アルゴリズムとデータ構造　第三回レポート課題

2020/06/28

US162039：梶田悠

2020

目次

[課題内容 1](#_Toc44780130)

[処理手順 2](#_Toc44780131)

[ハッシュの構築 2](#_Toc44780132)

[タイトルの検索 3](#_Toc44780133)

[頻度順にソート 3](#_Toc44780134)

[ソースコード 4](#_Toc44780135)

[実行結果 12](#_Toc44780136)

[考察 13](#_Toc44780137)

# 課題内容

Wikipediaのデータを分析し、カテゴリに属するページタイトルを得るためのプログラムを作成する。 Wikipediaの各ページには，複数のカテゴリが付与してある。例えば、「file長良川」のページは「愛知県の河川」、「木曽川水系」、「名水百選」等といったカテゴリが付与されている。Wikipediaのページタイトル（例：「長良川」）と、そのページに付与されたカテゴリを記録したファイル（data.txt）と、全てのカテゴリ一覧記録したファイル（category.list）からは、ページに付与されているカテゴリは分かるが、あるカテゴリに属しているページが分からない。そこで，データを分析し、カテゴリに属するページタイトルを表示できるプログラムを作成した（sample.cpp）。このようなデータを作成しておくことで、カテゴリに属するページを高速に検索することが可能となる。 しかし、sample.cpp は非常に動作が遅く、処理が完了するまで約１時間程度の時間がかかる。そこで、データ構造にハッシュを採用することで処理を高速化させ、短時間（全て表示が終わるまで約１０秒）で処理が終了できるようにプログラムを改良せよ。

# 処理手順

## ハッシュの構築

コードにおいてハッシュから構築されたデータはhashManager変数である。addData関数で、引数でカテゴリとタイトルを渡すことでデータの追加をすることができる。

data.txtファイルからのデータを走査し、ページタイトルに対応する各カテゴリのペアを引数にaddData関数を実行してデータ構造に追加していく。addData関数内での処理は以下のようになっている。

1. カテゴリ名からハッシュ値を計算
2. ハッシュ値をインデックスにtable配列の要素を取得する（要素は線形リストになっている）
3. リストを走査し対象のカテゴリのノードを得る
4. ノードのメンバのタイトルが連続した文字列に引数で渡されたタイトルを結合する（この時同時にページ数をカウントするpageNum変数をインクリメントする）

## タイトルの検索

hashManagerはgetTitles関数でカテゴリを渡すと、対応するタイトルが連続した文字列を取得できる。category.listファイルの内容から全カテゴリを走査し、各カテゴリを引数にgetTitles関数を実行することで全カテゴリに対応するタイトル列を取得する。

getTitles関数内の処理は以下のようになっている。

1. カテゴリ名からハッシュ値を計算
2. ハッシュ値をインデックスにtable配列の要素を取得する（要素は線形リストになっている）
3. リストを走査し対象のカテゴリのノードを得る
4. ノードのメンバのタイトルが連続した文字列を返す

## 頻度順にソート

頻度順にソートするためにまずhashManagerにあるデータを一つのvectorに要素として集める。hashManagerにはtable配列の要素数を返すgetTableSize関数と、引数で渡した整数をインデックスとするtable配列の要素（先頭のノード）を返すgetListAt関数が実装されている。この二つを使ってhashManagerの全要素を走査し、各要素ごとにカテゴリとpageNumをメンバに持つCategoryクラスの変数をつくりvectorに追加していく。このカテゴリのデータが追加されているvectorはコードにおいてはcategories変数である。categoriesをsortCategories関数でソートする。sortCategories関数は前回のレポートのクイックソートをほとんどそのまま使用している。Categoryクラスで比較演算子のオペレータを定義していてpageNumによって比較されるようにしている為にクイックソート自体には特に今回の課題に依存した修正はしていない。

# ソースコード

作成したソースコードは[github](https://github.com/HarukaKajita/DataStructure_report03)にアップロードしています。見づらい場合や情報に不足がある場合は適宜参照して下さい。

コード 　Node.h

|  |
| --- |
| #pragma once  #include <string>  #include <iostream>  using namespace std;  class Node  {  string category = "";  string titles = "";  int pageNum = 0;  Node\* next = NULL;  public:  Node(const string& category);  void addTitle(const string& title);  const string getCategory() const;  const string getTitles() const;  const int getPageNum() const;  Node\* const getNext() const;  void setNext(Node\* node);  const bool operator>(const Node& n) const;  const bool operator<(const Node& n) const;  const bool operator>=(const Node& n) const;  const bool operator<=(const Node& n) const;  }; |

コード 　Node.cpp

|  |
| --- |
| #include "Node.h"  Node::Node(const string& category) {  this->category = category;  }  void Node::addTitle(const string& title) {  this->titles += title;  pageNum++;  }  const string Node::getCategory() const {  return category;  }  const string Node::getTitles() const {  return titles;  }  const int Node::getPageNum() const {  return pageNum;  }  Node\* const Node::getNext() const {  return next;  }  void Node::setNext(Node\* node) {  next = node;  }  const bool Node::operator>(const Node& n) const { return this->pageNum > n.pageNum; }  const bool Node::operator<(const Node& n) const { return this->pageNum < n.pageNum; }  const bool Node::operator>=(const Node& n) const { return this->pageNum >= n.pageNum; }  const bool Node::operator<=(const Node& n) const { return this->pageNum <= n.pageNum; } |

コード 　HashManager.h

|  |
| --- |
| #pragma once  #include <iostream>  #include <string>  #include "Node.h"  using namespace std;  class HashManager  {  //key:category value:head  //map<string, vector<string>\*> headsDictionary;  int size;  Node\*\* table;  int getHash(const string& key) const;  Node\* searchNode(const string& category) const;  public:  HashManager(const int& size);  HashManager();  void addData(const string& category, const string& title);  string getTitles(const string& category) const;  const int getPageNum(const string& category) const;  const int getTableSize() const;  const Node\* getListAt(const int& index) const;  }; |

コード 　HashManager.cpp

|  |
| --- |
| #include "HashManager.h"  HashManager::HashManager(const int& size) {  this->size = size;  table = new Node \* [size];  for (int i = 0; i < size; i++)table[i] = NULL;  //cout << "init HashManager" << sizeof(table) / sizeof(Node\*) << endl;  }  HashManager::HashManager() {  this->size = 1000;  table = new Node \* [size];  for (int i = 0; i < size; i++)table[i] = NULL;  }  int HashManager::getHash(const string& key) const  {  unsigned int v = 0;  for (int i = 0; i < key.length(); i++) v += key[i] \* pow(2, i);  return v % size;  }  Node\* HashManager::searchNode(const string& category) const {  int hashValue = getHash(category);  Node\* node = table[hashValue];  //リストを走査しカテゴリを探す  while (node != NULL && node->getCategory() != category) node = node->getNext();  //なければNULLを返す  return node;  }  void HashManager::addData(const string& category, const string& title) {  const int hashValue = getHash(category);  Node\* node = table[hashValue];  if (node == NULL) {  table[hashValue] = new Node(category);  node = table[hashValue];  }  else {  //リストを走査しカテゴリを探す  while (node != NULL) {  string nodeCategory = node->getCategory();  bool isNotExist = nodeCategory != category;  if (!isNotExist) break;  Node\* next = node->getNext();  //カテゴリがなければ追加する  if (next == NULL) {  Node\* newNode = new Node(category);  node->setNext(newNode);  node = newNode;  break;  }  node = next;  }  }  node->addTitle(title);  }  string HashManager::getTitles(const string& category) const {  int hashValue = getHash(category);  Node\* node = searchNode(category);  if (node == NULL) return "";  return node->getTitles();  }  const int HashManager::getPageNum(const string& category) const {  int hashValue = getHash(category);  Node\* node = searchNode(category);  if (node == NULL) return 0;  return node->getPageNum();  }  const int HashManager::getTableSize() const {  return size;  }  const Node\* HashManager::getListAt(const int& index) const {  return table[index];  } |

コード 　Category.h

|  |
| --- |
| #pragma once  #include <string>  using namespace std;  struct Category  {  private:  string category = "";  int pageNum = 0;  public:  Category(const string category, const int pageNum);  const string getCategory();  const int getPageNum();  const bool operator>(const Category& c) const;  const bool operator<(const Category& c) const;  const bool operator>=(const Category& c) const;  const bool operator<=(const Category& c) const;  }; |

コード 　Category.cpp

|  |
| --- |
| #include "Category.h"  Category::Category(const string category = "", const int pageNum = 0) {  this->category = category;  this->pageNum = pageNum;  }  const string Category::getCategory() {  return category;  }  const int Category::getPageNum() {  return pageNum;  }  const bool Category::operator>(const Category& c) const { return this->pageNum > c.pageNum; }  const bool Category::operator<(const Category& c) const { return this->pageNum < c.pageNum; }  const bool Category::operator>=(const Category& c) const { return this->pageNum >= c.pageNum; }  const bool Category::operator<=(const Category& c) const { return this->pageNum <= c.pageNum; } |

コード 　report03.cpp

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include <fstream>  #include <vector>  #include <stdlib.h>  #include <string>  #include <algorithm>  #include <windows.h>  #include "HashManager.h"  #include "Category.h"  using namespace std;  void exitFailedToOpen(const string& fileName) {  cerr << "failed to open " << fileName << endl;  exit(EXIT\_FAILURE);  }  void setPages(HashManager& hashManager) {  string wikiFileName = "data.txt";  ifstream wikiData(wikiFileName);  if (!wikiData) exitFailedToOpen(wikiFileName);  string line;  while (wikiData >> line)  {  int commaIndex = line.find(",");  string title = line.substr(1, commaIndex-2);  string categories = line.substr(commaIndex + 1);  int n = 0;  for (int i = 0; i <= categories.length(); i = n + 1)  {  n = categories.find\_first\_of(",", i);  if (n == string::npos) n = categories.length();    string category = categories.substr(i, n - i);  hashManager.addData(category, title + ",");  }  }  wikiData.close();  }  void exchangeCategories(Category\* a, Category\* b)  {  Category tmp = \*a;  \*a = \*b;  \*b = tmp;  }  void sortCategories(vector<Category>& categories, const int& left, const int& right)  {  int index1 = left;  int index2 = right;  Category x = categories[left];  do  {  while (categories.at(index1) > x) index1++;  while (categories.at(index2) < x) index2--;  if (index1 <= index2)  {  exchangeCategories(&categories[index1], &categories[index2]);  index1++;  index2--;  }  } while (index1 < index2);  if (left < index2) exchangeCategories(&categories[left], &categories[index2]);  if (left < index2) sortCategories(categories, left, index2);  if (index1 < right) sortCategories(categories, index1, right);  }  int main(int argc, char\* argv[])  {  //open file  string categoryFileName = "category.list";  ifstream categoryData(categoryFileName);  if (!categoryData) exitFailedToOpen(categoryFileName);  string category;  HashManager hashManager = HashManager(10000);    //ハッシュの構築  cout << "ハッシュ構築開始" << endl;  LARGE\_INTEGER freq;  QueryPerformanceFrequency(&freq);  LARGE\_INTEGER start, end;  QueryPerformanceCounter(&start);  setPages(hashManager);  QueryPerformanceCounter(&end);  double time = static\_cast<double>(end.QuadPart - start.QuadPart) \* 1000.0 / freq.QuadPart;  printf("time %lf[ms]\n", time);  cout << "ハッシュ構築完了" << endl;  //１カテゴリ->タイトル検索  //全件表示  while (categoryData >> category)  {  cout << category << " -> ";  string titles = hashManager.getTitles(category);  cout << titles << endl;  cout << endl;  }  //２頻度ソート  cout << "カテゴリリスト構築開始" << endl;  vector<Category> categories;  const int tableSize = hashManager.getTableSize();  for (size\_t i = 0; i < tableSize; i++)  {  const Node\* node = hashManager.getListAt(i);  while (node != NULL) {  const string category = node->getCategory();  const int pageNum = node->getPageNum();  categories.push\_back(Category(category, pageNum));  node = node->getNext();  }  }  cout << "カテゴリリスト構築終了" << endl;  cout << "カテゴリリストソート開始" << endl;  // 時間計測  QueryPerformanceFrequency(&freq);  QueryPerformanceCounter(&start);  sortCategories(categories, 0, categories.size()-1);  QueryPerformanceCounter(&end);  time = static\_cast<double>(end.QuadPart - start.QuadPart) \* 1000.0 / freq.QuadPart;  printf("time %lf[ms]\n", time);  cout << "カテゴリリストソート終了" << endl;  //全件表示  for (int i = 0; i < categories.size(); i++)  cout << categories.at(i).getCategory() << "：" << categories.at(i).getPageNum() << endl;    return 0;  } |

# 実行結果

実行結果 　課題1のreport03.cppの実行結果の抜粋

|  |
| --- |
| .NETFramework -> アセンブリ　（共通言語基盤）,．ＮＥＴ　Ｃｏｒｅ,Ｓｅａｓａｒ,ＡＳＰ．ＮＥＴ　ＭＶＣ　Ｆｒａｍｅｗｏｒｋ,Ｅｎｔｉｔｙ　Ｆｒａｍｅｗｏｒｋ,Ｆ　Ｓｈａｒｐ,Ｗｉｎｄｏｗｓ　Ｐｒｅｓｅｎｔａｔｉｏｎ　Ｆｏｕｎｄａｔｉｏｎ,Ｍｉｃｒｏｓｏｆｔ　Ｓｉｌｖｅｒｌｉｇｈｔ,ＣｌｉｃｋＯｎｃｅ,Ｗｉｎｄｏｗｓ　Ｆｏｒｍｓ,Ｗｉｎｄｏｗｓ　Ｃｏｍｍｕｎｉｃａｔｉ ｏｎ　Ｆｏｕｎｄａｔｉｏｎ,デリゲート　（プログラミング）,Ｗｉｎｄｏｗｓ　ＣａｒｄＳｐａｃｅ,Ｃ　Ｓｈａｒｐ,  .hack -> ．ｈａｃｋ／／Ｌｉｎｋ,．ｈａｃｋ／／Ｎｅｗ　Ｗｏｒｌｄ,．ｈａｃｋ／／黄昏の腕輪伝説,．ｈａｃｋ／／Ｇ．Ｕ．　（ゲーム）,．ｈａｃｋ／／ｂｕｌｌｅｔ,ギルティドラゴン　罪竜と八つの呪い,．ｈａｃｋ／／Ｇ．Ｕ．　（小説）,Ｔｈｅ　Ｗｏｒｌｄ　（．ｈａｃｋ）,．ｈａｃｋ／／Ｇ．Ｕ．＋,．ｈａｃｋ　（ゲーム）,ヴィクトリースパーク,．ｈａｃｋ／／Ａｎｏｔｈｅｒ　Ｂｉｒｔｈ　もうひとつの誕生,  007シリーズの映画 -> ００７は二度死ぬ,００７　サンダーボール作戦,００７　ゴールドフィンガー,００７　ダイヤモンドは永遠に,００７　消されたライセンス,００７　死ぬのは奴らだ,００７　ムーンレイカー,００７　ロシアより愛をこめて,００７　ゴールデ ンアイ,ネバーセイ・ネバーアゲイン,００７　カジノロワイヤル　（１９６７年の映画）,  105mm砲 -> 試製５７式１０５ｍｍ軽りゅう弾砲,Ｍ５２　１０５ｍｍ自走榴弾砲,九九式十糎山砲,１０．５ｃｍ　ｌｅＦＨ　１８,１ ０．５ｃｍ　ｓＫ　１８,１０ｃｍ　Ｋ　１７,オート・メラーラＭｏｄ５６　１０５ｍｍ榴弾砲,１０．５　ｃｍ　ＦｌａＫ　３８,１０．５ｃｍ突撃榴弾砲４２,九一式十糎榴弾砲,三八式十糎加農砲,十四年式十糎加農砲,６０式自走無反動砲,ＬＧ１　１０５ｍｍ榴弾 砲,一式十糎自走砲,Ｌ１１８　１０５ｍｍ榴弾砲,試製五式砲戦車,九二式十糎加農砲,Ｍ３　１０５ｍｍ榴弾砲,Ｍ１０１　１０５ｍｍ榴弾砲,Ｍ１０８　１０５ｍｍ自走榴弾砲,Ｍ１１２８　ストライカーＭＧＳ,  10世紀の書籍 -> 医心方,占事略决,西宮記,旧唐書,蜻蛉日記,御堂関白記,太平広記,唐会要,太平御覧,  鼠小僧を題材にしたフィクション作品 -> てなもんや東海道,天下堂々,弥次喜多道中記,がんばれゴエモン！からくり道中,幕末てなもんや大騒動,浮世絵　女ねずみ小僧,江戸を斬る　（西郷輝彦）,  鼻 -> 鼻行類,鼻フック,鼻茸,副鼻腔,鼻血,鼻眼鏡,  龍が如くシリーズ -> 龍が如く,錦山彰,龍が如く　見参！,冴島大河,澤村遥,龍が如く２,風間新太郎,柏木修,龍が如く６　命の詩。, 龍が如く　維新！,真島吾朗,龍が如く３,桐生一馬,龍が如く０　誓いの場所,龍が如く７　光と闇の行方,龍が如く　ＯＮＬＩＮＥ,北 斗が如く,龍が如く４　伝説を継ぐもの,郷田龍司,龍が如く　ＯＦ　ＴＨＥ　ＥＮＤ,クロヒョウ　龍が如く新章,秋山駿　（龍が如く シリーズ）,龍が如く５　夢、叶えし者,  龍居由佳里脚本のテレビドラマ -> 星の金貨,ストロベリーナイト　（テレビドラマ）,流転の王妃・最後の皇弟,ピュア　（テレビド ラマ）,二十四の瞳,天使の歌声　?小児病棟の奇跡?,  ［Alexandros］のアルバム -> ＥＸＩＳＴ！,Ｗｈｅｒｅ＇ｓ　Ｍｙ　Ｐｏｔａｔｏ？,Ｓｌｅｅｐｌｅｓｓ　ｉｎ　Ｂｒｏｏｋｌｙｎ,ＡＬＸＤ,Ｍｅ　Ｎｏ　Ｄｏ　Ｋａｒａｔｅ．,Ｓｃｈｗａｒｚｅｎｅｇｇｅｒ, |

実行結果 　課題2の実行結果の抜粋

|  |
| --- |
| 存命人物：1906  携帯電話端末：1761  学校記事：1695  継続中の作品：1063  Windows用ゲームソフト：633  PlayStation2用ソフト：606  天文学に関する記事：601  国際ローミング対応機種：536  PlayStation用ソフト：484  東証一部上場企業：478 |

# 考察

イントロソート

課題２のソート問題を解く過程でstd::sortを実行してみたところ、自分が実装したクイックソートより明らかに計算時間が短かった。調べたところstd::sortはイントロソートというアルゴリズムを採用していることが分かった。クイックソートの最悪計算時間はO(n^2)であり、計算時間が長くなる様な場合にアルゴリズムをクイックソートからヒープソートに変えることで全体の計算時間を短くしたアルゴリズムである。今回は時間の都合で実装しなかったが、部分的に別のアルゴリズムに変えることでより適したアルゴリズムを作る考え方を学んだ。

リストではなくツリー

前述の部分的に採用するアルゴリズムを変える考え方から、今回実装したハッシュを使ったデータ構造の改良案を考えた。しかし、これも時間の都合で今回は実装できていない。今回実装したデータ構造はポインタの配列をテーブルとして持ち、各要素は線形リストの先頭要素となる構造をしていた。線形リストの部分を二分木にすることでデータの追加、検索の速度を上げられると考えた。テーブルの要素を二分木のルートノードとしてテーブルの要素数分の二分木が存在するデータ構造である。追加、検索したいデータからハッシュを計算しこれをインデックスとしてテーブルからポインタを取得する。この時点で再度ハッシュ値を計算する。この時使うハッシュ関数は前の工程で使ったものとは別のもので、二分木を構成する為の新しい値である。ノードはそれぞれデータとハッシュ値を持っており、二分木は親ノードのハッシュ値以上のハッシュ値をもつノードは親の右に、親ノードのハッシュ値より小さいハッシュ値をもつノードは親の左に続く構造をする。線形リストの場合データを追加する場合はリストの末尾までノードを辿る必要があり、検索する場合も線形に探索する必要があるので最悪の場合リストの末尾までノードを辿る必要がある。しかし、この二分木の構造であれば平均的には計s何時間が短くなることが期待できる。