

PROYECTO BUSCADOR AED1
Ángel Ruiz Fernandez & Carla Ramos García
12:51 Nov 27, 2024

PROYECTO BUSCADOR AED1: 6 files

PAGE

diccionario.hpp 72L 1929B	3
interprete.hpp 14L 261B	5
diccionario.cpp 219L 5664B	6
interprete.cpp 203L 6026B	10
main.cpp 23L 444B	14
Makefile 35L 640B	15

```
1 #pragma once
2
3 #include <string>
4 #include <vector>
5 #include <set>
6 #include <list>
7 #include <map>
8 #include <memory>
9
10 class Pagina {
11     private:
12         std::wstring url, titulo;
13         int relevancia;
14         std::vector<std::wstring> contenido;
15
16     public:
17         Pagina(const std::wstring& _url, const std::wstring& _titulo, int _rel,
18             const std::vector<std::wstring>& _cont);
19         const std::wstring& getUrl() const;
20         const std::wstring& getTitulo() const;
21         void setTitulo(const std::wstring& titulo);
22         int getRelevancia() const;
23         void setRelevancia(int rel);
24         int getContSize() const;
25         const std::vector<std::wstring>& getContenido() const;
26         void setContenido(const std::vector<std::wstring>& palabras);
27 };
28
29 //bool comparar_pagref(const std::list<Pagina>::iterator& l,
30 //    const std::list<Pagina>::iterator& r);
31
32 class PagListIt : public std::list<Pagina>::iterator {
33     public:
34         PagListIt(std::list<Pagina>::iterator& it);
35         bool operator<(const PagListIt& right) const;
36 };
37
38 struct nodo_trie_t {
39     std::set<PagListIt> paginas;
40     std::map<wchar_t, nodo_trie_t> hijos;
41 };
42
43 class Arbol {
44     private:
45         std::map<wchar_t, nodo_trie_t> raiz;
46
47     public:
48         void insertar(const std::wstring& palabra,
49             PagListIt pagineref);
50         std::set<PagListIt> buscar(const std::wstring& palabra);
51         std::vector<std::pair<std::wstring, int>>
52             palabrasPrefijo(const std::wstring& palabra);
53 };
54
55 class Diccionario {
56     private:
57         static constexpr int N = 20000;
58         std::list<Pagina> tabla[N];
59         Arbol arbol;
60         size_t hash(const std::wstring& key);
61         int size = 0;
62 }
```

```
63     public:
64     void insertar(const Pagina& p);
65     std::vector<Pagina> consultar(const std::wstring& url);
66     std::set<PagListIt> buscarPalabra(const std::wstring& palabra);
67     void pseudoDestructor();
68     size_t getTam();
69     std::vector<std::pair<std::wstring, int>>
70         palabrasPrefijo(const std::wstring& palabra);
71 };
72
```

```
1 #pragma once
2
3 #include <string>
4 #include <vector>
5
6 #include "diccionario.hpp"
7
8 auto normalizarc(wchar_t c);
9 auto normalizarstr(const std::wstring& s);
10 Pagina leerpagina();
11 std::vector<std::wstring> leerpalabras();
12
13 int interpretar(char cmd, Diccionario& dic);
14
```

```
1 #include "diccionario.hpp"
2
3 #include <algorithm>
4 #include <iostream>
5 #include <memory>
6
7 // Pagina
8 Pagina::Pagina(const std::wstring& _url, const std::wstring& _titulo, int _rel,
9               const std::vector<std::wstring>& _cont)
10 {
11     url = _url;
12     titulo = _titulo;
13     relevancia = _rel;
14     contenido = _cont;
15 }
16
17 const std::wstring& Pagina::getUrl() const {
18     return url;
19 }
20
21 const std::wstring& Pagina::getTitulo() const {
22     return titulo;
23 }
24
25 void Pagina::setTitulo(const std::wstring& titulo) {
26     this->titulo = titulo;
27 }
28
29 int Pagina::getRelevancia() const {
30     return relevancia;
31 }
32
33 void Pagina::setRelevancia(int rel) {
34     this->relevancia = rel;
35 }
36
37 int Pagina::getContSize() const {
38     return contenido.size();
39 }
40
41 const std::vector<std::wstring>& Pagina::getContenido() const {
42     return contenido;
43 }
44
45 void Pagina::setContenido(const std::vector<std::wstring>& palabras) {
46     contenido = palabras;
47 }
48
49
50 // Clase referencia a pagina
51
52 PagListIt::PagListIt(std::list<Pagina>::iterator& it)
53     : std::list<Pagina>::iterator(it)
54 {
55 }
56
57
58 bool PagListIt::operator<(const PagListIt& r) const {
59     if ((*this)->getRelevancia() != r->getRelevancia())
60         return (*this)->getRelevancia() > r->getRelevancia();
61     else return (*this)->getUrl() < r->getUrl();
62 }
```

```
63
64 // Diccionario
65
66 size_t Diccionario::hash(const std::wstring& key) {
67     size_t t = 5381;
68     for (auto c : key)
69         t = ((t << 5) + t) + c;
70     return t % N;
71 }
72
73 void insertar_palabras(const std::vector<std::wstring>& palabras,
74     const std::list<Pagina>::iterator& pagref, Arbol& arbol)
75 {
76     for (const std::wstring& p : palabras) {
77         arbol.insertar(p, (const PagListIt&)pagref);
78     }
79 }
80
81 void Diccionario::insertar(const Pagina& np) {
82     auto nhash = hash(np.getUrl());
83
84     std::list<Pagina>::iterator it;
85
86     auto& vec = tabla[nhash];
87     for (it = vec.begin(); it != vec.end(); it++) {
88         if (it->getUrl() == np.getUrl()) {
89             auto& p = *it;
90             p.setTitulo(np.getTitulo());
91             p.setRelevancia(np.getRelevancia());
92             p.setContenido(np.getContenido());
93             insertar_palabras(np.getContenido(), it, arbol);
94             return;
95         }
96     }
97
98     it = tabla[nhash].insert(tabla[nhash].end(), Pagina(np));
99     insertar_palabras(np.getContenido(), it, arbol);
100     size++;
101 }
102
103 std::vector<Pagina> Diccionario::consultar(const std::wstring& url) {
104     std::vector<Pagina> resultado;
105     for (const auto& p : tabla[hash(url)])
106         if (p.getUrl() == url)
107             resultado.push_back(p);
108     return resultado;
109 }
110
111 std::set<PagListIt>
112 Diccionario::buscarPalabra(const std::wstring& palabra) {
113     return arbol.buscar(palabra);
114 }
115
116 void Diccionario::pseudoDestructor() {
117     for (int i = 0; i < N; i++)
118         tabla[i].clear();
119 }
120
121 size_t Diccionario::getTam() {
122     return size;
123 }
124
```

```
125 std::vector<std::pair<std::wstring, int>>
126 Diccionario::palabrasPrefijo(const std::wstring& palabra) {
127     return arbol.palabrasPrefijo(palabra);
128 }
129
130 // Arbol
131
132 bool comparar_pagref(const PagListIt& l,
133     const PagListIt& r)
134 {
135     if (l->getRelevancia() != r->getRelevancia())
136         return l->getRelevancia() > r->getRelevancia();
137     else return l->getUrl() < r->getUrl();
138 }
139
140 void Arbol::insertar(const std::wstring& palabra,
141     PagListIt pagineref)
142 {
143     auto subarbol = &raiz; // puntero porque referencia rebindeable
144     std::map<wchar_t, nodo_trie_t>::iterator it;
145
146     for (wchar_t c : palabra) {
147         auto nuevonodo = nodo_trie_t {};
148         // solo inserta si c no existe
149         it = subarbol->insert({c, nuevonodo}).first;
150         subarbol = &it->second.hijos;
151     }
152
153     if (std::find(it->second.paginas.begin(),
154         it->second.paginas.end(), pagineref) == it->second.paginas.end())
155     {
156         it->second.paginas.insert(pagineref);
157     }
158 }
159
160 // no const ref return porque si no se encuentra resultado,
161 // retorna nuevo vector vacio
162 std::set<PagListIt>
163 Arbol::buscar(const std::wstring& palabra) {
164     auto subarbol = &raiz;
165     std::map<wchar_t, nodo_trie_t>::iterator it;
166
167     for (wchar_t c : palabra) {
168         auto nuevonodo = nodo_trie_t();
169         it = subarbol->find(c);
170         if (it == subarbol->end())
171             // retornar vector vacio si no hay resultados
172             return std::set<PagListIt>();
173
174         subarbol = &it->second.hijos;
175     }
176
177     return it->second.paginas;
178 }
179
180 void palabrasPrefijoRecurzar(const std::map<wchar_t, nodo_trie_t>& a,
181     std::wstring p, std::vector<std::pair<std::wstring, int>>& palabras)
182 {
183     for (auto n : a) {
184         if (n.second.paginas.size() > 0)
185             palabras.push_back({p + n.first, n.second.paginas.size()});
186         palabrasPrefijoRecurzar(n.second.hijos, p + n.first, palabras);
187     }
188 }
```



```
187     }
188 }
189
190 bool comparadorPalabrasPrefijo(const std::pair<std::wstring, int>& l,
191     const std::pair<std::wstring, int>& r)
192 {
193     if (l.second != r.second) return l.second > r.second;
194     else return l.first < r.first;
195 }
196
197 std::vector<std::pair<std::wstring, int>>
198 Arbol::palabrasPrefijo(const std::wstring& prefijo) {
199     auto subarbol = &raiz;
200     std::map<wchar_t, nodo_trie_t>::iterator it;
201
202     for (wchar_t c : prefijo) {
203         auto nuevonodo = nodo_trie_t();
204         it = subarbol->find(c);
205         if (it == subarbol->end())
206             return std::vector<std::pair<std::wstring, int>>();
207
208         subarbol = &it->second.hijos;
209     }
210
211     std::vector<std::pair<std::wstring, int>> palabras;
212
213     if (it->second.paginas.size() > 0)
214         palabras.push_back({prefijo, it->second.paginas.size()});
215     palabrasPrefijoRecurzar(*subarbol, prefijo, palabras);
216     std::sort(palabras.begin(), palabras.end(), comparadorPalabrasPrefijo);
217     return palabras;
218 }
219
```

```
1 #include "interprete.hpp"
2
3 #include <iostream>
4 #include <limits>
5 #include <sstream>
6 #include <algorithm>
7 #include <iterator>
8
9 #include "diccionario.hpp"
10
11 auto normalizarc(wchar_t c) {
12     c = std::tolower(c); // convierte solo ASCII-7
13
14     if (c == L'Ñ') c = L'ñ';
15
16     if (c == L'Á') c = 'a';
17     if (c == L'É') c = 'e';
18     if (c == L'Í') c = 'i';
19     if (c == L'Ó') c = 'o';
20     if (c == L'Ú') c = 'u';
21     if (c == L'Ü') c = 'u';
22
23     if (c == L'á') c = 'a';
24     if (c == L'é') c = 'e';
25     if (c == L'í') c = 'i';
26     if (c == L'ó') c = 'o';
27     if (c == L'ú') c = 'u';
28     if (c == L'ü') c = 'u';
29     return c;
30 }
31
32 auto normalizarstr(const std::wstring& s) {
33     std::wstring o;
34     for (auto c : s)
35         o += normalizarc(c);
36     return o;
37 }
38
39 Pagina leerpagina() {
40     int rel;
41     std::wstring url, titulo;
42     std::vector<std::wstring> contenido;
43
44     std::wcin >> rel;
45     std::wcin >> url;
46     std::wcin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
47     std::getline(std::wcin, titulo);
48
49     std::wstring in;
50     while ((std::wcin >> in) && (normalizarstr(in) != L"findepagina"))
51         contenido.push_back(normalizarstr(in));
52
53     return Pagina(url, titulo, rel, contenido);
54 }
55
56 std::vector<std::wstring> leerpalabras() {
57     std::vector<std::wstring> palabras;
58     std::wstring palabrasstr;
59     std::getline(std::wcin, palabrasstr);
60
61     palabrasstr = normalizarstr(palabrasstr);
62 }
```

```
63     auto ss = std::wstringstream(palabrasstr);
64     std::wstring palabra;
65
66     while (ss >> palabra)
67         palabras.push_back(palabra);
68
69     return palabras;
70 }
71
72 int interpretar(char cmd, Diccionario& dic) {
73     switch (cmd) {
74         case 'i': {
75             Pagina np = leerpagina();
76             dic.insertar(np);
77             std::wcout << dic.getTam() << ". " << np.getUrl() << ", "
78                 << np.getTitulo() << ", Rel. " << np.getRelevancia()
79                 << std::endl << np.getContSize() << " palabras" << std::endl;
80         } break;
81         case 'u': {
82             std::wstring url;
83             std::wcin >> url;
84
85             auto p = dic.consultar(url);
86
87             std::wcout << "u " << url << std::endl;
88             for (size_t i = 0; i < p.size(); i++)
89                 std::wcout << i + 1 << ". " << p[i].getUrl() << ", "
90                     << p[i].getTitulo() << ", Rel. " << p[i].getRelevancia()
91                     << std::endl;
92             std::wcout << "Total: " << p.size() << " resultados" << std::endl;
93
94         } break;
95         case 'b': {
96             std::wstring palabra;
97             std::wcin >> palabra;
98             palabra = normalizarstr(palabra);
99
100             auto paginas = dic.buscarPalabra(palabra);
101
102             std::wcout << "b " << palabra << std::endl;
103             int i = 1;
104             for (auto& p : paginas) {
105                 std::wcout << i << ". " << p->getUrl() << ", "
106                     << p->getTitulo() << ", Rel. "
107                     << p->getRelevancia() << std::endl;
108                 i++;
109             }
110             std::wcout << "Total: " << paginas.size() << " resultados"
111                 << std::endl;
112         } break;
113         case 'a': {
114             std::vector<std::wstring> palabras = leerpalabras();
115
116             std::wcout << "a";
117             for (auto p : palabras)
118                 std::wcout << " " << p;
119             std::wcout << std::endl;
120
121             std::set<PagListIt> interseccion;
122
123             if (!palabras.empty()) {
124                 interseccion = dic.buscarPalabra(palabras[0]);
```

```
125         palabras.erase(palabras.begin());
126     }
127
128     for (auto& p : palabras) {
129         std::set<PagListIt> paginas = dic.buscarPalabra(p), nuevo;
130
131         std::set_intersection(interseccion.begin(), interseccion.end(),
132                               paginas.begin(), paginas.end(),
133                               std::inserter(nuevo, nuevo.begin()));
134
135         interseccion = nuevo;
136     }
137
138     int i = 1;
139     for (auto& p : interseccion) {
140         std::wcout << i << ". " << p->getUrl() << ", "
141                     << p->getTitulo() << ", Rel. "
142                     << p->getRelevancia() << std::endl;
143         i++;
144     }
145     std::wcout << "Total: " << interseccion.size()
146               << " resultados" << std::endl;
147 } break;
148 case 'o': {
149     std::vector<std::wstring> palabras = leerpalabras();
150
151     std::wcout << "o";
152     for (auto p : palabras)
153         std::wcout << " " << p;
154     std::wcout << std::endl;
155
156     std::set<PagListIt> unionp;
157
158     for (auto& p : palabras) {
159         std::set<PagListIt> paginas = dic.buscarPalabra(p), nuevo;
160
161         std::set_union(unionp.begin(), unionp.end(),
162                        paginas.begin(), paginas.end(),
163                        std::inserter(unionp, unionp.begin()));
164     }
165
166     int i = 1;
167     for (auto& p : unionp) {
168         std::wcout << i << ". " << p->getUrl() << ", "
169                     << p->getTitulo() << ", Rel. "
170                     << p->getRelevancia() << std::endl;
171         i++;
172     }
173
174     std::wcout << "Total: " << unionp.size() << " resultados" << std::endl;
175 } break;
176 case 'p': {
177     std::wstring prefijo;
178     std::wcin >> prefijo;
179     prefijo = normalizarstr(prefijo);
180
181     auto palabras = dic.palabrasPrefijo(prefijo);
182
183     std::wcout << "p " << prefijo << std::endl;
184
185     int i = 1;
186     for (auto& p : palabras) {
```

```
187         std::wcout << i << ". " << p.first << ", " << p.second
188         << std::endl;
189         i++;
190     }
191
192     std::wcout << "Total: " << palabras.size()
193     << " resultados" << std::endl;
194 } break;
195 case 's': {
196     std::wcout << "Saliendo..." << std::endl;
197     return 0;
198 } break;
199 default: return -1;
200 }
201 return 0;
202 }
203
```

```
1 #include <iostream>
2 #include <vector>
3 #include <string>
4 #include <sstream>
5 #include <limits>
6
7 #include "diccionario.hpp"
8 #include "interprete.hpp"
9
10 int main() {
11     // trabajar con UTF-8 y caracteres anchos
12     std::setlocale(LC_ALL, "es_ES.UTF-8");
13
14     Diccionario dic;
15
16     std::wstring in;
17     while (std::wcin >> in) {
18         if (in.length() != 1) continue;
19         int r = interpretar(in[0], dic);
20         if (r < 0) return 1;
21     }
22 }
23
```

```
1 PROJECT := buscador
2 BINARY := a.out
3 CXX := g++
4 CXXFLAGS := --std=c++17 -Wall -pedantic -g -O0
5 LDFLAGS :=
6
7 SRC := $(wildcard *.cpp)
8 OBJ := $(patsubst %.cpp,%.o,$(SRC))
9
10 all: $(BINARY)
11
12 $(BINARY): $(OBJ)
13     $(CXX) -o $(BINARY) $(OBJ) $(LDFLAGS)
14
15 %.o: %.cpp %.hpp
16     $(CXX) -c $(CXXFLAGS) $<
17
18 .PHONY: test
19 test: $(BINARY)
20     ./$(BINARY) < test_stdin.txt > run_stdout.txt
21     cmp -s test_stdout.txt run_stdout.txt && echo "PASS" || echo "FAIL"
22
23 .PHONY: clean
24 clean:
25     rm $(BINARY) *.o run* *.tar
26
27 .PHONY: tar
28 tar: $(PROJECT).tar
29
30 $(PROJECT).tar: $(SRC) Makefile
31     tar cf $(PROJECT).tar *.cpp *.hpp Makefile
32
33 .PHONY: submit
34 submit: tar
35     python3 submit.py
```