the file
202
203
204
                  if (currLine[0] != '>'){ // condition to check is the line doesnot start with > }
205
                      memcpy(data + size, currLine.c_str(), currLine.size()); //copying the current line content
206
                      size+=currLine.size(); // incrementing the size
207
208
209
              cout<<"total chars read from gnome file:"<<size<<endl; // printing the total chars read from the gnome file
              queryFile.close(); // closing the file
210
211
212
          ~GNOME() // destructor to free the allocated memory for gnome
213
              free(data); // free the allocated memory
214
215
216
```

1. How long did it take you to search for the first 10K, 100K, and 1M 32-character long fragments of the <u>subject dataset</u> within the <u>query dataset</u>?

Ans:

Time taken to search for the first 10K:

18624 seconds

Approximately it takes 5.1733333 hours.

Time taken to search for the first 100K:

181741 seconds

Approximately it takes 50.4836111 hours.

Time taken to search for the first 1M: 504.836111 hours approx.

I have calculated the approximate calculation for 1M is as below:

Time taken to search for the first 1M = (Time taken for the first 100K fragments) * <math>(1M / 100K)

Time taken to search for first 1M fragments = 50.4836111 hours * (1,000,000 / 100,000)

Time taken to search for first 1M fragments = 50.4836111 hours * 10

Time taken to search for first 1M fragments = 504.836111 (21 days approximately)

2. How long would it take to search for every possible 32-character long fragment of the <u>subject dataset</u> within the <u>query dataset</u>? Please note that depending on the efficiency of your algorithm, this step may take a long time. If the total time is greater than 24 CPU hours, provide an estimate rather than an exact number.

Ans:

The time required for every possible 32-character long fragment within the query data set, it takes approximately **1543378.21 hours**.

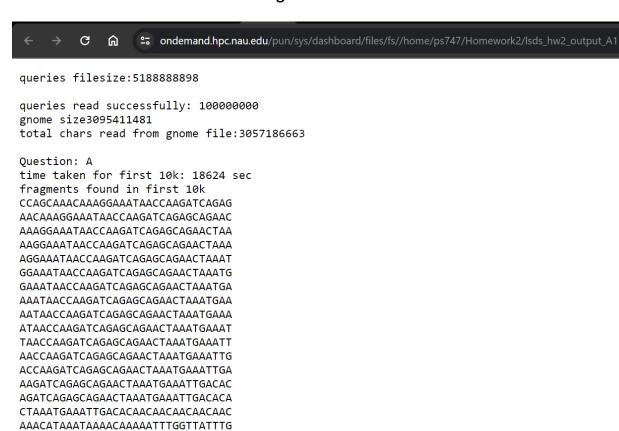
To translate it in days it would take **64307.43 days** and if we converted it into years, it would take **176.18 years** to compute.

As I have used Linear search here takes these many years to run the program.

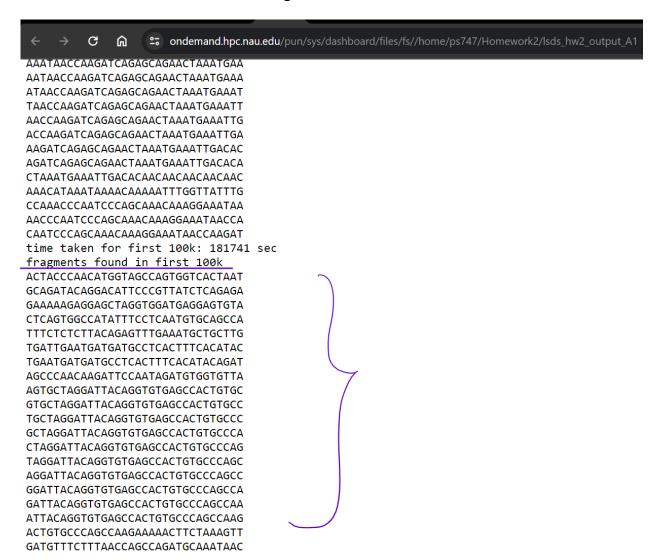
 Print the 20 query fragments found within the <u>subject dataset</u> that have the largest indices (i.e. found later in the subject) for the first 10K, 100K, and 1M 32-character long fragments.

Screenshot that shows the last 20 fragments for 10k:

CCAAACCCAATCCCAGCAAACAAAGGAAATAA AACCCAATCCCAGCAAACAAAGGAAATAACCA CAATCCCAGCAAACAAAGGAAATAACCAAGAT



Screenshot that shows the last 20 fragments for 10k:



For 1M last 20 fragments:

The program is still running for the 1M 32-character long fragments. Due to this reason, I do not have the last 20 fragments for the 1M character long fragments.

B. (30 pts)

1: How long did it take you to search for the first 10K, 100K, and 1M 32-character long fragments of the <u>subject dataset</u> within the <u>query dataset</u>?

Ans:

Time taken to search for the first 10K: 0.074871 seconds.

Time taken to search for the first 100K: 0.625749 seconds.

Time taken to search for the first 1M: 4.95256 seconds.

2: How long would it take to search for every possible 32-character long fragment of the <u>subject dataset</u> within the <u>query dataset</u>? Please note that depending on the efficiency of your algorithm, this step may take a long time. If the total time estimate is greater than 24 CPU hours, provide an estimate rather than exact number.

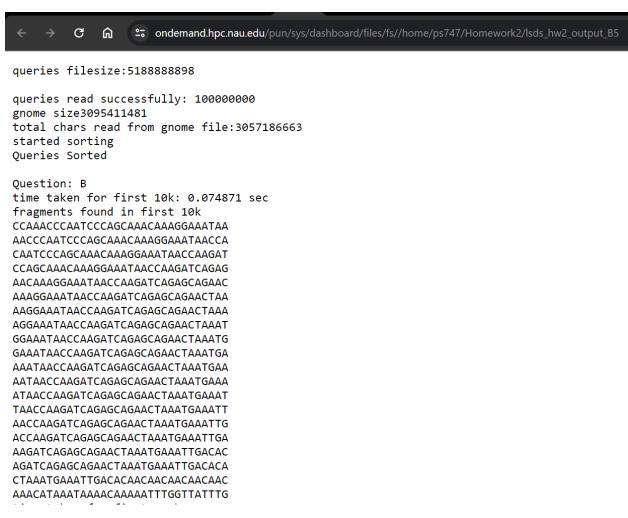
Total time taken is: 13519.3 seconds and it is approximately **3.7552778 hours.**



3: Print the 20 query fragments found within the <u>subject dataset</u> that have the largest indices (i.e. found later in the subject) for the first 10K, 100K, and 1M 32-character long fragments.

Ans:

Below is the screenshot for the 20 query fragments found within the first 10K.



Below is the screenshot for the 20 query fragments found within the first 100K.



time taken for first 100k: 0.625749 sec fragments found in first 100k AGTGCTAGGATTACAGGTGTGAGCCACTGTGC GTGCTAGGATTACAGGTGTGAGCCACTGTGCC TGCTAGGATTACAGGTGTGAGCCACTGTGCCC GCTAGGATTACAGGTGTGAGCCACTGTGCCCA CTAGGATTACAGGTGTGAGCCACTGTGCCCAG TAGGATTACAGGTGTGAGCCACTGTGCCCAGC AGGATTACAGGTGTGAGCCACTGTGCCCAGCC GGATTACAGGTGTGAGCCACTGTGCCCAGCCA GATTACAGGTGTGAGCCACTGTGCCCAGCCAA ATTACAGGTGTGAGCCACTGTGCCCAGCCAAG ACTGTGCCCAGCCAAGAAAACTTCTAAAGTT GATGTTTCTTTAACCAGCCAGATGCAAATAAC ACTACCCAACATGGTAGCCAGTGGTCACTAAT GCAGATACAGGACATTCCCGTTATCTCAGAGA GAAAAAGAGGAGCTAGGTGGATGAGGAGTGTA CTCAGTGGCCATATTTCCTCAATGTGCAGCCA TTTCTCTCTTACAGAGTTTGAAATGCTGCTTG TGATTGAATGATGCCTCACTTTCACATAC TGAATGATGCCTCACTTTCACATACAGAT AGCCCAACAAGATTCCAATAGATGTGGTGTTA

Below is the screenshot for the 20 query fragments found within the first 1M.

← → C ← condemand.hpc.nau.edu/pun/sys/dashboard/files/fs//home/ps747/Homework2/lsds_hw2_output_B5 AGCCCAACAAGATTCCAATAGATGTGGTGTTA time taken for first 1 million: 4.95256 sec fragments found in first million

ATCCTGAGGGCCAGGTGCAGTGGCTCACGCCT AGGGCCAGGTGCAGTGGCTCACGCCTGTAATC GCCAGGTGCAGTGGCTCACGCCTGTAATCACA GTGCAGTGGCTCACGCCTGTAATCACAGCACT TGCAGTGGCTCACGCCTGTAATCACAGCACTT GCAGTGGCTCACGCCTGTAATCACAGCACTTT CAGTGGCTCACGCCTGTAATCACAGCACTTTG AGTGGCTCACGCCTGTAATCACAGCACTTTGG GCCCAGGAGTTTGAGACCAACCTGGGCAACAT CCCAGGAGTTTGAGACCAACCTGGGCAACATG AGGAGTTTGAGACCAACCTGGGCAACATGGCA ATTAGCCGGGTGTGGTGGCATGTGCCTGTAGT AGCCGGGTGTGGTGGCATGTGCCTGTAGTTCC CCGGGTGTGGTGGCATGTGCCTGTAGTTCCAG GGTGGGAGGATCACTTGACCCTAGGAGGACAA CTCCAGCCTGGACGACAGAGTGAGACTCTGTC TCCAGCCTGGACGACAGAGTGAGACTCTGTCT CCAGCCTGGACGACAGAGTGAGACTCTGTCTC CAGCCTGGACGACAGAGTGAGACTCTGTCTCA GCCTGGACGACAGAGTGAGACTCTGTCTCAAA total time taken: 13519.3 sec