

《The Anatomy of a Large-Scale Hypertextual Web Search Engine》中文版

摘要

這篇文章中，我們介紹了 google，它是一個大型的搜索引擎（of a large-scale search engine）的原型，搜索引擎在超文本中應用廣泛。Google 的設計能夠高效地抓網頁並建立索引，它的查詢結果比其他現有系統都高明。這個原型的全文和超連接的資料庫至少包含 24 '000 '000 個網頁。我們可以從 <http://google.stanford.edu/> 下載。

設計搜索引擎是一項富有挑戰性的工作。搜索引擎為上億個網頁建立索引，其中包含大量迥然不同的辭彙。而且每天要回答成千上萬個查詢。在網路中，儘管大型搜索引擎非常重要，但是學術界卻很少研究它。此外由於技術的快速發展和網頁的大量增加，現在建立一個搜索引擎和三年前完全不同。

本文詳細介紹了我們的大型搜索引擎，據我們所知，在公開發表的論文中，這是第一篇描述地如此詳細。除了把傳統資料搜索技術應用到如此大量級網頁中所遇到的問題，還有許多新的技術挑戰，包括應用超文本中的附加資訊改進搜索結果。

本文將解決這個問題，描述如何運用超文本中的附加資訊，建立一個大型實用系統。任何人都可以在網上隨意發佈資訊，如何有效地處理這些無組織的超文本集合，也是本文要關注的問題。關鍵字 World Wide Web，搜索引擎，資訊檢索，PageRank, Google 1 緒論 Web 給資訊檢索帶來了新的挑戰。Web 上的信息量快速增長，同時不斷有毫無經驗的新用戶來體驗 Web 這門藝術。人們喜歡用超連結來編網，通常都以象 Yahoo 這樣重要的網頁或搜索引擎開始。大家認為 List(目錄)有效地包含了大家感興趣的主題，但是它具有主觀性，建立和維護的代價高，升級慢，不能包括所有深奧的主題。基於關鍵字的自動搜索引擎通常返回太多的低品質的匹配。使問題更遭的是，一些廣告為了贏得人們的關注想方設法誤導自動搜索引擎。

我們建立了一個大型搜索引擎解決了現有系統中的很多問題。應用超文本結構，大大提高了查詢品質。我們的系統命名為 google，取名自 googol 的通俗拼法，即 10 的 100 次方，這和我們的目標建立一個大型搜索引擎不謀而合。

1.1 網路搜索引擎一升級換代（scaling up）：1994-2000 搜索引擎技術不得不快速升級（scale dramatically）跟上成倍增長的 web 數量。1994 年，第一個 Web 搜索引擎，World Wide Web Worm(WWWW)可以檢索到 110,000 個網頁和 Web 的文件。到 1994 年 11 月，頂級的搜索引擎聲稱可以檢索到 2 '000' 000 (WebCrawler) 至 100 '000' 000 個網路檔（來自 Search Engine Watch）。可以預見到 2000 年，可檢索到的網頁將超過 1 '000' 000 '000。同時，搜索引擎的訪問量也會以驚人的速度增長。在 1997 年的三四月份，World Wide Web Worm 平均每天收到 1500 個查詢。

在 1997 年 11 月，Altavista 聲稱它每天要處理大約 20' 000' 000 個查詢。隨著網路用戶的增長，到 2000 年，自動搜索引擎每天將處理上億個查詢。我們系統的設計目標要解決許多問題，包括品質和可升級性，引入升級搜索引擎技術（scaling search engine technology），把它升級到如此大量的資料上。

1.2 Google：跟上 Web 的步伐（Scaling with the Web）建立一個能夠和當今 web 規模相適應的搜索引擎會面臨許多挑戰。抓網頁技術必須足夠快，才能跟上網頁變化的速度（keep them up to

date)。存儲索引和文檔的空間必須足夠大。索引系統必須能夠有效地處理上千億的資料。處理查詢必須快，達到每秒能處理成百上千個查詢 (hundreds to thousands per second.)。隨著 Web 的不斷增長，這些任務變得越來越艱巨。然而硬體的執行效率和成本也在快速增長，可以部分抵消這些困難。

還有幾個值得注意的因素，如磁片的尋道時間 (disk seek time)，作業系統的效率 (operating system robustness)。在設計 Google 的過程中，我們既考慮了 Web 的增長速度，又考慮了技術的更新。Google 的設計能夠很好的升級處理海量資料集。它能夠有效地利用存儲空間來存儲索引。優化的資料結構能夠快速有效地存取 (參考 4.2 節)。進一步，我們希望，相對於所抓取的文字檔案和 HTML 網頁的數量而言，存儲和建立索引的代價盡可能的小 (參考附錄 B)。對於象 Google 這樣的集中式系統，採取這些措施得到了令人滿意的系統可升級性 (scaling properties)。

1.3 設計目標

1.3.1 提高搜索品質我們的主要目標是提高 Web 搜索引擎的品質。1994 年，有人認為建立全搜索索引 (a complete search index) 可以使查找任何資料都變得容易。根據 Best of the Web 1994 -- Navigators，“最好的導航服務可以使在 Web 上搜索任何資訊都很容易 (當時所有的資料都可以被登錄)”。然而 1997 年的 Web 就迥然不同。近來搜索引擎的用戶已經證實索引的完整性不是評價搜索品質的唯一標準。用戶感興趣的搜索結果往往湮沒在“垃圾結果 Junk result”中。實際上，到 1997 年 11 月為止，四大商業搜索引擎中只有一個能夠找到它自己 (搜索自己名字時返回的前十個結果中有它自己)。導致這一問題的主要原因是文檔的索引數目增加了好幾個數量級，但是用戶能夠看的文檔數卻沒有增加。用戶仍然只希望看前面幾十個搜索結果。因此，當集合增大時，我們就需要工具使結果精確 (在返回的前幾十個結果中，有關文檔的數量)。由於是從成千上萬個有點相關的文檔中選出幾十個，實際上，相關的概念就是指最好的文檔。高精確非常重要，甚至以回應 (系統能夠返回的有關文檔的總數) 為代價。令人高興的是利用超文本鏈結提供的資訊有助於改進搜索和其他應用。尤其是鏈結結構和鏈結文本，為相關性的判斷和高品質的過濾提供了大量的資訊。Google 既利用了鏈結結構又用到了 anchor 文本 (見 2.1 和 2.2 節)。

1.3.2 搜索引擎的學術研究隨著時間的流逝，除了發展迅速，Web 越來越商業化。1993 年，只有 1.5% 的 Web 服務是來自 .com 功能變數名稱。到 1997 年，超過了 60%。同時，搜索引擎從學術領域走進商業。到現在大多數搜索引擎被公司所有，很少技公開術細節。這就導致搜索引擎技術很大程度上仍然是暗箱操作，並傾向做廣告 (見附錄 A)。Google 的主要目標是推動學術領域在此方面的發展，和對它的瞭解。另一個設計目標是給大家一個實用的系統。應用對我們來說非常重要，因為現代網路系統中存在大量的有用資料 (us because we think some of the most interesting research will involve leveraging the vast amount of usage data that is available from modern web systems)。例如，每天有幾千萬個研究。然而，得到這些資料卻非常困難，主要因為它們沒有商業價值。我們最後的設計目標是建立一個體系結構能夠支援新的關於海量 Web 資料的研究。為了支持新研究，Google 以壓縮的形式保存了實際所抓到的文檔。設計 google 的目標之一就是要建立一個環境使其他研究者能夠很快進入這個領域，處理海量 Web 資料，得到滿意的結果，而通過其他方法卻很難得到結果。系統在短時間內被建立起來，已經有幾篇論文用到了 Google 建的資料庫，更多的在起步中。我們的另一個目標是建立一個宇宙空間實驗室似的環境，在這裏研究者甚至學生都可以對我們的海量 Web 資料設計或做一些實驗。

2. 系統特點 Google 搜索引擎有兩個重要特點，有助於得到高精度的搜索結果。

第一點，應用 Web 的鏈結結構計算每個網頁的 Rank 值，稱為 PageRank，將在 98 頁詳細描述它。

第二點，Google 利用超鏈結改進搜索結果。

2.1 PageRank:給網頁排序 Web 的引用（鏈結）圖是重要的資源，卻被當今的搜索引擎很大程度上忽視了。我們建立了一個包含 518 '000' 000 個超鏈結的圖，它是一個具有重要意義的樣本。這些圖能夠快速地計算網頁的 PageRank 值，它是一個客觀的標準，較好的符合人們心目中對一個網頁重要程度的評價，建立的基礎是通過引用判斷重要性。因此在 web 中，PageRank 能夠優化關鍵字查詢的結果。對於大多數的主題，在網頁標題查詢中用 PageRank 優化簡單文本匹配，我們得到了令人驚歎的結果（從 google.stanford.edu 可以得到演示）。對於 Google 主系統中的全文搜索，PageRank 也幫了不少忙。

2.1.1 計算 PageRank 文獻檢索中的引用理論用到 Web 中，引用網頁的鏈結數，一定程度上反映了該網頁的重要性和品質。PageRank 發展了這種思想，網頁間的鏈結是不平等的。

PageRank 定義如下：我們假設 $T_1 \dots T_n$ 指向網頁 A（例如，被引用）。參數 d 是制動因數，使結果在 0, 1 之間。通常 d 等於 0.85。在下一節將詳細介紹 d 。 $C(A)$ 定義為網頁 A 指向其他網頁的鏈結數，網頁 A 的 PageRank 值由下式給出： $PR(A) = (1-d) + d(PR(T_1)/C(T_1) + \dots + PR(T_n)/C(T_n))$ 注意 PageRank 的形式，分佈到各個網頁中，因此所有網頁的 PageRank 和是 1。PageRank 或 $PR(A)$ 可以用簡單的迭代演算法計算，相應規格化 Web 鏈結矩陣的主特徵向量。中等規模的網站計算 26 '000' 000 網頁的 PageRank 值要花費幾小時。還有一些技術細節超出了本文論述的範圍。

2.1.2 直覺判斷 PageRank 被看作用戶行為的模型。我們假設網是隨機的，不中斷點擊鏈結，從不返回，最終煩了，另外隨機選一個網頁重新開始衝浪。隨機訪問一個網頁的可能性就是它的 PageRank 值。制動因數 d 是隨機訪問一個網頁煩了的可能性，隨機另選一個網頁。對單個網頁或一組網頁，一個重要的變數加入到制動因數 d 中。這允許個人可以故意地誤導系統，以得到較高的 PageRank 值。我們還有其他的 PageRank 演算法，見 98 頁。

另外的直覺判斷是一個網頁有很多網頁指向它，或者一些 PageRank 值高的網頁指向它，則這個網頁很重要。直覺地，在 Web 中，一個網頁被很多網頁引用，那麼這個網頁值得一看。一個網頁被象 Yahoo 這樣重要的主頁引用即使一次，也值得一看。如果一個網頁的品質不高，或者是死鏈結，象 Yahoo 這樣的主頁不會鏈向它。PageRank 處理了這兩方面因素，並通過網路鏈結遞迴地傳遞。

2.2 鏈結描述文字（Anchor Text）我們的搜索引擎對鏈結文本進行了特殊的處理。大多數搜索引擎把鏈結文字和它所鏈向的網頁（the page that the link is on）聯繫起來。另外，把它和鏈結所指向的網頁聯繫起來。這有幾點好處。

第一，通常鏈結描述文字比網頁本身更精確地描述該網頁。

第二，鏈結描述文字可能鏈向的文檔不能被文本搜索引擎檢索到，例如圖像，程式和資料庫。有可能使返回的網頁不能被抓到。注意哪些抓不到的網頁將會帶來一些問題。在返回給用戶前檢測不了它們的有效性。這種情況搜索引擎可能返回一個根本不存在的網頁，但是有超鏈結指向它。然而這種結果可以被挑出來的，所以此類的問題很少發生。鏈結描述文字是對被鏈向網頁的宣傳，這個思想被用在 World Wide Web Worm 中，主要因為它有助於搜索非文本資訊，能夠用少

量的已下載文檔擴大搜索範圍。我們大量應用鏈結描述文字，因為它有助於提高搜索結果的品質。有效地利用鏈結描述文字技術上存在一些困難，因為必須處理大量的資料。現在我們能抓到 24 '000' 000 個網頁，已經檢索到 259 '000' 000 多個鏈結描述文字。

2.3 其他特點除了 PageRank 和應用鏈結描述文字外，Google 還有一些其他特點。

第一，所有 hit 都有位置資訊，所以它可以在搜索中廣泛應用鄰近性 (proximity)。

第二，Google 跟蹤一些視覺化外表細節，例如字型大小。黑體大號字比其他文字更重要。

第三，知識庫存儲了原始的全文 html 網頁。

3 有關工作 Web 檢索研究的歷史簡短。World Wide Web Worm () 是最早的搜索引擎之一。後來出現了一些用於學術研究的搜索引擎，現在它們中的大多數被上市公司擁有。與 Web 的增長和搜索引擎的重要性相比，有關當今搜索引擎技術的優秀論文相當少。根據 Michael Mauldin (Lycos Inc 的首席科學家)， “各種各樣的服務 (包括 Lycos) 非常關注這些資料庫的細節。” 雖然在搜索引擎的某些特點上做了大量工作。具有代表性的工作有，對現有商業搜索引擎的結果進行傳遞，或建立小型的個性化的搜索引擎。最後有關資訊檢索系統的研究很多，尤其在有組織機構集合 (well controlled collections) 方面。在下面兩節，我們將討論在資訊檢索系統中的哪些領域需要改進以便更好的工作在 Web 上。

3.1 資訊檢索資訊檢索系統誕生在幾年前，並發展迅速。然而大多數資訊檢索系統研究的物件是小規模的單一的有組織結構的集合，例如科學論文集，或相關主題的新聞故事。實際上，資訊檢索的主要基準，the Text Retrieval Conference ()，用小規模的、有組織結構的集合作為它們的基準。

大型文集基準只有 20GB，相比之下，我們抓到的 24000000 個網頁占 147GB。在 TREC 上工作良好的系統，在 Web 上卻不一定產生好的結果。例如，標準向量空間模型企圖返回和查詢請求最相近的文檔，把查詢請求和文檔都看作由出現在它們中的辭彙組成的向量。在 Web 環境下，這種策略常常返回非常短的文檔，這些文檔往往是查詢詞再加幾個字。例如，查詢 “Bill Clinton”，返回的網頁只包含 “Bill Clinton Sucks”，這是我們從一個主要搜索引擎中看到的。網路上有些爭議，用戶應該更準確地表達他們想查詢什麼，在他們的查詢請求中用更多的詞。我們強烈反對這種觀點。如果用戶提出象 “Bill Clinton” 這樣的查詢請求，應該得到理想的查詢結果，因為這個主題有許多高品質的資訊。象所給的例子，我們認為資訊檢索標準需要發展，以便有效地處理 Web 資料。

3.2 有組織結構的集合 (Well Controlled Collections) 與 Web 的不同點 Web 是完全無組織的異構的大量文檔的集合。Web 中的文檔無論內在資訊還是隱含資訊都存在大量的異構性。例如，文檔內部就用了不同的語言 (既有人類語言又有程式)，辭彙 (email 位址，鏈結，郵遞區號，電話號碼，產品號)，類型 (文本，HTML，PDF，圖像，聲音)，有些甚至是機器創建的檔 (log 檔，或資料庫的輸出)。可以從文檔中推斷出來，但並不包含在文檔中的資訊稱為隱含資訊。隱含資訊包括來源的信譽，更新頻率，品質，訪問量和引用。不但隱含資訊的可能來源各種各樣，而且被檢測的資訊也大不相同，相差可達好幾個數量級。例如，一個重要主頁的使用量，象 Yahoo 每天流覽數達到上百萬次，于此相比無名的歷史文章可能十年才被訪問一次。很明顯，搜索引擎對這兩類資訊的處理是不同的。Web 與有組織結構集合之間的另外一個明顯區別是，事實上，向 Web 上傳資訊沒有任何限制。靈活利用這點可以發佈任何對搜索引擎影響重大的資訊，使路由

阻塞，加上為牟利故意操縱搜索引擎，這些已經成為一個嚴重的問題。這些問題還沒有被傳統的封閉的資訊檢索系統所提出來。它關心的是元資料的努力，這在 Web 搜索引擎中卻不適用，因為網頁中的任何文本都不會向用戶聲稱企圖操縱搜索引擎。甚至有些公司為牟利專門操縱搜索引擎。

4 系統分析 (System Anatomy) 首先，我們提供高水準的有關體系結構的討論。然後，詳細描述重要的資料結構。最後，主要應用：抓網頁，索引，搜索將被嚴格地檢查。 Figure 1. High Level Google Architecture 4.1Google 體系結構概述這一節，我們將看看整個系統是如何工作的 (give a high level)，見圖 1。本節不討論應用和資料結構，在後幾節中討論。為了效率大部分 Google 是用 c 或 c++實現的，既可以在 Solaris 也可以在 Linux 上運行。

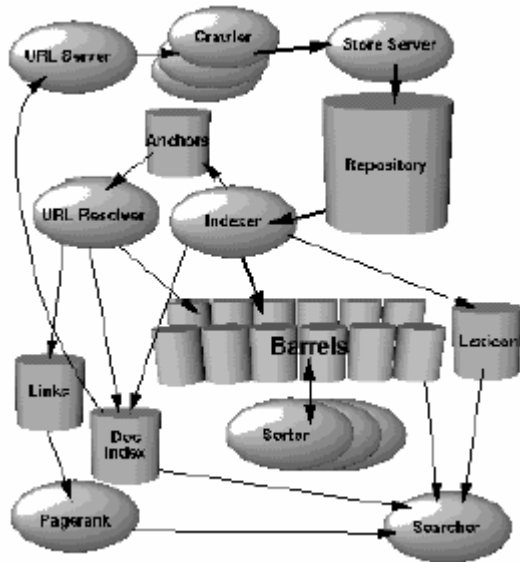


圖 1 高階的 Google 架構

Google 系統中，抓網頁（下載網頁）是由幾個分散式 crawlers 完成的。一個 URL 伺服器負責向 crawlers 提供 URL 列表。抓來的網頁交給儲存伺服器 storeserver。然後，由儲存伺服器壓縮網頁並把它們存到知識庫 repository 中。每個網頁都有一個 ID，稱作 docID，當新 URL 從網頁中分析出時，就被分配一個 docID。由索引器和排序器負責建立索引 index function。索引器從知識庫中讀取文檔，對其解壓縮和分析。每個文檔被轉換成一組詞的出現情況，稱作命中 hits。Hits 紀錄了詞，詞在文檔中的位置，最接近的字型大小，大小寫。索引器把這些 hits 分配到一組桶 barrel 中，產生經過部分排序後的索引。索引器的另一個重要功能是分析網頁中所有的鏈結，將有關的重要資訊存在鏈結描述 anchors 檔中。該檔包含了足夠的資訊，可以用來判斷每個鏈結鏈出鏈入節點的資訊，和鏈結文本。 URL 分解器 resolver 閱讀鏈結描述 anchors 檔，並把相對 URL 轉換成絕對 URL，再轉換成 docID。為鏈結描述文本編制索引，並與它所指向的 docID 關聯起來。同時建立由 docID 對組成的鏈結資料庫。用於計算所有文檔的 PageRank 值。用 docID 分類後的 barrels，送給排序器 sorter，再根據 wordID 進行分類，建立反向索引 inverted index。這個操作要恰到好處，以便幾乎不需要暫存空間。排序器還給出 docID 和偏移量列表，建立反向索引。一個叫 DumpLexicon 的程式把這個列表和由索引器產生的字典結合在一起，建立一個新的字典，供搜索器使用。這個搜索器就是利用一個 Web 伺服器，使用由 DumpLexicon 所生成的字典，利用上述反向索引以及頁面等級 PageRank 來回答用戶的提問。 4.2 主要資料結構經過優化的 Google

資料結構，能夠用較小的代價抓取大量文檔，建立索引和查詢。雖然近幾年 CPU 和輸入輸出速率迅速提高。磁片尋道仍然需要 10ms。任何時候 Google 系統的設計都盡可能地避免磁片尋道。這對資料結構的設計影響很大。

4.2.1 大檔大檔 BigFiles 是指虛擬檔生成的多檔系統，用長度是 64 位元的整型資料定址。多檔系統之間的空間分配是自動完成的。BigFiles 包也處理已分配和未分配檔描述符。由於操縱系統不能滿足我們的需要，BigFiles 也支援基本的壓縮選項。

4.2.2 知識庫 Figure 2. Repository Data Structure 知識庫包含每個網頁的全部 HTML。每個網頁用 zlib (見 RFC1950) 壓縮。壓縮技術的選擇既要考慮速度又要考慮壓縮率。我們選擇 zlib 的速度而不是壓縮率很高的 bzip。知識庫用 bzip 的壓縮率接近 4:1。而用 zlib 的壓縮率是 3:1。文檔一個挨著一個的存儲在知識庫中，首碼是 docID，長度，URL，見圖 2。訪問知識庫不需要其他的資料結構。這有助於資料一致性和升級。用其他資料結構重構系統，我們只需要修改知識庫和 crawler 錯誤列表檔。

4.2.3 檔索引檔索引保存了有關文檔的一些資訊。索引以 docID 的順序排列，定寬 ISAM (Index sequential access mode)。每條記錄包括當前檔狀態，一個指向知識庫的指標，檔校驗和，各種統計表。如果一個文檔已經被抓到，指標指向 docinfo 檔，該檔的寬度可變，包含了 URL 和標題。否則指標指向包含這個 URL 的 URL 列表。這種設計考慮到簡潔的資料結構，以及在查詢中只需要一個磁片尋道時間就能夠訪問一條記錄。還有一個檔用於把 URL 轉換成 docID。它是 URL 校驗和與相應 docID 的列表，按校驗和排序。要想知道某個 URL 的 docID，需要計算 URL 的校驗和，然後在校驗和檔中執行二進位查找，找到它的 docID。通過對這個檔進行合併，可以把一批 URL 轉換成對應的 docID。URL 分析器用這項技術把 URL 轉換成 docID。這種成批更新的模式是至關重要的，否則每個鏈結都需要一次查詢，假如用一塊磁片，322 '000' 000 個鏈結的資料集合將花費一個多月的時間。

4.2.4 詞典詞典有幾種不同的形式。和以前系統的重要不同是，詞典對記憶體的要求可以在合理的價格內。現在實現的系統，一台 256M 記憶體的機器就可以把詞典裝入到記憶體中。現在的詞典包含 14000000 辭彙(雖然一些很少用的辭彙沒有加入到詞典中)。它執行分兩部分一辭彙表(用 null 分隔的連續串)和指針的哈希表。不同的函數，辭彙表有一些輔助資訊，這超出了本文論述的範圍。

4.2.5 hit list hit list 是一篇文檔中所出現的詞的列表，包括位置，字型大小，大小寫。Hit list 占很大空間，用在正向和反向索引中。因此，它的表示形式越有效越好。我們考慮了幾種方案來編碼位置，字型大小，大小寫一簡單編碼(3 個整型數)，緊湊編碼(支援優化分配比特位元)，哈夫曼編碼。Hit 的詳細資訊見圖 3。我們的緊湊編碼每個 hit 用 2 位元組。有兩種類型 hit，特殊 hit 和普通 hit。特殊 hit 包含 URL，標題，鏈結描述文字，meta tag。普通 hit 包含其他每件事。它包括大小寫特徵位元，字型大小，12 比特用於描述詞在文檔中的位置(所有超過 4095 的位置標記為 4096)。字型大小採用相對於文檔的其他部分的相對大小表示，占 3 比特(實際只用 7 個值，因為 111 標誌是特殊 hit)。特殊 hit 由大小寫特徵位元，字型大小位元為 7 表示它是特殊 hit，用 4 比特表示特殊 hit 的類型，8 比特表示位置。對於 anchor hit 八比特位置位分出 4 比特用來表示在 anchor 中的位置，4 比特用於表明 anchor 出現的哈希表 hash of the docID。短語查詢是有限的，對某些詞沒有足夠多的 anchor。我們希望更新 anchor hit 的存儲方式，以便解決位址位元和

docIDhash 域位數不足的問題。

因為搜索時，你不會因為文檔的字型大小比別的文檔大而特殊對待它，所以採用相對字型大小。hit 表的長度存儲在 hit 前。為節省空間 hit 表長度，在正向索引中和 wordID 結合在一起，在反向索引中和 docID 結合存儲。這就限制它相應地只占 8 到 5 比特（用些技巧，可以從 wordID 中借 8bit）如果大於這些比特所能表示的長度，用溢出碼填充，其後兩位元組是真正的長度。 Figure

3. Forward and Reverse Indexes and the Lexicon

4.2.6 正向索引實際上，正向索引已經部分排序。它被存在一定數量的 barrel 中（我們用 64 個 barrels）。每個 barrel 裝著一定範圍的 wordID。如果一篇文檔中的詞落到某個 barrel，它的 docID 將被記錄到這個 barrel 中，緊跟著那些詞（文檔中所有的辭彙，還是落入該 barrel 中的辭彙）對應的 hitlist。這種模式需要稍多些的存儲空間，因為一個 docID 被用多次，但是它節省了桶數和時間，最後排序器進行索引時降低編碼的複雜度。更進一步的措施是，我們不是存儲 docID 本身，而是存儲相對於該桶最小的 docID 的差。用這種方法，未排序的 barrel 的 docID 只需 24 位元，省下 8 位元記錄 hitlist 長。

4.2.7 反向索引除了反向索引由 sorter 加工處理之外，它和正向索引包含相同的桶。對每個有效的 docID，字典包含一個指向該詞所在桶的指標。它指向由 docID 和它的相應 hitlist 組成的 doclist，這個 doclist 代表了所有包含該詞的文檔。doclist 中 docID 的順序是一個重要的問題。最簡單的解決辦法是用 doclist 排序。這種方法合併多個詞時很快。另一個可選方案是用文檔中該詞出現的次數排序。這種方法回答單詞查詢，所用時間微不足道。當多詞查詢時幾乎是從頭開始。並且當用其他 Rank 演算法改進索引時，非常困難。我們綜合了這兩種方法，建立兩組反向索引 barrel，一組 barrels 的 hitlist 只包含標題和 anchor hit，另一組 barrel 包含全部的 hitlist。我們首先查第一組索引桶，看有沒有匹配的項，然後查較大的那組桶。

4.3 抓網頁運行網路爬行機器人是一項具有挑戰性的任務。執行的性能和可靠性甚至更重要，還有一些社會焦點。網路爬行是一項非常薄弱的應用，它需要成百上千的 web 伺服器和各種功能變數名稱伺服器的參與，這些伺服器不是我們系統所能控制的。為了覆蓋幾十億的網頁，Google 擁有快速的分散式網路爬行系統。一個 URL 伺服器給若干個網路爬行機器人（我們採用 3 個）提供 URL 列表。URL 伺服器和網路爬行機器人都是用 Python 實現的。每個網路爬行機器人可以同時打開 300 個鏈結。抓取網頁必須足夠快。最快時，用 4 個網路爬行機器人每秒可以爬行 100 個網頁。速率達每秒 600K。執行的重點是找 DNS。每個網路爬行機器人有它自己的 DNS cache，所以它不必每個網頁都查 DNS。每一百個連接都有幾種不同的狀態：查 DNS，連接主機，發送請求，接收回答。這些因素使網路爬行機器人成為系統比較複雜的部分。它用非同步 IO 處理事件，若干請求佇列從一個網站到另一個網站不停的抓取網頁。運行一個鏈結到 500 多萬台伺服器的網頁爬行機器人，產生 1 千多萬登陸口，導致了大量的 Email 和電話。因為線民眾多，總有些人不知道網路爬行機器人是何物，這是他們看到的第一個網路爬行機器人。幾乎每天我們都會收到這樣的 Email “哦，你從我們的網站看了太多的網頁，你想幹什麼？”還有一些人不知道網路搜索機器人避免協定（the robots exclusion protocol），以為他們的網頁上寫著“版權所有，勿被索引”的字樣就會被保護不被索引，不必說，這樣的話很難被 web crawler 理解。因為資料量如此之大，還會遇到一些意想不到的事情。例如，我們的系統曾經企圖抓一個線上遊戲，結果抓到了遊戲中的大量垃圾資訊。解決這個問題很簡單。但是我們下載了幾千萬網頁後才發現了這

個問題。因為網頁和伺服器的種類繁多，實際上不在大部分 Internet 上運行它就測試一個網頁爬行機器人是不可能。總是有幾百個隱含的問題發生在整個 web 的一個網頁上，導致網路爬行機器人崩潰，或者更糟，導致不可預測的不正確的行為。能夠訪問大部分 Internet 的系統必須精力充沛並精心測試過。由於象 crawler 這樣大型複雜的系統總是產生這樣那樣的問題，因此花費一些資源讀這些 Email，當問題發生時解決它，是有必要的。

4.4 Web 索引分析—任何運行在整個 Web 上的分析器必須能夠處理可能包含錯誤的大型集合。範圍從 HTML 標記到標記之間幾 K 位元組的 0，非 ASCII 字元，幾百層 HTML 標記的嵌套，各種各樣令人難以想像的錯誤。為了獲得最大的速度，我們沒有採用 YACC 產生上下文無關文法 CFG 分析器，而是採用靈活的方式產生辭彙分析器，它自己配有堆疊。分析器的改進大大提高了運行速度，它的精力如此充沛完成了大量工作。把文檔裝入 barrel 建立索引—分析完一篇文檔，之後把該文檔裝入 barrel 中，用記憶體中的 hash 表—字典，每個辭彙被轉換成一個 wordID。當 hash 表字典中加入新的項時，笨拙地存入檔。一旦辭彙被轉換成 wordID，它們在當前文檔的出現就轉換成 hitlist，被寫進正向 barrel。索引階段並行的主要困難是字典需要共用。

我們採用的方法是，基本字典中有 140 萬個固定辭彙，不在基本字典中的辭彙寫入日誌，而不是共用字典。這種方法多個索引器可以並行工作，最後一個索引器只需處理一個較小的額外辭彙日誌。排序—為了建立反向索引，排序器讀取每個正向 barrel，以 wordID 排序，建立只有標題 anchor hit 的反向索引 barrel 和全文反向索引 barrel。這個過程一次只處理一個 barrel，所以只需要少量暫存空間。排序階段也是並行的，我們簡單地同時運行盡可能多的排序器，不同的排序器處理不同的桶。由於 barrel 不適合裝入主存，排序器進一步依據 wordID 和 docID 把它分成若干籃子，以便適合裝入主存。然後排序器把每個籃子裝入主存進行排序，並把它的内容寫回到短反向 barrel 和全文反向 barrel。

4.5 搜索搜索的目標是提供有效的高品質的搜索結果。多數大型商業搜索引擎好像在效率方面花費了很大力氣。因此我們的研究以搜索品質為重點，相信我們的解決方案也可以用到那些商業系統中。

Google 查詢評價過程見圖 4。

1. 分析查詢。
2. 把辭彙轉換成 wordID。
3. 在短 barrel 中查找每個辭彙 doclist 的開頭。
4. 掃描 doclist 直到找到一篇匹配所有關鍵字的文檔
5. 計算該文檔的 rank
6. 如果我們在短 barrel，並且在所有 doclist 的末尾，開始從全文 barrel 的 doclist 的開頭查找每個詞，goto 第四步
7. 如果不在任何 doclist 的結尾，返回第四步。
8. 根據 rank 排序匹配文檔，返回前 k 個。圖 4 Google 查詢評價在有限的回應時間內，一旦找到一定數量的匹配文檔，搜索引擎自動執行步驟 8。這意味著，返回的結果是子優化的。我們現在研究其他方法來解決這個問題。過去根據 PageRank 排序 hit，看來能夠改進這種狀況。

4.5.1 Ranking 系統 Google 比典型搜索引擎保存了更多的 web 資訊。每個 hitlist 包括位置，字型大小，大小寫。另外，我們還考慮了鏈結描述文字。Rank 綜合所有這些資訊是困難的。ranking

函數設計依據是沒有某個因素對 rank 影響重大。首先，考慮最簡單的情況——單個詞查詢。為了單個詞查詢中一個文檔的 rank，Google 在文檔的 hitlist 中查找該詞。Google 認為每個 hit 是幾種不同類型（標題，鏈結描述文字 anchor，URL，普通大字型大小文本，普通小字型大小文本，……）之一，每種有它自己的類型權重。類型權重建立了一個類型索引向量。Google 計算 hitlist 中每種 hit 的數量。然後每個 hit 數轉換成 count-weight。Count-weight 開始隨 hit 數線性增加，很快逐漸停止，以至於 hit 數與此不相關。我們計算 count-weight 向量和 type-weight 向量的標量積作為文檔的 IR 值。最後 IR 值結合 PageRank 作為文檔的最後 rank 對於多詞查詢，更複雜些。現在，多詞 hitlist 必須同時掃描，以便關鍵字出現在同一文檔中的權重比分別出現時高。相鄰詞的 hit 一起匹配。對每個匹配 hit 的集合計算相鄰度。相鄰度基於 hit 在文檔中的距離，分成 10 個不同的 bin 值，範圍從短語匹配到根本不相關。不僅計算每類 hit 數，而且要計算每種類型的相鄰度，每個類型相似度對，有一個類型相鄰度權 type-prox-weight。Count 轉換成 count-weight，計算 count-weight type-prox-weight 的標量積作為 IR 值。應用某種 debug mode 所有這些數和矩陣與查詢結果一起顯示出來。這些顯示有助於改進 rank 系統。

4.5.2 回饋 rank 函數有很多參數象 type-weight 和 type-prox-weight。指明這些參數的正確值有點黑色藝術 black art。為此，我們的搜索引擎有一個用戶回饋機制。值得信任的用戶可以隨意地評價返回的結果。保存回饋。然後，當修改 rank 函數時，對比以前搜索的 rank，我們可以看到修改帶來的影響。雖然不是十全十美，但是它给出了一些思路，當 rank 函數改變時對搜索結果的影響。

5 執行和結果搜索結果的品質是搜索引擎最重要的度量標準。完全用戶評價體系超出了本文的論述範圍，對於大多數搜索，我們的經驗說明 Google 的搜索結果比那些主要的商業搜索引擎好。作為一個應用 PageRank，鏈結描述文字，相鄰度的例子，圖 4 给出了 Google 搜索 bill Clinton 的結果。它說明了 Google 的一些特點。伺服器對結果進行聚類。這對過濾結果集合相當有幫助。這個查詢，相當一部分結果來自 whitehouse.gov 域，這正是我們所需要的。現在大多數商業搜索引擎不會返回任何來自 whitehouse.gov 的結果，這是相當不對的。注意第一個搜索結果沒有標題。因為它不是被抓到的。Google 是根據鏈結描述文字決定它是一個好的查詢結果。同樣地，第五個結果是一個 Email 位址，當然是不可能抓到的。也是鏈結描述文字的結果。所有這些結果品質都很高，最後檢查沒有死鏈結。因為它們中的大部分 PageRank 值較高。PageRank 百分比用紅色線條表示。沒有結果只含 Bill 沒有 Clinton 或只含 Clinton 沒有 Bill。因為詞出現的相近性非常重要。當然搜索引擎品質的真實測試包含廣泛的用戶學習或結果分析，此處篇幅有限，請讀者自己去體驗 Google，<http://google.stanford.edu/>。5.1 存儲需求除了搜索品質，Google 的設計可以隨著 Web 規模的增大而有效地增大成本。一方面有效地利用存儲空間。表 1 列出了一些統計數字的明細表和 Google 存儲的需求。由於壓縮技術的應用知識庫只需 53GB 的存儲空間。是所有要存儲資料的三分之一。按當今磁片價格，知識庫相對於有用的資料來說比較便宜。搜索引擎需要的所有資料的存儲空間大約 55GB。大多數查詢請求只需要短反向索引。檔索引應用先進的編碼和壓縮技術，一個高品質的搜索引擎可以運行在 7GB 的新 PC。

5.2 系統執行搜索引擎抓網頁和建立索引的效率非常重要。Google 的主要操作是抓網頁，索引，排序。很難測試抓全部網頁需要多少時間，因為磁片滿了，功能變數名稱伺服器崩潰，或者其他問題導致系統停止。總的來說，大約需要 9 天時間下載 26000000 網頁（包括錯誤）。然而，一旦

系統運行順利，速度非常快，下載最後 11000000 網頁只需要 63 小時，平均每天 4000000 網頁，每秒 48.5 個網頁。索引器和網路爬行機器人同步運行。索引器比網路爬行機器人快。因為我們花費了大量時間優化索引器，使它不是瓶頸。這些優化包括批量更新文檔索引，本地磁片資料結構的安排。索引器每秒處理 54 個網頁。排序器完全並行，用 4 台機器，排序的整個過程大概需要 24 小時。

5.3 搜索執行改進搜索執行不是我們研究的重點。當前版本的 Google 可以在 1 到 10 秒間回答查詢請求。時間大部分花費在 NFS 磁片 IO 上（由於磁片普遍比機器慢）。進一步說，Google 沒有做任何優化，例如查詢緩衝區，常用辭彙子索引，和其他常用的優化技術。我們傾向於通過分散式，硬體，軟體，和演算法的改進來提高 Google 的速度。我們的目標是每秒能處理幾百個請求。表 2 有幾個現在版本 Google 回應查詢時間的例子。它們說明 IO 緩衝區對再次搜索速度的影響。

6 結論 Google 設計成可伸縮的搜索引擎。主要目標是在快速發展的 World Wide Web 上提供高品質的搜索結果。Google 應用了一些技術改進搜索品質包括 PageRank，鏈結描述文字，相鄰資訊。進一步說，Google 是一個收集網頁，建立索引，執行搜索請求的完整的體系結構。

6.1 未來的工作大型 Web 搜索引擎是個複雜的系統，還有很多事情要做。我們直接的目標是提高搜索效率，覆蓋大約 100000000 個網頁。一些簡單的改進提高了效率包括請求緩衝區，巧妙地分配磁碟空間，子索引。另一個需要研究的領域是更新。我們必須有一個巧妙的演算法來決定哪些舊網頁需要重新抓取，哪些新網頁需要被抓取。這個目標已經由實現了。受需求驅動，用代理 cache 創建搜索資料庫是一個有前途的研究領域。我們計畫加一些簡單的已經被商業搜索引擎支援的特徵，例如布林算術符號，否定，填充。然而另外一些應用剛剛開始探索，例如相關回饋，聚類（Google 現在支援簡單的基於主機名的聚類）。我們還計畫支援用戶上下文（象用戶地址），結果摘要。我們正在擴大鏈結結構和鏈結文本的應用。簡單的實驗證明，通過增加用戶主頁的權重或書籤，PageRank 可以個性化。對於鏈結文本，我們正在試驗用鏈結周圍的文本加入到鏈結文本。Web 搜索引擎提供了豐富的研究課題。如此之多以至於我們不能在此一一列舉，因此在不久的將來，我們希望所做的工作不止本節提到的。

6.2 高品質搜索當今 Web 搜索引擎用戶所面臨的最大問題是搜索結果的品質。結果常常是好笑的，並且超出用戶的眼界，他們常常灰心喪氣浪費了寶貴的時間。例如，一個最流行的商業搜索引擎搜索“Bill Clilton”的結果是 the Bill Clinton Joke of the Day: April 14, 1997。Google 的設計目標是隨著 Web 的快速發展提供高品質的搜索結果，容易找到資訊。為此，Google 大量應用超文本資訊包括鏈結結構和鏈結文本。Google 還用到了相鄰性和字型大小資訊。評價搜索引擎是困難的，我們主觀地發現 Google 的搜索品質比當今商業搜索引擎高。通過 PageRank 分析鏈結結構使 Google 能夠評價網頁的品質。用鏈結文本描述鏈結所指向的網頁有助於搜索引擎返回相關的結果（某種程度上提高了品質）。最後，利用相鄰性資訊大大提高了很多搜索的相關性。

6.3 可升級的體系結構除了搜索品質，Google 設計成可升級的。空間和時間必須高效，處理整個 Web 時固定的幾個因素非常重要。實現 Google 系統，CPU、訪存、記憶體容量、磁片尋道時間、磁片吞吐量、磁片容量、網路 IO 都是瓶頸。在一些操作中，已經改進的 Google 克服了一些瓶頸。Google 的主要資料結構能夠有效利用存儲空間。進一步，網頁爬行，索引，排序已經足夠建立大部分 web 索引，共 24000000 個網頁，用時不到一星期。我們希望能在一個月內建立 100000000 網頁的索引。

6.4 研究工具 Google 不僅是高品質的搜索引擎，它還是研究工具。Google 搜集的資料已經用在許多其他論文中，提交給學術會議和許多其他方式。最近的研究，例如，提出了 Web 查詢的局限性，不需要網路就可以回答。這說明 Google 不僅是重要的研究工具，而且必不可少，應用廣泛。我們希望 Google 是全世界研究者的資源，帶動搜索引擎技術的更新換代。7 致謝 Scott Hassan and Alan Steremberg 評價了 Google 的改進。他們的才智無可替代，作者由衷地感謝他們。感謝 Hector Garcia-Molina, Rajeev Motwani, Jeff Ullman, and Terry Winograd 和全部 WebBase 開發組的支援和富有深刻見解的討論。最後感謝 IBM, Intel, Sun 和投資者的慷慨支援，為我們提供設備。這裏所描述的研究是 Stanford 綜合數位圖書館計畫的一部分，由國家科學自然基金支援，合作協議號 IRI-9411306。DARPA, NASA, Interva 研究, Stanford 數位圖書館計畫的工業合作夥伴也為這項合作協議提供了資金。參考文獻 ·

Google 的設計目標是可升級到 10 億網頁。我們的磁片和機器大概能處理這麼多網頁。系統各個部分耗費的總時間是並行的和線性的。包括網頁爬行機器人，索引器和排序器。擴展後我們認為大多數資料結構運行良好。然而 10 億網頁接近所有常用作業系統的極限(我們目前運行在 Solaris 和 Linux 上)。包括主存位址，開放檔描述符的數量，網路 socket 和帶寬，以及其他因素。我們認為當網頁數量大大超過 10 億網頁時，會大大增加系統複雜性。9.2 集中式索引體系的可升級性隨著電腦性能的提高，海量文本索引的成本比較公平。當然帶寬需求高的其他應用如視頻，越來越普遍。但是，與多媒體例如視頻相比，文本產品的成本低，因此文本仍然普遍。

Repository: 53.5 GB = 147.8 GB uncompressed

sync	length	compressed	packet
sync	length	compressed	packet

...
Packet (stored compressed in repository)

docid	ecode	url	pagelen	url	page
-------	-------	-----	---------	-----	------

圖 2 Google 系統的工作流程圖

(注：原圖來自 Sergey Brin and Lawrence Page, The Anatomy of a Large-Scale Hypertextual. Web Search Engine, 1998.http://www-db.stanford.edu/%7Ebackrub/Google.html)

①Google 使用高速的分散式爬行器(Crawler)系統中的漫遊遍曆器(Googlebot)定時地遍曆網頁，將遍曆到的網頁送到存儲伺服器(Store Server)中。

②存儲伺服器使用 zlib 格式壓縮軟體將這些網頁進行無損壓縮處理後存入資料庫 Repository 中。Repository 獲得了每個網頁的完全 Html 代碼後，對其壓縮後的網頁及 URL 進行分析，記錄下網頁長度、URL、URL 長度和網頁內容，並賦予每個網頁一個文檔號(docID)，以便當系統出現故障的時候，可以及時完整地進行網頁的資料恢復。

③索引器(Indexer)從 Repository 中讀取資料，以後做以下四步工作：

④(a)將讀取的資料解壓縮後進行分析，它將網頁中每個有意義的詞進行統計後，轉化為關鍵字(wordID)的若干索引項(Hits)，生成索引項列表，該列表包括關鍵字、關鍵字的位置、關鍵字的大小和大小寫狀態等。索引項列表被存入到資料桶(Barrels)中，並生成以文檔號(docID)部分排序的順排檔索引。

索引項根據其重要程度分為兩種：當索引項中的關鍵字出現在 URL、標題、錨文本(Anchor Text)

和標籤中時，表示該索引項比較重要，稱為特殊索引項(Fancy Hits)；其餘情況則稱為普通索引項(Plain Hits)。在系統中每個 Hit 用兩個位元組(byte)存儲結構表示：特殊索引項用 1 位元(bit)表示大小寫，用二進位碼 111(占 3 位元)表示是特殊索引項，其餘 12 位元有 4 位元表示特殊索引項的類型(即 hit 是出現在 URL、標題、鏈結結點還是標籤中)，剩下 8 位表示 hit 在網頁中的具體位置；普通索引項是用 1 位元表示大小寫，3 位元表示字體大小，其餘 12 位元表示在網頁中的具體位置。

順排檔索引和 Hit 的存儲結構如圖 3 所示。

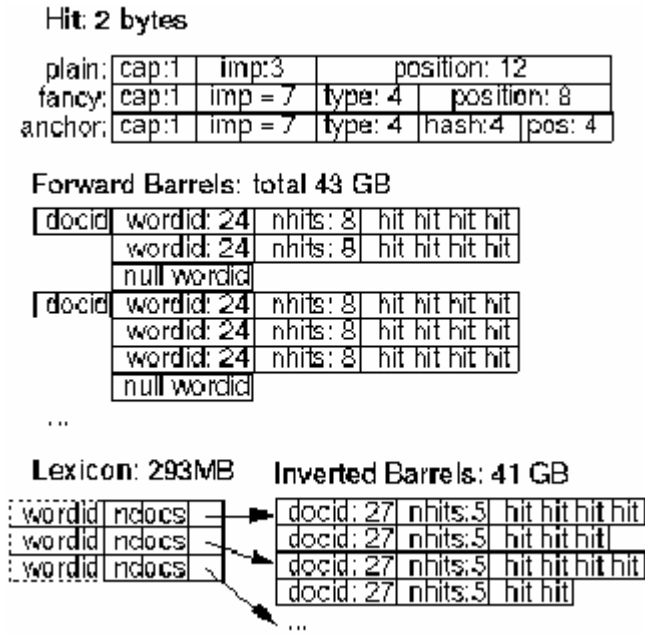


圖 3 順排檔索引和 Hit 的存儲結構

值得注意的是，當特殊索引項來自 Anchor Text 時，特殊索引項用來表示位置的資訊（8 位元）將分為兩部分：4 位表示 Anchor Text 出現的具體位置，另 4 位則用來與表示 Anchor Text 所鏈結網頁的 docID 相連接，這個 docID 是由 URL Resolver 經過轉化存入順排檔索引的。

- (b)索引器除了對網頁中有意義的詞進行分析外，還分析網頁的所有超文本鏈結，將其 Anchor Text、URL 指向等關鍵資訊存入到 Anchor 文檔庫中。
- (c)索引器生成一個索引詞表(Lexicon)，它包括兩個部分：關鍵字的列表和指標列表，用於倒排檔文檔相連接(如圖 3 所示)。
- (d)索引器還將分析過的網頁編排成一個與 Repository 相連接的文檔索引(Document Index)，並記錄下網頁的 URL 和標題，以便可以準確查找出在 Repository 中存儲的原網頁內容。而且把沒有分析的網頁傳給 URL Server，以便在下一次工作流程中進行索引分析。
- ⑤URL 分析器（URL Resolver）讀取 Anchor 文檔中的資訊，然後做⑥中的工作。
- ⑥(a)將其錨文本(Anchor Text)所指向的 URL 轉換成網頁的 docID；(b)將該 docID 與原網頁的 docID 形成“鏈結對”，存入 Link 資料庫中；(c)將 Anchor Text 指向的網頁的 docID 與順排檔特殊索引項 Anchor Hits 相連接。

⑦資料庫 Link 記錄了網頁的鏈結關係，用來計算網頁的 PageRank 值。

⑧文檔索引(Document Index)把沒有進行索引分析的網頁傳遞給 URL Server，URL Server 則向 Crawler 提供待遍曆的 URL，這樣，這些未被索引的網頁在下一工作流程中將被索引分析。

⑨排序器 (Sorter) 對資料桶(Barrels)的順排檔索引重新進行排序，生成以關鍵字(wordID)為索引的倒排檔索引。倒排檔索引結構如圖 4 所示：

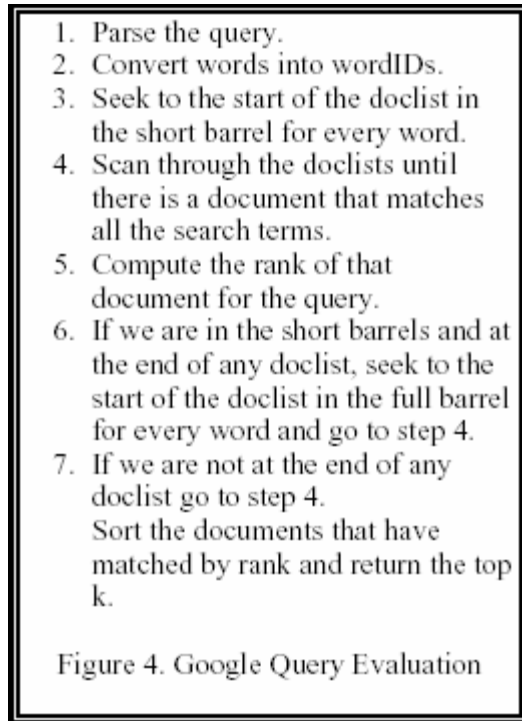


圖 4 倒排檔索引結構

⑩將生成的倒排檔索引與先前由索引器產生的索引詞表(Lexicon)相連接產生一個新的索引詞表供搜索器(Searcher)使用。搜索器的功能是由網頁伺服器實現的，根據新產生的索引詞表結合上述的文檔索引(Document Index)和 Link 資料庫計算的網頁 PageRank 值來匹配檢索。

在執行檢索時，Google 通常遵循以下步驟（以下所指的是單個檢索詞的情況）：

- (1)將檢索詞轉化成相應的 wordID；
- (2)利用 Lexicon，檢索出包含該 wordID 的網頁的 docID；
- (3)根據與 Lexicon 相連的倒排檔索引，分析各網頁中的相關索引項的情況，計算各網頁和檢索詞的匹配程度，必要時調用順排檔索引；
- (4)根據各網頁的匹配程度，結合根據 Link 產生的相應網頁的 PageRank 情況，對檢索結果進行排序；
- (5)調用 Document Index 中的 docID 及其相應的 URL，將排序結果生成檢索結果的最終列表，提供給檢索用戶。

用戶檢索包含多個檢索詞的情況與以上單個檢索詞的情況類似：先做單個檢索詞的檢索，然後根據檢索式中檢索符號的要求進行必要的布林操作或其他操作。