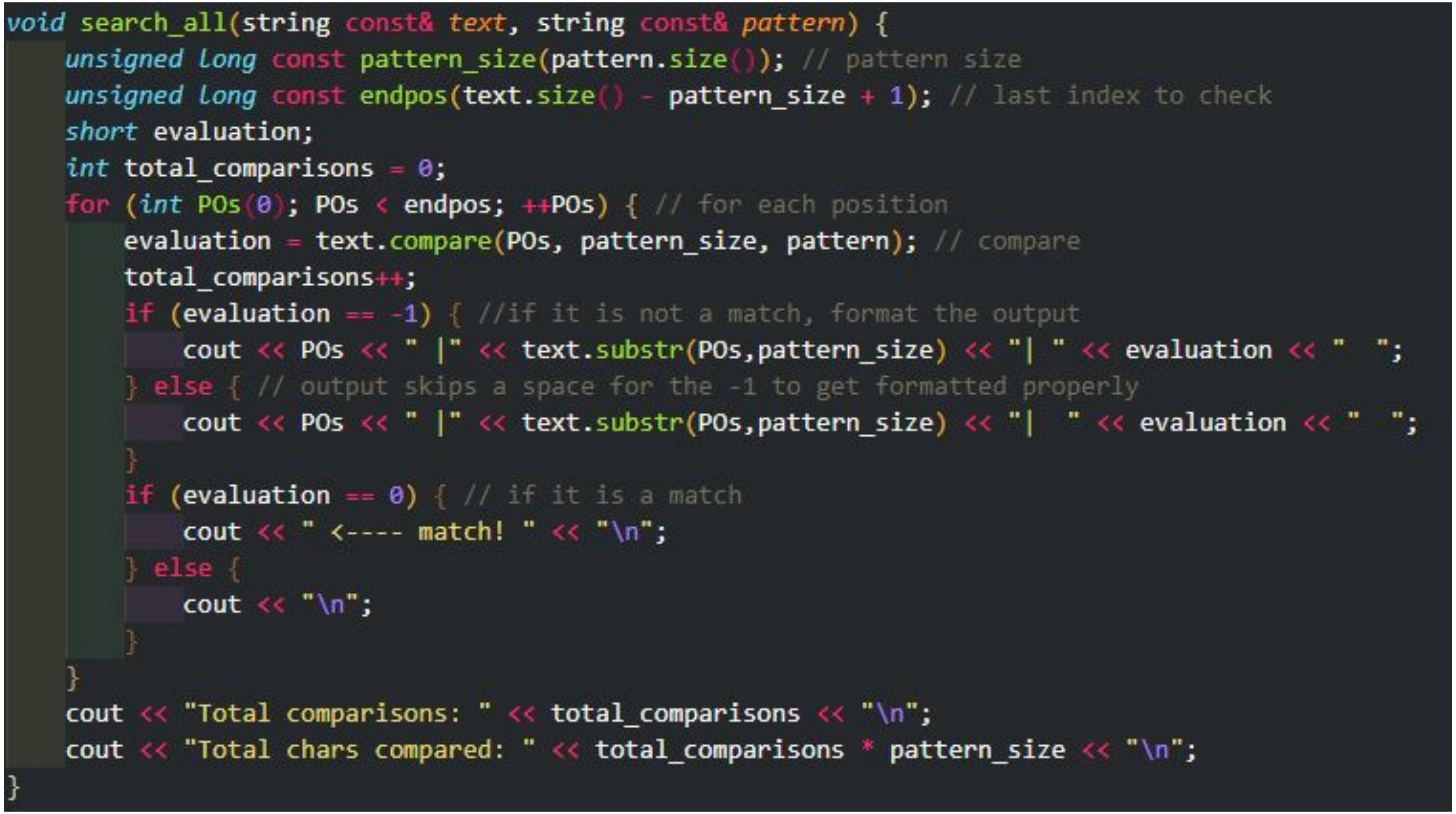
Act 1.3.1 - Algoritmos de búsqueda

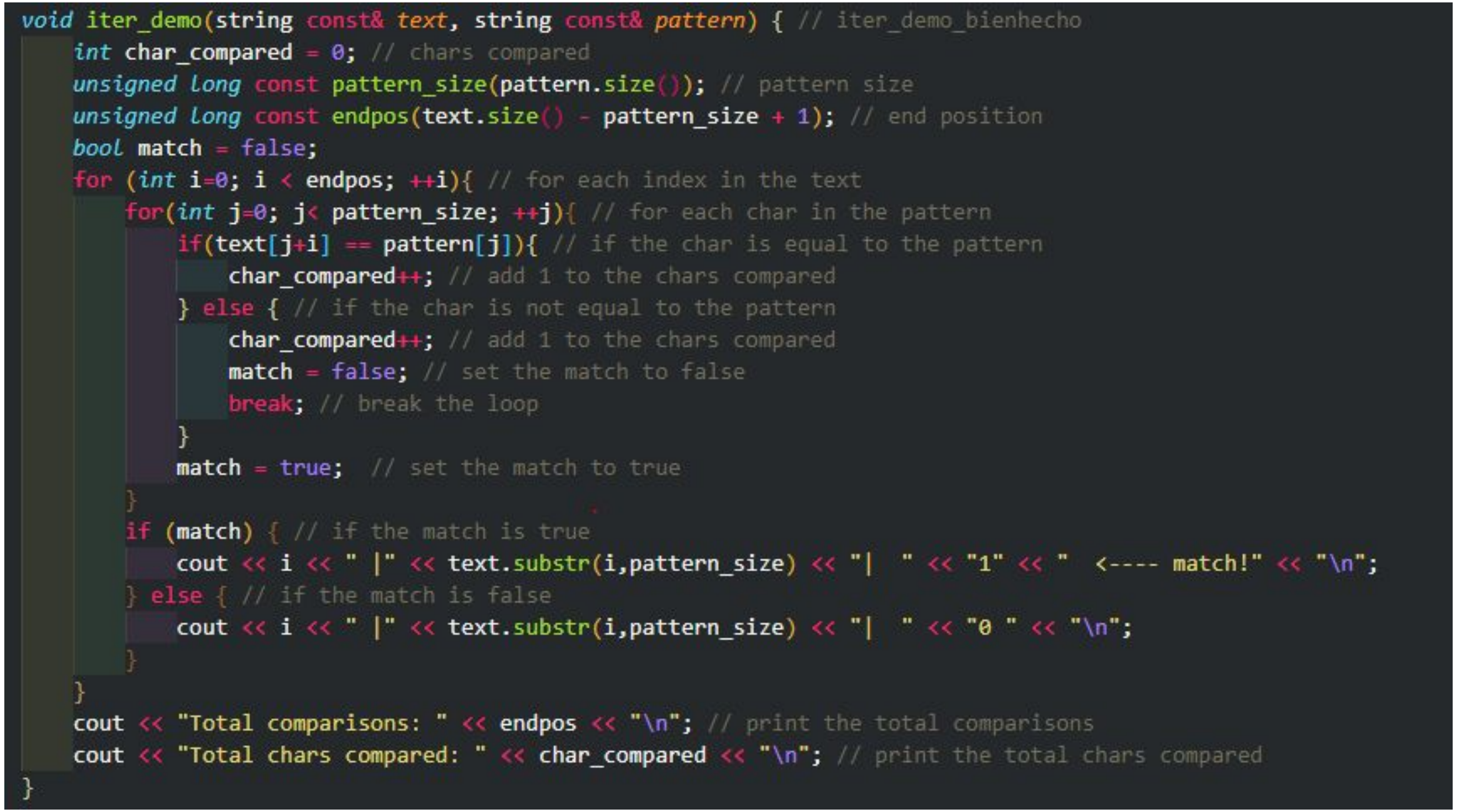
Por: Antonio Noguerón Bárcenas y Armando Arredondo Valle

Algoritmo 1:



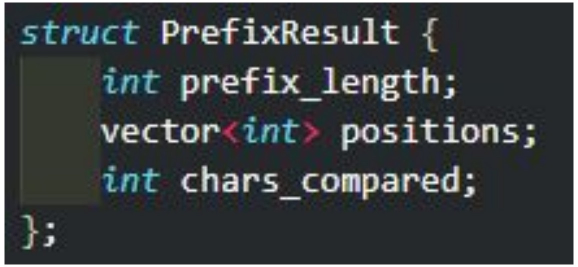
```
void search_all(string const& text, string const& pattern) {
    unsigned long const pattern_size(pattern.size()); // pattern size
    unsigned long const endpos(text.size() - pattern_size + 1); // last index to check
    short evaluation;
    int total_comparisons = 0;
    for (int POs(0); POs < endpos; ++POs) { // for each position
        evaluation = text.compare(POs, pattern_size, pattern); // compare
        total_comparisons++;
        if (evaluation == -1) { //if it is not a match, format the output
            cout << POs << " |" << text.substr(POs,pattern_size) << "| " << evaluation << "  ";
        } else { // output skips a space for the -1 to get formatted properly
            cout << POs << " |" << text.substr(POs,pattern_size) << "|  " << evaluation << "  ";
        }
        if (evaluation == 0) { // if it is a match
            cout << " <---- match! " << "\n";
        } else {
            cout << "\n";
        }
    }
    cout << "Total comparisons: " << total_comparisons << "\n";
    cout << "Total chars compared: " << total_comparisons * pattern_size << "\n";
}
```

Algoritmo 2

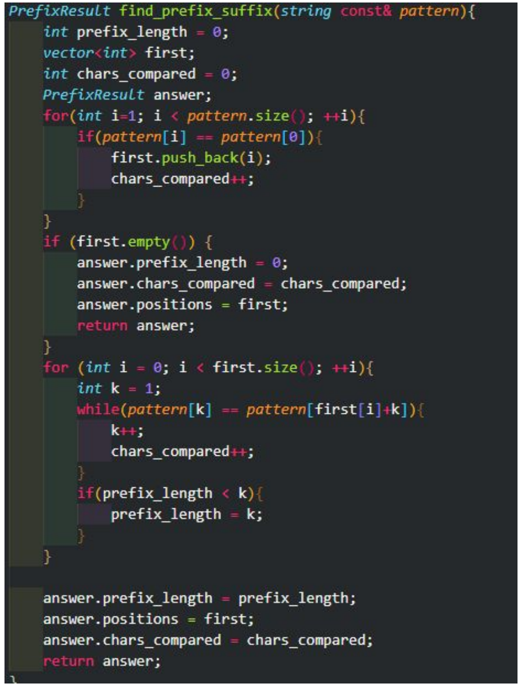


```cpp
void iter_demo(string const& text, string const& pattern) { // iter_demo_bienhecho
    int char_compared = 0; // chars compared
    unsigned long const pattern_size(pattern.size()); // pattern size
    unsigned long const endpos(text.size() - pattern_size + 1); // end position
    bool match = false;
    for (int i=0; i < endpos; ++i){ // for each index in the text
        for(int j=0; j< pattern_size; ++j){ // for each char in the pattern
            if(text[j+i] == pattern[j]){ // if the char is equal to the pattern
                char_compared++; // add 1 to the chars compared
            } else { // if the char is not equal to the pattern
                char_compared++; // add 1 to the chars compared
                match = false; // set the match to false
                break; // break the loop
            }
            match = true;  // set the match to true
        }
        if (match) { // if the match is true
            cout << i << " |" << text.substr(i,pattern_size) << "|  " << "1" << "  <---- match!" << "\n";
        } else { // if the match is false
            cout << i << " |" << text.substr(i,pattern_size) << "|  " << "0 " << "\n";
        }
    }
    cout << "Total comparisons: " << endpos << "\n"; // print the total comparisons
    cout << "Total chars compared: " << char_compared << "\n"; // print the total chars compared
}
```

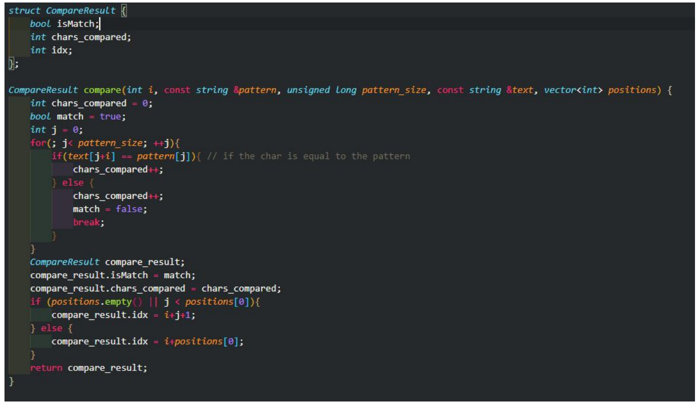
Algoritmo 3:



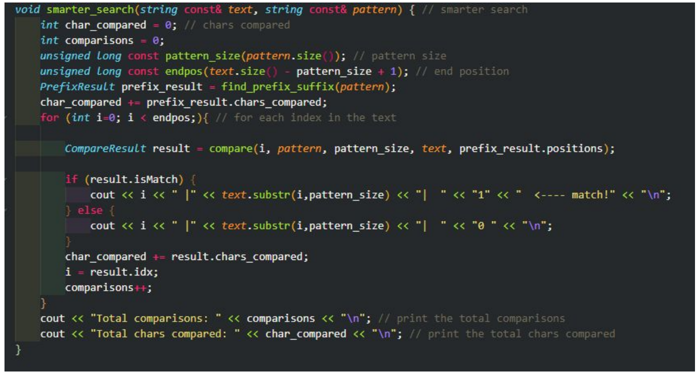
```cpp
struct PrefixResult {
    int prefix_length;
    vector<int> positions;
    int chars_compared;
};
```



```cpp
PrefixResult find_prefix_suffix(string const& pattern){
    int prefix_length = 0;
    vector<int> first;
    int chars_compared = 0;
    PrefixResult answer;
    for(int i=1; i < pattern.size(); ++i){
        if(pattern[i] == pattern[0]){
            first.push_back(i);
            chars_compared++;
        }
    }
    if (first.empty()) {
        answer.prefix_length = 0;
        answer.chars_compared = chars_compared;
        answer.positions = first;
        return answer;
    }
    for (int i = 0; i < first.size(); ++i){
        int k = 1;
        while(pattern[k] == pattern[first[i]+k]){
            k++;
            chars_compared++;
        }
        if(prefix_length < k){
            prefix_length = k;
        }
    }

    answer.prefix_length = prefix_length;
    answer.positions = first;
    answer.chars_compared = chars_compared;
    return answer;
```

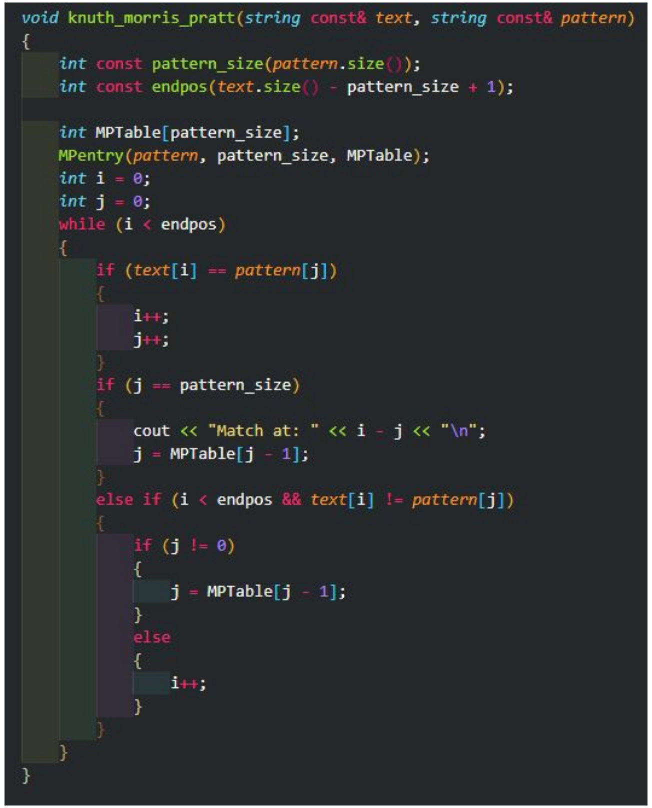


```cpp
struct CompareResult {
    bool isMatch;
    int chars_compared;
    int idx;
};

CompareResult compare(int i, const string &pattern, unsigned long pattern_size, const string &text, vector<int> positions) {
    int chars_compared = 0;
    bool match = true;
    int j = 0;
    for(; j< pattern_size; ++j){
        if(text[j+i] == pattern[j]){ // if the char is equal to the pattern
            chars_compared++;
        } else {
            chars_compared++;
            match = false;
            break;
        }
    }
    CompareResult compare_result;
    compare_result.isMatch = match;
    compare_result.chars_compared = chars_compared;
    if (positions.empty() || j < positions[0]){
        compare_result.idx = i+j+1;
    } else {
        compare_result.idx = i+positions[0];
    }
    return compare_result;
}
```



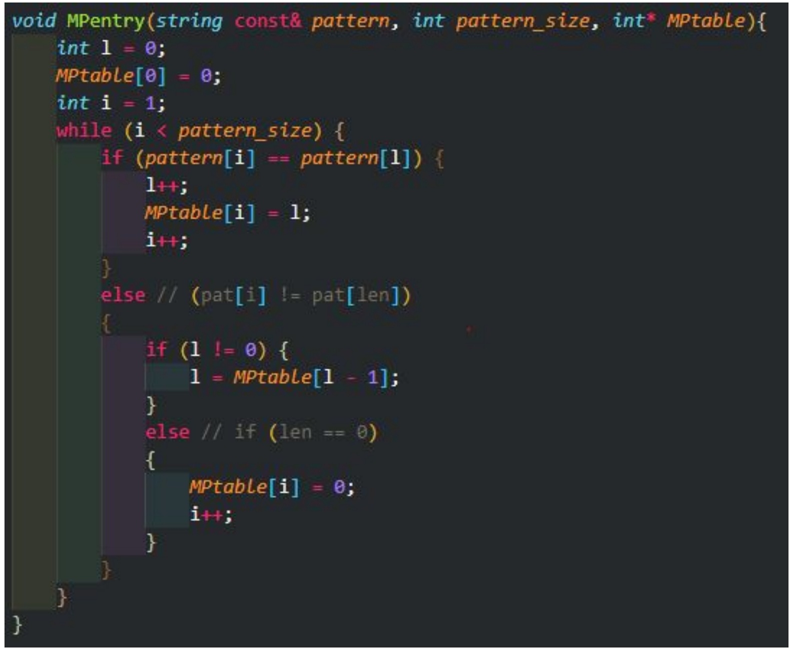
```cpp
void smarter_search(string const& text, string const& pattern) { // smarter search
    int char_compared = 0; // chars compared
    int comparisons = 0;
    unsigned long const pattern_size(pattern.size()); // pattern size
    unsigned long const endpos(text.size() - pattern_size + 1); // end position
    PrefixResult prefix_result = find_prefix_suffix(pattern);
    char_compared += prefix_result.chars_compared;
    for (int i=0; i < endpos;){ // for each index in the text

        CompareResult result = compare(i, pattern, pattern_size, text, prefix_result.positions);

        if (result.isMatch) {
            cout << i << " |" << text.substr(i,pattern_size) << "|  " << "1" << "  <---- match!" << "\n";
        } else {
            cout << i << " |" << text.substr(i,pattern_size) << "|  " << "0 " << "\n";
        }
        char_compared += result.chars_compared;
        i = result.idx;
        comparisons++;
    }
    cout << "Total comparisons: " << comparisons << "\n"; // print the total comparisons
    cout << "Total chars compared: " << char_compared << "\n"; // print the total chars compared
}
```

Algoritmo 4:

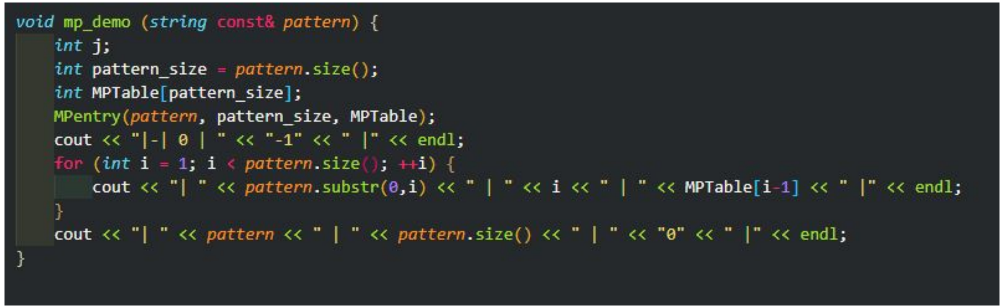


```
void knuth_morris_pratt(string const& text, string const& pattern)
{
    int const pattern_size(pattern.size());
    int const endpos(text.size() - pattern_size + 1);

    int MPTable[pattern_size];
    MPentry(pattern, pattern_size, MPTable);
    int i = 0;
    int j = 0;
    while (i < endpos)
    {
        if (text[i] == pattern[j])
        {
            i++;
            j++;
        }
        if (j == pattern_size)
        {
            cout << "Match at: " << i - j << "\n";
            j = MPTable[j - 1];
        }
        else if (i < endpos && text[i] != pattern[j])
        {
            if (j != 0)
            {
                j = MPTable[j - 1];
            }
            else
            {
                i++;
            }
        }
    }
}
```

Algoritmo 5:



```cpp
void MPentry(string const& pattern, int pattern_size, int* MPtable){
    int l = 0;
    MPtable[0] = 0;
    int i = 1;
    while (i < pattern_size) {
        if (pattern[i] == pattern[l]) {
            l++;
            MPtable[i] = l;
            i++;
        }
        else // (pat[i] != pat[len])
        {
            if (l != 0) {
                l = MPtable[l - 1];
            }
            else // if (len == 0)
            {
                MPtable[i] = 0;
                i++;
            }
        }
    }
}
```



```cpp
void mp_demo (string const& pattern) {
    int j;
    int pattern_size = pattern.size();
    int MPTable[pattern_size];
    MPentry(pattern, pattern_size, MPTable);
    cout << "|-| 0 | " << "-1" << " |" << endl;
    for (int i = 1; i < pattern.size(); ++i) {
        cout << "| " << pattern.substr(0,i) << " | " << i << " | " << MPTable[i-1] << " |" << endl;
    }
    cout << "| " << pattern << " | " << pattern.size() << " | " << "0" << " |" << endl;
}
```

Preguntas Algoritmo 1:

Question 1:

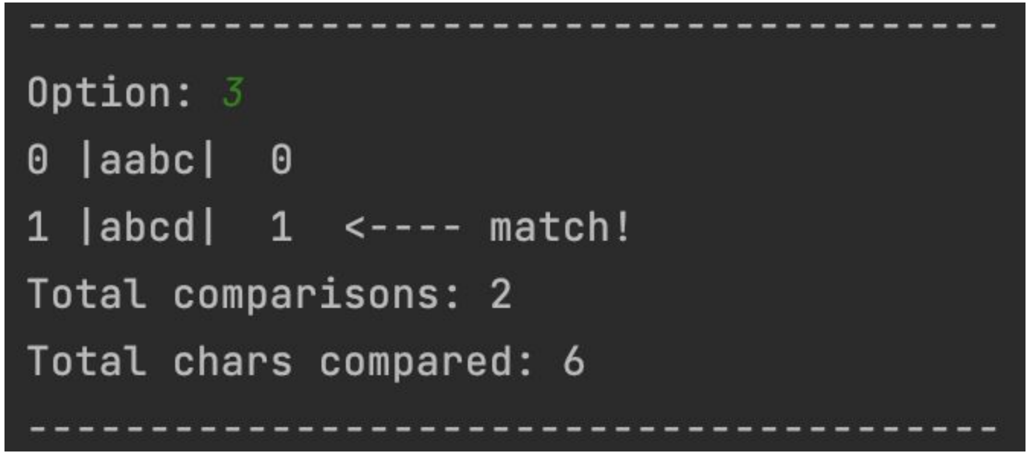
Si el texto es de longitud *n,* y el string a comparar es de longitud *m*, ¿Cuántas llamadas se le harán al método de comparación con la llamada de la función *search_all(text, pat)*?

- Se llamará n-m+1 veces.

Preguntas Algoritmo 1:

Question 2:

¿Cuántas comparaciones de caracteres se harán por el search_all("aabcd","abcd")? R = 6

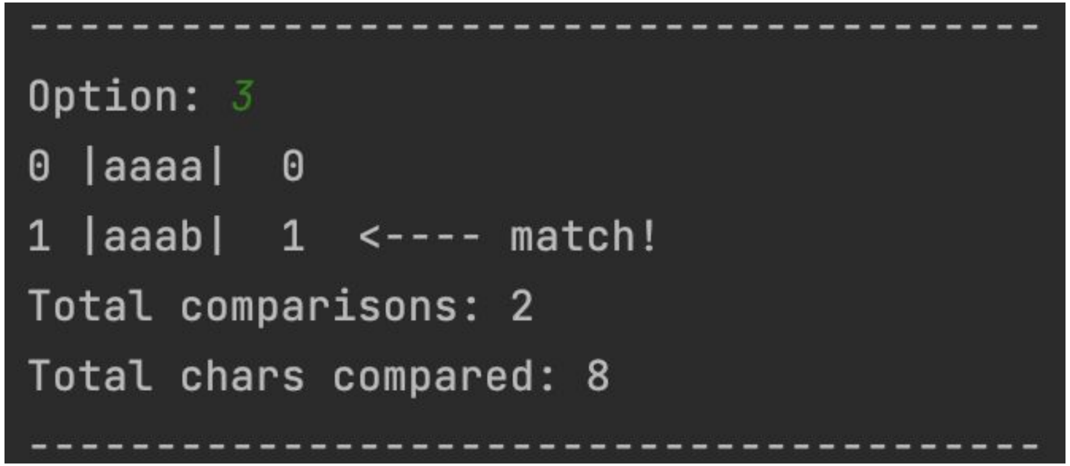


```
----------------------------------------
Option: 3
0 |aabc|  0
1 |abcd|  1  <---- match!
Total comparisons: 2
Total chars compared: 6
----------------------------------------
```

Preguntas Algoritmo 1:

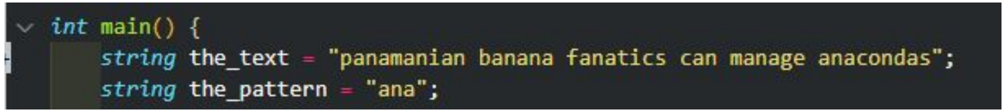
Question 3:

¿Cuántas comparaciones de caracteres se harán por el search_all("aaaab","aaab")? R = 8

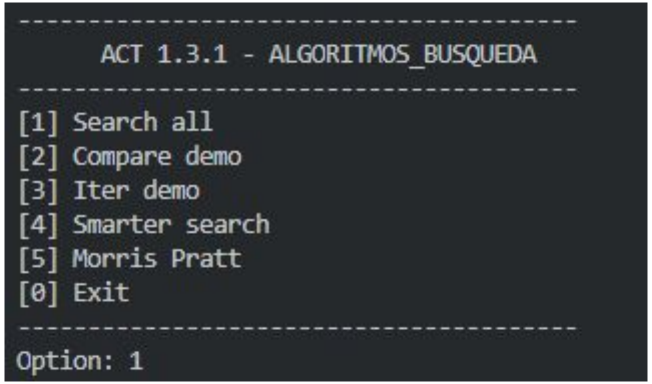


```
----------------------------------------

Option: 3
0 |aaaa|  0
1 |aaab|  1  <---- match!
Total comparisons: 2
Total chars compared: 8

----------------------------------------
```

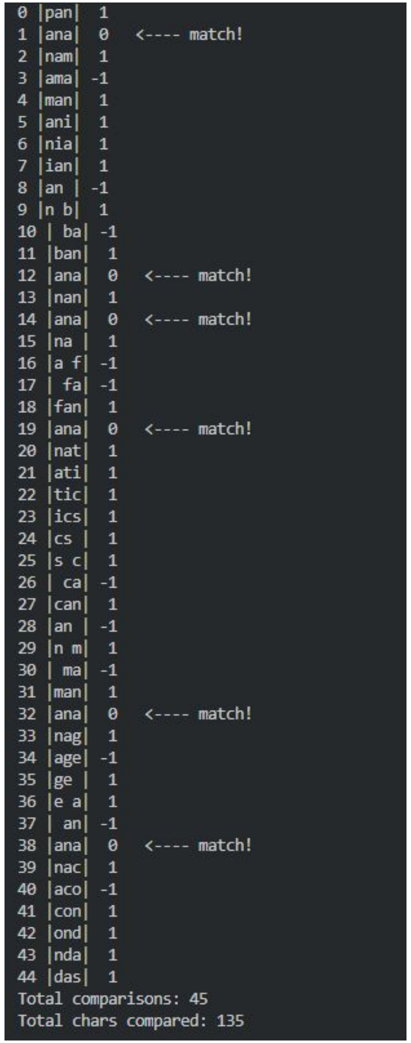
CASO DE PRUEBA ALGORITMO 1:



```cpp
int main() {
    string the_text = "panamanian banana fanatics can manage anacondas";
    string the_pattern = "ana";
```

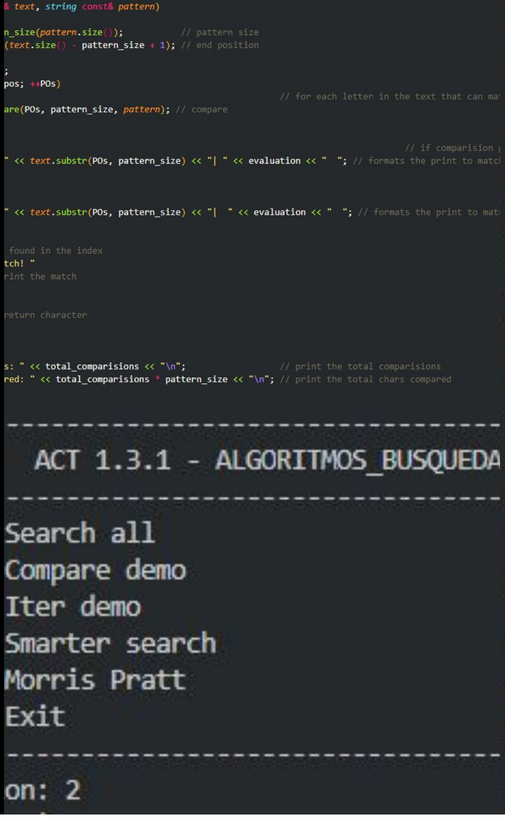


```
----------------------------------------
      ACT 1.3.1 - ALGORITMOS_BUSQUEDA
----------------------------------------
[1] Search all
[2] Compare demo
[3] Iter demo
[4] Smarter search
[5] Morris Pratt
[0] Exit
----------------------------------------
Option: 1
```



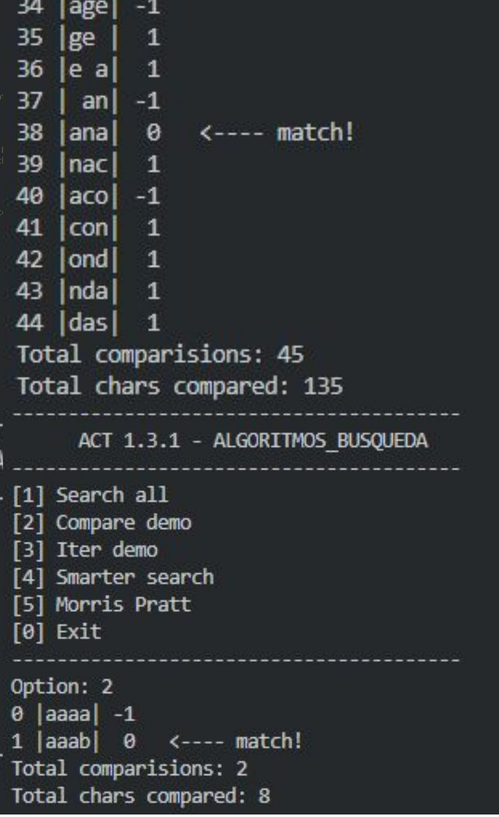
```
0 |pan|  1
1 |ana|  0   <---- match!
2 |nam|  1
3 |ama| -1
4 |man|  1
5 |ani|  1
6 |nia|  1
7 |ian|  1
8 |an | -1
9 |n b|  1
10 | ba| -1
11 |ban|  1
12 |ana|  0   <---- match!
13 |nan|  1
14 |ana|  0   <---- match!
15 |na |  1
16 |a f| -1
17 | fa| -1
18 |fan|  1
19 |ana|  0   <---- match!
20 |nat|  1
21 |ati|  1
22 |tic|  1
23 |ics|  1
24 |cs |  1
25 |s c|  1
26 | ca| -1
27 |can|  1
28 |an | -1
29 |n m|  1
30 | ma| -1
31 |man|  1
32 |ana|  0   <---- match!
33 |nag|  1
34 |age| -1
35 |ge |  1
36 |e a|  1
37 | an| -1
38 |ana|  0   <---- match!
39 |nac|  1
40 |aco| -1
41 |con|  1
42 |ond|  1
43 |nda|  1
44 |das|  1
Total comparisons: 45
Total chars compared: 135
```

CASOS DE PRUEBA ALGORITMO 2:



```
& text, string const& pattern)

n_size(pattern.size());          // pattern size
(text.size() - pattern_size + 1); // end position

;
pos; ++POs)
                                                   // for each letter in the text that can ma
are(POs, pattern_size, pattern); // compare

                                                                  // if comparision
" << text.substr(POs, pattern_size) << "| " << evaluation << "  "; // formats the print to matc

" << text.substr(POs, pattern_size) << "|  " << evaluation << "  "; // formats the print to mat

 found in the index
tch! "
rint the match

return character

s: " << total_comparisions << "\n";                   // print the total comparisions
red: " << total_comparisions * pattern_size << "\n"; // print the total chars compared
```

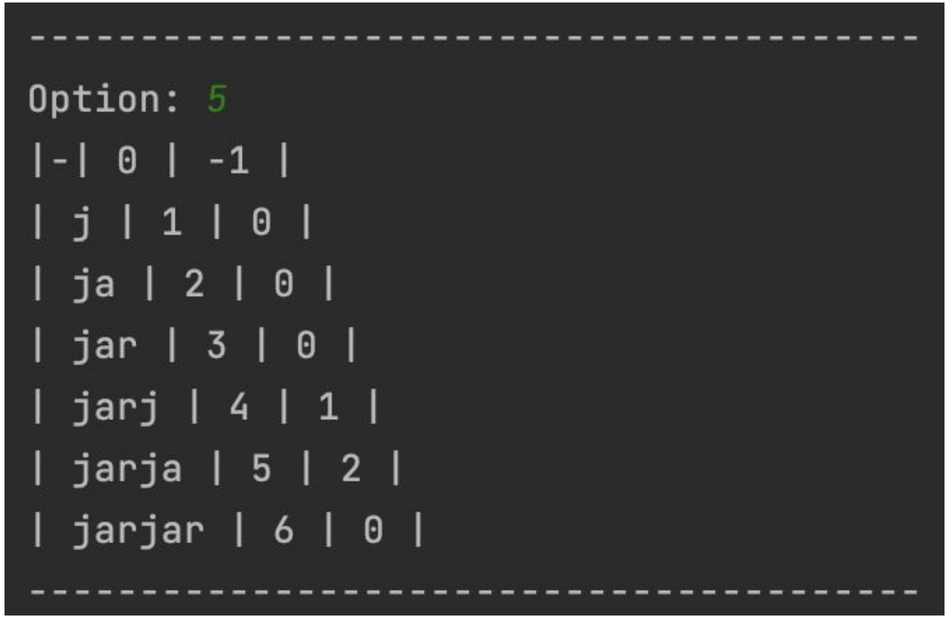
```
-----------------------------------
 ACT 1.3.1 - ALGORITMOS_BUSQUEDA
-----------------------------------
Search all
Compare demo
Iter demo
Smarter search
Morris Pratt
Exit
-----------------------------------
on: 2
```



```
34 |age| -1
35 |ge |  1
36 |e a|  1
37 | an| -1
38 |ana|  0   <---- match!
39 |nac|  1
40 |aco| -1
41 |con|  1
42 |ond|  1
43 |nda|  1
44 |das|  1
Total comparisions: 45
Total chars compared: 135
---------------------------------------
     ACT 1.3.1 - ALGORITMOS_BUSQUEDA
---------------------------------------
[1] Search all
[2] Compare demo
[3] Iter demo
[4] Smarter search
[5] Morris Pratt
[0] Exit
---------------------------------------
Option: 2
0 |aaaa| -1
1 |aaab|  0   <---- match!
Total comparisions: 2
Total chars compared: 8
```

Morris-Pratt Algorithm

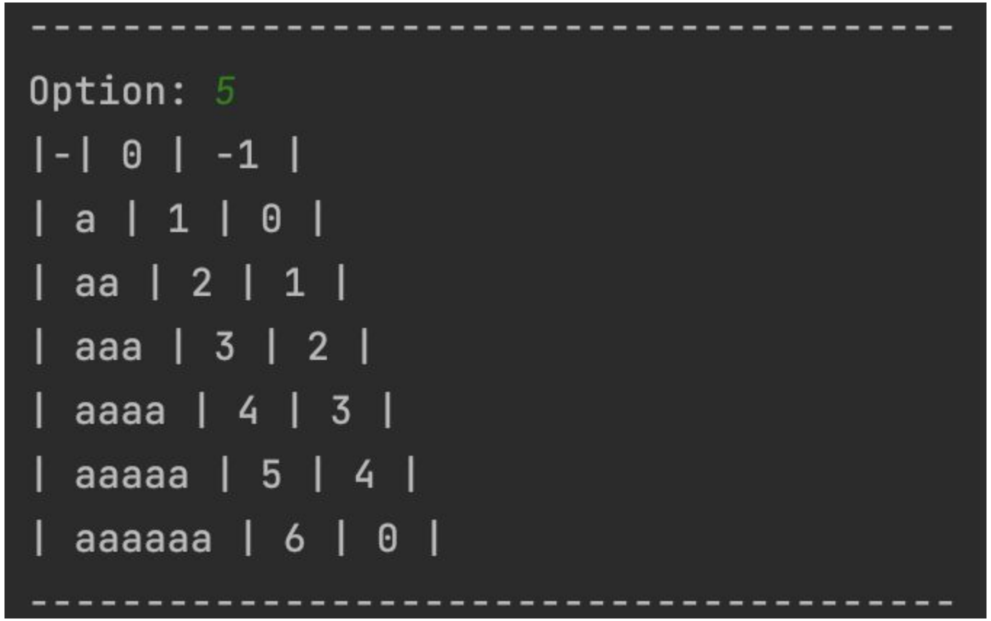
Question 4: Jar Jar table.



```
----------------------------------------

Option: 5
|-| 0 | -1 |
| j | 1 | 0 |
| ja | 2 | 0 |
| jar | 3 | 0 |
| jarj | 4 | 1 |
| jarja | 5 | 2 |
| jarjar | 6 | 0 |
----------------------------------------
```

Morris-Pratt Algorithm

Question 5: “aaaaaa” table.



```
----------------------------------------

Option: 5
|-| 0 | -1 |
| a | 1 | 0 |
| aa | 2 | 1 |
| aaa | 3 | 2 |
| aaaa | 4 | 3 |
| aaaaa | 5 | 4 |
| aaaaaa | 6 | 0 |

----------------------------------------
```