Tentamen
C++-programmering

2010–03–11, 8.00–13.00

Hjälpmedel: En valfri C++-bok. OH-bilderna från föreläsningarna är *inte* tillåtna.

Du ska i dina lösningar visa att du behärskar C++ och att du kan använda C++ standardklasser. ”C-lösningar” ger inga poäng, även om de är korrekta.

Uppgifterna ger preliminärt $10 + 20 + 20 = 50$ poäng. För godkänt krävs 25 poäng (3/25, 4/33, 5/42 respektive G/25, VG/38).

---

1. Vi vill i en klass hålla reda på namn. Namnen lagrar vi i ett objekt av STL-klassen `set`. Vi har valt att i mängden lagra *pekare* på strängarna som innehåller namnen. Klassen ser ut så här:

```
class NameList {
public:
    NameList() {}
    ~NameList() {}
    void insert(const std::string& name) {
        names.insert(new std::string(name));
    }
    void printSorted() const {
        for (list_type::const_iterator it = names.begin(); it != names.end(); ++it) {
            std::cout << *it << std::endl;
        }
    }
private:
    typedef std::set<std::string*> list_type;
    list_type names;
};
```

   a) Klassen innehåller en uppenbar minnesläcka. Korrigera den.

   b) Om man skriver (och anropar) en funktion `void f(NameList nl)` får man också minnesproblem. Varför? Nämn två sätt att komma tillrätta med problemet.

   c) Utskriften i `printSorted` blir inte alls som förväntat — man får hexadecimala tal i stället för namn. Varför? Korrigera funktionen så att det skrivs ut namn i stället för tal.

   d) Även efter korrigeringen blir det fel — namnen skrivs inte ut i sorterad ordning, trots att ett `set` ju ska hålla elementen sorterade. Varför inte? Korrigera också detta fel.

   e) Det vore bra om man kunde skriva ut en namnlista med `cout << nl`. Implementera detta (visa också vilka ändringar du behöver göra i klassen `NameList`). Förutsätt att `printSorted` tas bort.

2. Uppgiften handlar om inverterade index, som man använder för att ta reda på dokument där ord förekommer. Läs mera i den bifogade Wikipedia-artikeln.

   Du ska skriva en klass som kan 1) bygga upp och spara ett inverterat index (”record level inverted index”), 2) läsa in ett index och utföra sökningar. Exempel på program som utför dessa båda moment:

```
int main() {
    vector<Index::docname> doclist; // list with names of documents (files) which
    doclist.push_back("t0");        // shall be indexed (docname is std::string)
    doclist.push_back("t1");
    doclist.push_back("t2");
    Index ind;
    ind.build(doclist);             // build the index
    ofstream out("index.txt");
    ind.write(out);                 // write it to the file
}

int main(int argc, char** argv) {
    Index ind;
    ifstream in("index.txt");
    ind.read(in);                   // read the index from the file
    vector<string> words;           // words to search for
    for (int i = 1; i != argc; ++i) {
        words.push_back(argv[i]);
    }
    Index::docset res = ind.search(words); // res is a set with names of the
                                    // documents in which all of the words occur
    for (Index::docset::const_iterator it = res.begin(); it != res.end(); ++it) {
        cout << *it << " ";
    }
    cout << endl;
}
```

   Klassen `Index` har följande specifikation:

```
class Index {
public:
    typedef std::string docname;
    typedef std::set<docname> docset;
    void build(const std::vector<docname>& doclist);
    void write(std::ostream& out) const;
    void read(std::istream& in);
    docset search(const std::vector<std::string>& wordlist) const;
private:
    typedef std::map<std::string, docset> index_type;
    index_type index;
};
```

   Implementera medlemsfunktionerna. Anvisningar:

   - Du ska använda den givna datastrukturen för indexet (`index_type`).
   - Du kan förutsätta att alla filerna som ska indexeras existerar. Filerna innehåller bara ord med små bokstäver åtskilda av whitespace, inga skiljetecken (som i Wikipedia-exemplet).
   - Funktionen `write` ska skriva ut indexet i ett format som kan läsas av funktionen `read`.
   - När man i `search` ska beräkna snittet mellan två mängder är det enklast att använda STL-algoritmen `set_intersection` och lagra resultatet i en ny mängd (med en insert-iterator).

3. XML (eXtended Markup Language) är ett språk för beskrivning av ”halvstrukturerade” data i form av text. XML liknar HTML på det sättet att strukturen i ett dokument beskrivs av taggar, men reglerna för XML är mera strikta än för HTML. De viktigaste reglerna är:

   - Ett dokument består av *en* XML-nod.
   - En nod inleds med en starttagg, därefter ett antal tecken och sist en sluttagg.
   - Noder får kapslas, dvs tecknen i en nod får innehålla nya noder.
   - En starttag har utseendet `<text>`, en sluttagg utseendet `</text>`. En starttagg och motsvarande sluttagg måste ha samma text. Texten är godtycklig.

   Exempel på korrekta XML-dokument (`abc` står för en följd godtyckliga tecken, utom `<`):

   ```
   <d1>abc</d1>
   <d1>abc<d2>abc</d2>abc<d2>abc</d2></d1>
   ```

   Exempel på felaktiga XML-dokument:

   ```
   abc                          // ingen nod
   <d1>abc</d1><d2>abc</d2>     // två noder
   <d1>abc                      // sluttagg saknas
   <d1>abc</d1></d1>            // starttagg saknas
   <d1>abc</d2>                 // start- och sluttagg inte samma
   <d1>abc<d2>abc</d1></d2>     // dito
   ```

   Funktionen `parse()` i klassen `XMLParser` läser en inström och avgör om strömmen innehåller ett korrekt XML-dokument:

   ```
   struct parse_error {};

   class XMLParser {
   public:
       XMLParser(istream& input) : in(input) {}
       void parse() throw(parse_error); // parse the document in the 'in' stream
   private:
       istream& in; // stream to read from
       struct Tag {
           string text;
           enum {open, close} kind;
       };
       Tag get_tag() throw(parse_error); // skip everything up to next tag, return it
                                         // throws parse_error if ill-formed tag
                                         // returns an empty tag and in.eof() true if
                                         // there are no more tags
   };
   ```

   a) Skriv en `main`-funktion som läser en fil och som skriver ut `Wellformed XML` om filen innehåller ett korrekt XML-dokument, `Syntax error` annars. Namnet på filen ska ges som argument på kommandoraden. Om argumentet saknas eller filen inte går att öppna ska ett lämpligt felmeddelande skrivas ut.

   b) Implementera funktionen `parse()`. Funktionen `get_tag()` är färdigskriven; den ska du naturligtvis utnyttja.

Inverted index

From Wikipedia, the free encyclopedia

In computer science, an **inverted index** (also referred to as **postings file** or **inverted file**) is an index data structure storing a mapping from content, such as words or numbers, to its locations in a database file, or in a document or a set of documents. The purpose of an inverted index is to allow fast full text searches, at a cost of increased processing when a document is added to the database. The inverted file may be the database file itself, rather than its index. It is the most popular data structure used in document retrieval systems.[1] Several significant general-purpose mainframe-based database management systems have used inverted list architectures, including ADABAS, DATACOM/DB, and Model 204.

There are two main variants of inverted indexes: A **record level inverted index** (or **inverted file index** or just **inverted file**) contains a list of references to documents for each word. A **word level inverted index** (or **full inverted index** or **inverted list**) additionally contains the positions of each word within a document.[2] The latter form offers more functionality (like phrase searches), but needs more time and space to be created.

Contents

Example

Given the texts $T_0 =$ `"it is what it is"`, $T_1 =$ `"what is it"` and $T_2 =$ `"it is a banana"`, we have the following inverted file index (where the integers in the set notation brackets refer to the subscripts of the text symbols, T_0, T_1 etc.):

```
"a":      {2}
"banana": {2}
"is":     {0, 1, 2}
"it":     {0, 1, 2}
"what":   {0, 1}
```

A term search for the terms `"what"`, `"is"` and `"it"` would give the set $\{0,1\} \cap \{0,1,2\} \cap \{0,1,2\} = \{0,1\}$.

With the same texts, we get the following full inverted index, where the pairs are document numbers and

local word numbers. Like the document numbers, local word numbers also begin with zero. So, `"banana": {(2, 3)}` means the word "banana" is in the third document (T_2), and it is the fourth word in that document (position 3).

```
"a":      {(2, 2)}
"banana": {(2, 3)}
"is":     {(0, 1), (0, 4), (1, 1), (2, 1)}
"it":     {(0, 0), (0, 3), (1, 2), (2, 0)}
"what":   {(0, 2), (1, 0)}
```

If we run a phrase search for `"what is it"` we get hits for all the words in both document 0 and 1. But the terms occur consecutively only in document 1.

Applications

The inverted index data structure is a central component of a typical search engine indexing algorithm. A goal of a search engine implementation is to optimize the speed of the query: find the documents where word X occurs. Once a forward index is developed, which stores lists of words per document, it is next inverted to develop an inverted index. Querying the forward index would require sequential iteration through each document and to each word to verify a matching document. The time, memory, and processing resources to perform such a query are not always technically realistic. Instead of listing the words per document in the forward index, the inverted index data structure is developed which lists the documents per word.

With the inverted index created, the query can now be resolved by jumping to the word id (via random access) in the inverted index.

In pre-computer times, concordances to important books were manually assembled. These were effectively inverted indexes with a small amount of accompanying commentary, that required a tremendous amount of effort to produce.

See also

- Vector space model
- Index (search engine)
- SQL Server Full Text Search (component of)

Bibliography

- Knuth, D. E. (1997) [1973]. *The Art of Computer Programming* (Third ed.). Reading, Massachusetts: Addison-Wesley. ISBN 0-201-89685-0.
- Zobel, Justin; Moffat, Alistair; Ramamohanarao, Kotagiri (December 1998). "Inverted files versus signature files for text indexing". *ACM Transactions on Database Systems* (New York: Association for Computing Machinery) **23** (4): pp. 453–490. doi:10.1145/296854.277632 (http://dx.doi.org/10.1145%2F296854.277632) .
- Zobel, Justin RMIT University, Australia; Moffat, Alistair The University of Melbourne, Australia (July 2006). "Inverted Files for Text Search Engines". *ACM Computing Surveys* (New York: Association for Computing Machinery) **38** (2): 6. doi:10.1145/1132956.1132959

(http://dx.doi.org/10.1145%2F1132956.1132959) .
- Baeza-Yates, Ricardo; Ribeiro-Neto, Berthier (1999). *Modern information retrieval*. Reading, Massachusetts: Addison-Wesley Longman. p. 192. ISBN 0-201-39829-X.
- Salton, Gerard; Fox, Edward A.; Wu, Harry (1983). "Extended Boolean information retrieval". *Commun. ACM* (ACM) **26** (11). doi:10.1145/182.358466 (http://dx.doi.org/10.1145%2F182.358466) .

References

- Knuth 1997, pp. 560–563 of section 6.5: *Retrieval on Secondary Keys*

1. ^ Zobel, Moffat & Ramamohanarao 1998
2. ^ Baeza-Yates & Ribeiro-Neto 1999, p. 192

External links

- NIST's Dictionary of Algorithms and Data Structures: inverted index (http://www.nist.gov/dads/HTML/invertedIndex.html)
- Managing Gigabytes for Java (http://mg4j.dsi.unimi.it) a free full-text search engine for large document collections written in Java.
- Lucene (http://lucene.apache.org/java/docs/) - Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

Retrieved from "http://en.wikipedia.org/wiki/Inverted_index"
Categories: Data management | Search algorithms

- This page was last modified on 22 February 2010 at 07:03.
- Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. See Terms of Use for details.
  Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.