PRÁCTICA T1 y 2

Ángel Ruiz Fernandez, Carla Ramos García

16:57 Dec 05, 2024

PRÁCTICA T1 y 2: 6 files PAGE

diccionario.hpp 72L 1941B 3
interprete.hpp 14L 261B 5
diccionario.cpp 219L 5676B 6
interprete.cpp 196L 6005B 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>::const_iterator {
33 public:
34     PagListIt(std::list<Pagina>::const_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 paginaref);
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
```

```cpp
#include "diccionario.hpp"

#include <algorithm>
#include <iostream>
#include <memory>

// Pagina
Pagina::Pagina(const std::wstring& _url, const std::wstring& _titulo, int _rel,
    const std::vector<std::wstring>& _cont)
{
    url = _url;
    titulo = _titulo;
    relevancia = _rel;
    contenido = _cont;
}

const std::wstring& Pagina::getUrl() const {
    return url;
}

const std::wstring& Pagina::getTitulo() const {
    return titulo;
}

void Pagina::setTitulo(const std::wstring& titulo) {
    this->titulo = titulo;
}

int Pagina::getRelevancia() const {
    return relevancia;
}

void Pagina::setRelevancia(int rel) {
    this->relevancia = rel;
}

int Pagina::getContSize() const {
    return contenido.size();
}

const std::vector<std::wstring>& Pagina::getContenido() const {
    return contenido;
}

void Pagina::setContenido(const std::vector<std::wstring>& palabras) {
    contenido = palabras;
}


// Clase referencia a pagina

PagListIt::PagListIt(std::list<Pagina>::const_iterator& it)
    : std::list<Pagina>::const_iterator(it)
{

}

bool PagListIt::operator<(const PagListIt& r) const {
    if ((*this)->getRelevancia() != r->getRelevancia())
        return (*this)->getRelevancia() > r->getRelevancia();
    else return (*this)->getUrl() < r->getUrl();
}
```

```
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 paginaref)
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(), paginaref) == it->second.paginas.end())
155     {
156         it->second.paginas.insert(paginaref);
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 palabrasPrefijoRecursar(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         palabrasPrefijoRecursar(n.second.hijos, p + n.first, palabras);
```

```
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     palabrasPrefijoRecursar(*subarbol, prefijo, palabras);
216     std::sort(palabras.begin(), palabras.end(), comparadorPalabrasPrefijo);
217     return palabras;
218 }
219
```

```cpp
#include "interprete.hpp"

#include <iostream>
#include <limits>
#include <sstream>
#include <algorithm>
#include <iterator>

#include "diccionario.hpp"

auto normalizarc(wchar_t c) {
    c = std::tolower(c); // convierte solo ASCII-7

    switch (c) {
        case L'Ñ': return L'ñ'; break;
        case L'Á': case L'á': return L'a'; break;
        case L'É': case L'é': return L'e'; break;
        case L'Í': case L'í': return L'i'; break;
        case L'Ó': case L'ó': return L'o'; break;
        case L'Ú': case L'ú': case L'Ü': case L'ü': return L'u'; break;
        default: return c;
    }
}

auto normalizarstr(const std::wstring& s) {
    std::wstring o;
    for (auto c : s)
        o += normalizarc(c);
    return o;
}

Pagina leerpagina() {
    int rel;
    std::wstring url, titulo;
    std::vector<std::wstring> contenido;

    std::wcin >> rel;
    std::wcin >> url;
    std::wcin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    std::getline(std::wcin, titulo);

    std::wstring in;
    while ((std::wcin >> in) && (normalizarstr(in) != L"findepagina"))
        contenido.push_back(normalizarstr(in));

    return Pagina(url, titulo, rel, contenido);
}

std::vector<std::wstring> leerpalabras() {
    std::vector<std::wstring> palabras;
    std::wstring palabrasstr;
    std::getline(std::wcin, palabrasstr);

    palabrasstr = normalizarstr(palabrasstr);

    auto ss = std::wstringstream(palabrasstr);
    std::wstring palabra;

    while (ss >> palabra)
        palabras.push_back(palabra);

    return palabras;
```

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

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

```
187         } break;
188         case 's': {
189             std::wcout << "Saliendo..." << std::endl;
190             return 0;
191         } break;
192         default: return -1;
193     }
194     return 0;
195 }
196
```

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
#include <limits>

#include "diccionario.hpp"
#include "interprete.hpp"

int main() {
    // trabajar con UTF-8 y caracteres anchos
    std::setlocale(LC_ALL, "es_ES.UTF-8");

    Diccionario dic;

    std::wstring in;
    while (std::wcin >> in) {
        if (in.length() != 1) continue;
        int r = interpretar(in[0], dic);
        if (r < 0) return 1;
    }
}

```

```
PROJECT := buscador
BINARY := a.out
CXX := g++
CXXFLAGS := --std=c++17 -Wall -pedantic -g -O0
LDFLAGS :=

SRC := $(wildcard *.cpp)
OBJ := $(patsubst %.cpp,%.o,$(SRC))

all: $(BINARY)

$(BINARY): $(OBJ)
    $(CXX) -o $(BINARY) $(OBJ) $(LDFLAGS)

%.o: %.cpp %.hpp
    $(CXX) -c $(CXXFLAGS) $<

.PHONY: test
test: $(BINARY)
    ./$(BINARY) < test_stdin.txt > run_stdout.txt
    cmp -s test_stdout.txt run_stdout.txt && echo "PASS" || echo "FAIL"

.PHONY: clean
clean:
    rm $(BINARY) *.o run* *.tar

.PHONY: tar
tar: $(PROJECT).tar

$(PROJECT).tar: $(SRC) Makefile
    tar cf $(PROJECT).tar *.cpp *.hpp Makefile

.PHONY: submit
submit: tar
    python3 submit.py
```