УДК 004.424

С.С. СИНЕЛЬНИКОВ, В.А. РЕЗНИКОВ

Донецкий государственный институт искусственного интеллекта, Украина

РАЗРАБОТКА АЛГОРИТМА «УСКОРЕННОГО БИНАРНОГО ПОИСКА» В ОТСОРТИРОВАННОМ МАССИВЕ НА ЯЗЫКЕ С++

Рассмотрены методы поиска данных в отсортированном массиве, улучшен бинарный поиск.

информационно-поисковые системы, поиск данных, «бинарный поиск», «ускоренный бинарный поиск», «бинарный спуск», численные методы, массив, интерполяционный метод

Введение

Эффективность файловых систем, баз данных, систем искусственного интеллекта и т.д. во многом зависит от оперативности поиска необходимой информации. Причем с повышением уровня технического обеспечения информационно-поисковых систем важность задачи поиска нисколько не снижается, учитывая стремление разработчиков указанных систем в полной мере использовать возможности современных компьютеров.

Известно, что задача поиска [1 – 3] заключается в нахождении индекса i элемента массива f по заранее известному значению элемента $= key$, т.е. поиск индекса i, при котором $f[i] = key$.

В настоящее время известными методами поиска данных в отсортированном массиве из N элементов являются бинарный поиск [1, 2], интерполяционный метод [1] и с применением бинарных деревьев [1], а скорость поиска для каждого из методов соответственно порядка $logN$, $loglogN$, $logN$. Также применяются методы, основанные на численных методах [4 – 7].

Из всех перечисленных методов наиболее распространенным является метод бинарного поиска, который не требует построения дополнительных структур, характеризуется довольно высокой скоростью и достаточно просто реализуется в виде соответствующего программного обеспечения. Однако именно по теоретической сложности этот метод уступает интерполяционному методу, и поэтому большинство работ, посвященных усовершенствованию и развитию данного метода, направлены, в первую очередь, на повышение его быстродействия.

В данной работе предложен один из возможных способов усовершенствования бинарного поиска, названный алгоритмом «ускоренного бинарного поиска».

1. Анализ бинарного метода поиска данных

Эффективность метода зависит не только от алгоритма, но и от его реализации, задачи, к которой он применяется (специфики предметной области), носителя информации, а также многих других факторов связанных с аппаратным обеспечением и ОС.

В данной работе будет рассмотрена реализация алгоритмов на языке С++ [8 – 10], как наиболее популярного и эффективного языка программирования.

Бинарный метод возможно реализовать в соответствии с алгоритмом, представленном на рис. 1.

Входными данными являются – отсортированный массив *m* (индексация элементов начинается с нуля), и количество элементов *size*. Переменные *low* и *high* – соответствуют нижнему и верхнему преде-

лу индексов массива, *mid* – индекс среднего элемента, *key* – искомый элемент.

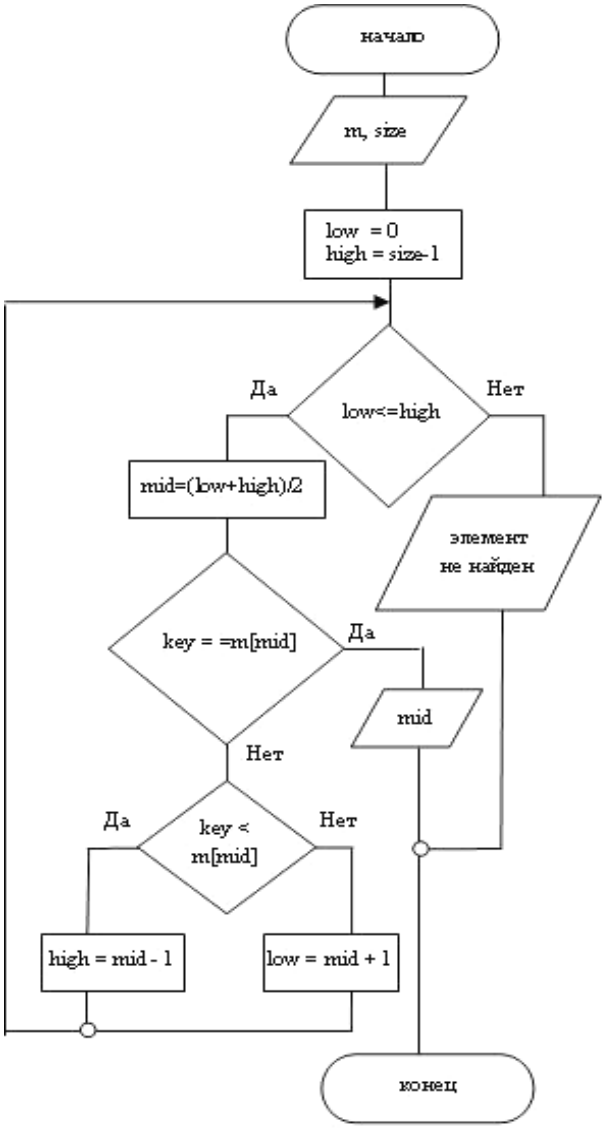

Рис. 1. Блок-схема работы первого алгоритма бинарного метода поиска

Данный алгоритм не учитывает важной статистической особенности – при большом количестве элементов в массиве мы не получим сразу искомый элемент в среднем. По статистическим расчетам [11 – 12], вероятность попадания на искомый элемент при первом сравнении (1/*size*) будет ниже вероятности попадания на элемент больший или меньший среднего (0,5). В соответствии с этими рассуждениями, логичнее сначала проверять средний элемент на больше или меньше, а не на равенство.

Таким образом, мы приходим ко второму алгоритму бинарного метода поиска, блок-схема которого представлена на рис. 2. Данный подход имеет меньшее количество сравнений, чем алгоритм 1, что обеспечивает большую скорость выполнения задачи поиска.

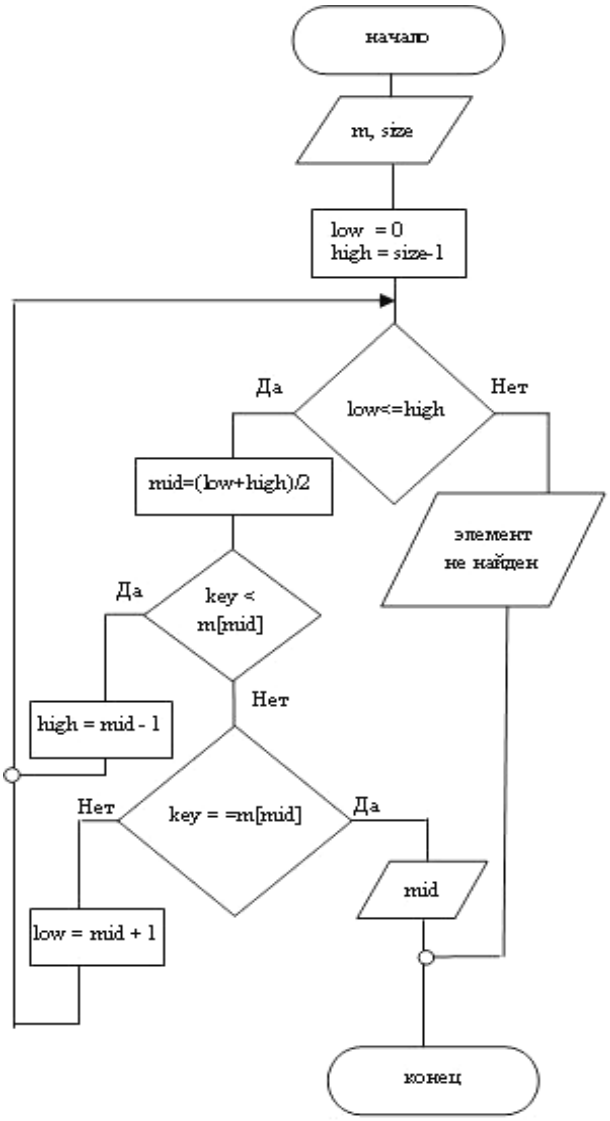

Рис. 2. Блок-схема работы второго алгоритма бинарного метода поиска

Анализ второго алгоритма показывает, что количество сравнений уменьшается, причем дополнительных каких-либо вычислительных затрат не производится.

Рассмотрим детальней второй алгоритм и проанализируем, за счет чего скорость поиска может быть увеличена. На вычислительную сложность данного алгоритма влияют все операции, которые

производятся в нем. В нашем случае вычисление среднего элемента может быть ускорено путем замены деления на сдвиг, т.е. вместо *mid*=(*low*+*high*)/2 можно использовать в реализации программы *mid*=(*low*+*high*)>>1. Данная операция выполняется быстрее любым процессором, чем операция деления на два.

Анализируя выражение вычисления среднего элемента *mid*=(*low*+*high*)/2, заметим, что его формирует операция сложения верхнего (*low*) и нижнего (*high*) индексов. Возникает естественный вопрос – можно ли избавиться или заменить операцию сложения на более быструю.

Очевидно, что операцию сложения можно не выполнять в том случае, если *low*=0. Как добиться такого условия? Проанализировав второй алгоритм, можно заметить, что перед циклом данное условие выполняется, чем необходимо воспользоваться и выполнить «бинарный спуск».

2. Процедура «бинарного спуска»

Под «бинарным спуском» будем понимать процедуру, выполняющую уменьшение верхнего индекса *high* в два раза до тех пор, пока искомый элемент *key* больше элемента *m*[*high*/2]. Блок-схема алгоритма «бинарного спуска» показана на рис. 3.

Следует отметить, что после выполнения процедуры «бинарного спуска» мы можем занести в *low*=*high*/2 (это видно из того, что условие *key* < *m*[*high*/2] не выполнилось, и, следовательно, *key* >= *m*[*high*/2]). Такой подход обеспечивает еще большую скорость работы метода за счет сужения диапазона поиска.

3. Метод «бинарного спуска с разделенным бинарным поиском»

Недостатком процедуры «бинарного спуска» есть, то, что она не работает, если *key* < *m*[0]. Для устранения этого недостатка модифицируем ее, добавив дополнительную проверку на выход *key* за приделы массива, и получим алгоритм 3 (рис. 4).

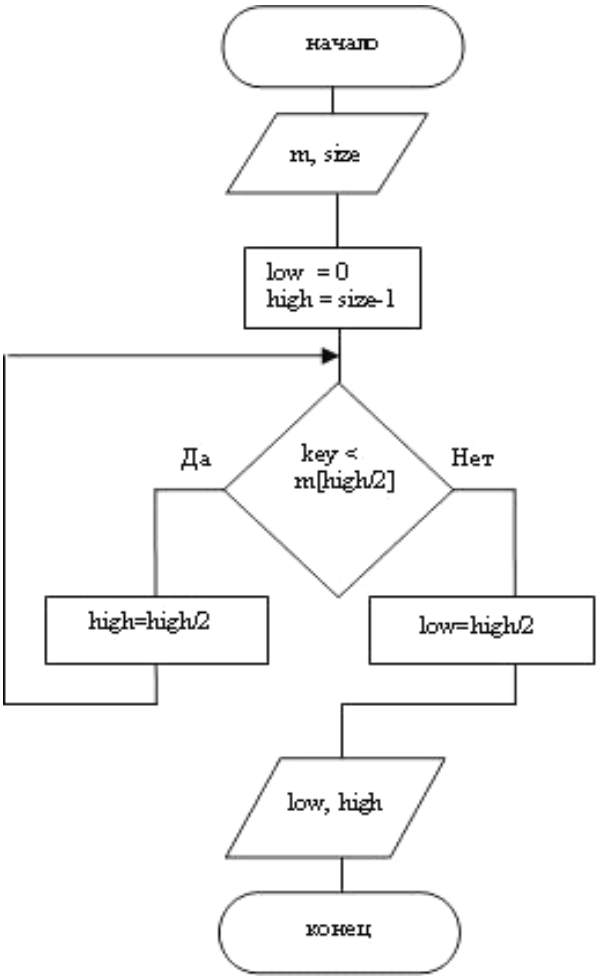

Рис. 3. Блок-схема описания работы процедуры «бинарного спуска»

Первая проверка *key* < *m*[*high*/2] обусловлена тем, что только при ее выполнении следует проверять условие *key* < *m*[*low*].

Таким образом, получаем алгоритм «бинарного спуска и бинарного поиска с разделением», имеющий большую скорость работы по сравнению с предыдущими алгоритмами.

Скорость возросла за счет уменьшения количества операций сложения. Заметим дополнительную положительную сторону данного алгоритма – если элемента нет в массиве, то данный алгоритм быстрее узнает это, чем обычный алгоритм бинарного поиска.

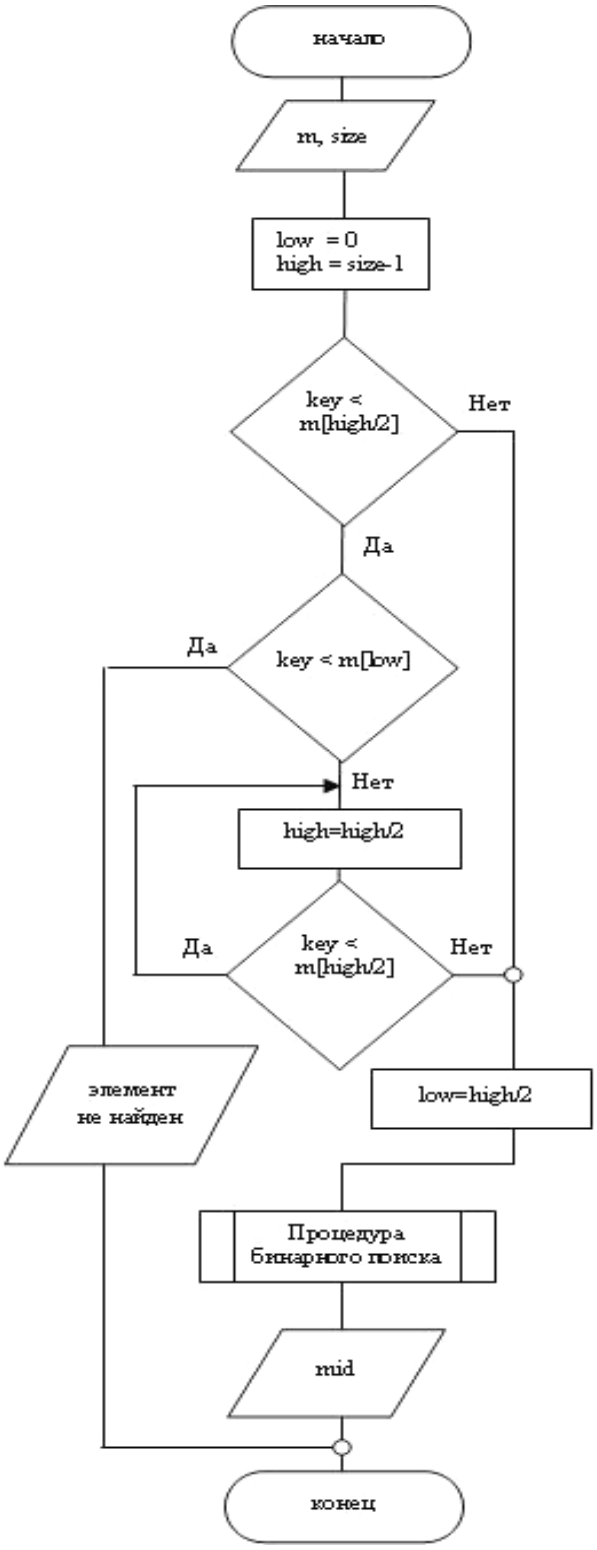

Рис. 4. Блок-схема работы алгоритма «бинарного спуска и бинарного поиска с разделением»

4. Реализация метода «бинарного спуска» на C++

«Бинарный спуск» применим, когда нижний индекс массива *low*=0. Возможно ли постоянное достижение этого условия? Да, если переназначить индексацию массива.

Некоторые языки программирования, такие как C++, позволяют это делать. Для этого необходимо к имени массива (указателю) добавить *low*, после чего массив начнется с элемента с индексом *low* и при этом его индексация будет производиться с нуля. Но в таком случае необходимо выполнять ряд дополнительных операций – $low = 0$, пересчитывать $high = high - low$, запоминать адрес первого элемента массива (*pFst*), вычислять искомый индекс при возврате – ($m–pFst+low$), что приводит к дополнительным расходам. Алгоритм «бинарного спуска» представлен на рис. 5.

5. Реализация метода «бинарного спуска и бинарного поиска с объединением» на C++

Тестирование алгоритма показало, что «бинарный спуск» медленнее по времени и больше выполняет сравнений, чем бинарный поиск, поэтому далее предлагается вариант объединения этих двух алгоритмов с целью получения наибольшей скорости выполнения по времени и наименьшего количества сравнений элементов массива, как наиболее важной составляющей скорости выполнения программы.

Алгоритм «бинарного спуска и бинарного поиска с объединением» представлен на рис. 6. В данном алгоритме после процедуры бинарного спуска выполняется бинарный поиск, после чего выше описанные действия повторяются.

Полученный алгоритм показал свою эффективность как по скорости выполнения, так и по количеству выполненных операций сравнения относительно бинарного поиска.

6. Реализация метода «ускоренного бинарного поиска» на C++

Опытным путем было выявлено, что при размере массива до 28 – 50 элементов эффект от

применения процедуры «бинарного спуска» падает.

В соответствии с этим для алгоритма «бинарного спуска и бинарного поиска с объединением» была введена проверка (*high* < 28), при срабатывании которой выполнялся обычный алгоритм бинарного поиска.

Полученный алгоритм назовем методом «ускоренного бинарного поиска».

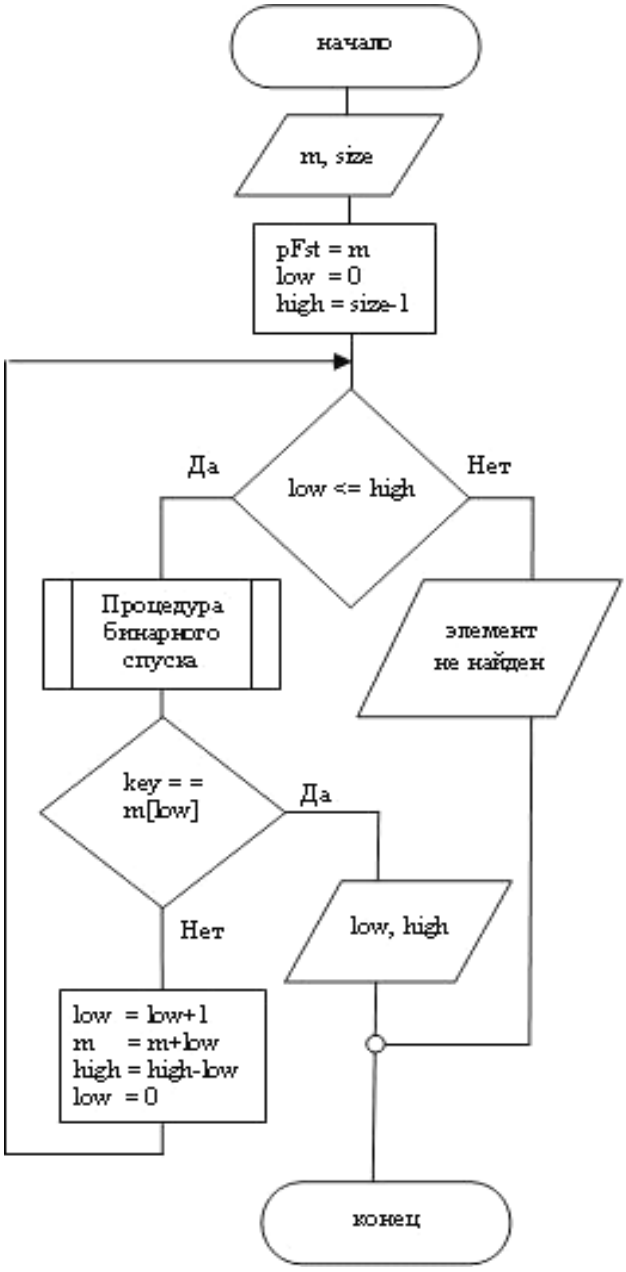

Рис. 5. Блок-схема работы алгоритма «бинарного спуска»

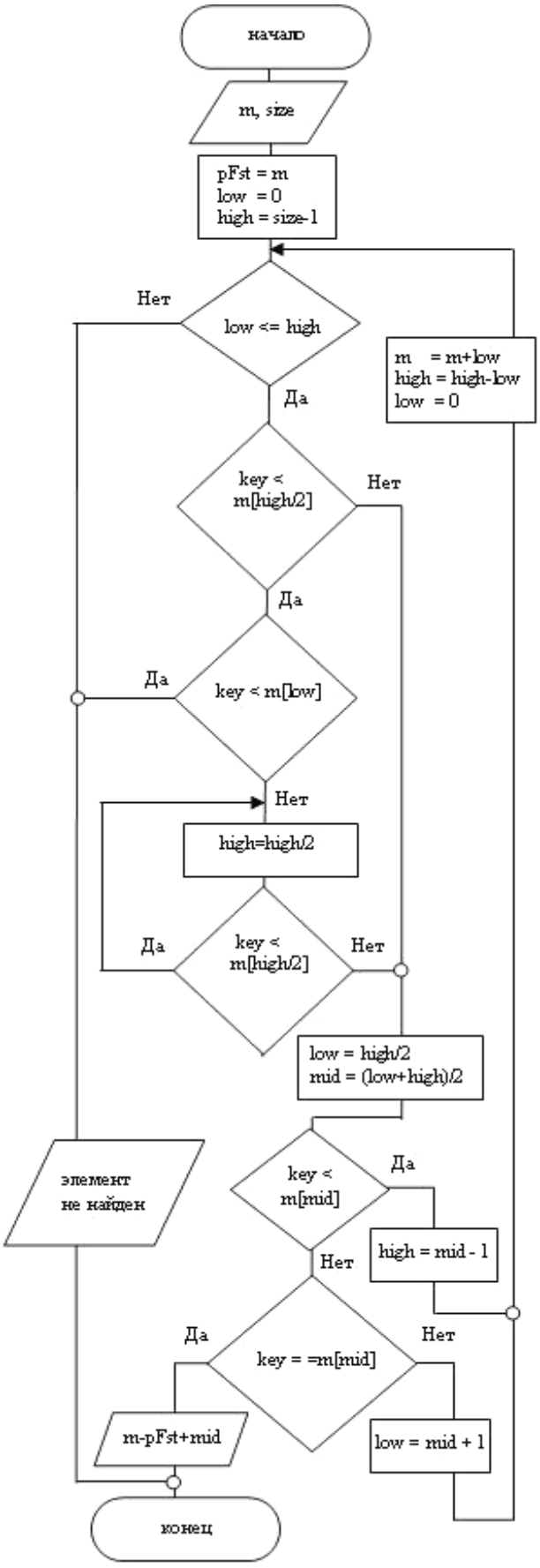

Рис. 6. Блок-схема работы алгоритма «бинарного спуска и бинарного поиска с объединением»

7. Практический анализ алгоритмов «бинарного поиска» и «ускоренного бинарного поиска»

Проведя ряд тестов по сравнению алгоритмов «бинарного поиска» и «ускоренного бинарного поиска», были получены результаты, представленные в табл. 1 – 3.

Таблица 1

Время работы «ускоренного бинарного поиска» и «бинарного поиска» для данных типа __int32

Количество элементов	Время работы алгоритма (сек)	
	«бинарный поиск»	«ускоренный бинарный поиск»
500.000	0,109	0,110 (+0,9%)
1.000.000	0,250	0,234 (–6,6%)
10.000.000	2,922	2,719 (–6,9%)
40.000.000	12,5	11,5 (–8,0%)

Таблица 2

Время работы «ускоренного бинарного поиска» и «бинарного поиска» для данных типа __int64

Количество элементов	Время работы алгоритма (сек)	
	«бинарный поиск»	«ускоренный бинарный поиск»
500.000	0,172	0,172 (–0,0%)
1.000.000	0,375	0,360 (–4,0%)
10.000.000	4,406	4,219 (–4,2%)
40.000.000	19,1	18,2 (–4,7%)

Таблица 3

Результаты тестирования алгоритмов для данных типа __int32 – среднее количество сравнений и прирост сравнений в процентах относительно «бинарного поиска»

Количество элементов	Количество сравнений	
	«бинарный поиск»	«ускоренный бинарный поиск»
10	5,1	5,5 (+7,8%)
28	6,64	6,67 (+0,4%)
50	7,78	7,74 (–0,5%)
1.000	13,85	13,34 (–3,6%)
1.000.000	28,85	26,61 (–7,7%)
10.000.000	34,03	31,20 (–8,3%)

Тест проводился по двум направлениям – для типов данных __int32 и __int64, как наиболее часто употребляемых, и по количеству элементов в массиве. В тестах показаны основные показатели алгоритмов – время выполнения и количество сделанных сравнений элементов массива и ключа. Тест проводился на компьютере со следующими характеристиками: ОС – MS Windows XP sp2, CPU – AMD Athlon XP 2000+ 1.67GHz, RAM – 496Mb. При запуске и во время работы тестирующей программы никакие другие приложения и процессы не выполнялись.

Проанализировав полученные результаты, можно сказать, что метод «ускоренного бинарного поиска» работает быстрее, чем алгоритм «бинарного поиска» для обоих типов данных при количестве элементов массива более 500000. Причем чем больше элементов, тем быстрее метод «ускоренного бинарного поиска». Из табл. 3 видно, что метод «ускоренного бинарного поиска» следует применять, если массив имеет размер более 28 элементов.

Отметим, что количество прямых сравнений элементов массива уменьшается примерно на 8–9%, что достаточно важно. Если операция сравнения ресурсоемкая и требует большого времени выполнения, то преимущество этого алгоритма станет еще большим.

Таким образом, применяя метод «ускоренного бинарного поиска», можно получить прирост производительности порядка 7–9% для данных типа __int32 и 4-5% для __int64.

Также заметим, что, если элемента в массиве нет, то алгоритм «ускоренного бинарного поиска» быстрее об этом «узнает», что является преимуществом относительно метода «бинарного поиска».

Заключение

В данной работе был проведен анализ бинарного метода поиска. Выявлены его недостатки и предложены улучшения.

Предложен и проанализирован алгоритм «бинарного спуска», на основе которого были реализо-

ваны алгоритмы комбинированные с «бинарным поиском», создан алгоритм «ускоренного бинарного поиска» на языке С++.

Выявлено, что «процедуру бинарного спуска» можно применять для любого алгоритма поиска в начале его работы, что является полезным усовершенствованием для всех методов поиска.

Проведен сравнительный анализ методов «бинарного поиска» и «ускоренного бинарного поиска», который показал, что метод «ускоренного бинарного поиска» при определенном пороговом размере массива работает быстрее алгоритма «бинарного поиска». Также алгоритм «ускоренного бинарного поиска» хорош тем, что при условии, если элемента в массиве нет, то он быстрее это определяет, чем «бинарный поиск», что является преимуществом относительно метода «бинарного поиска».

Показано, что применение метода «ускоренного бинарного поиска» позволяет: уменьшить количество прямых сравнений элементов массива относительно «бинарного поиска» на 8–9%; увеличить скорость поиска данных на 1–9% в большинстве программных комплексов, использующих поиск данных (например, сервере баз данных, файловых системах), что говорит о высокой практической значимости полученных результатов.

Литература

1. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Ш. Алгоритмы: построение и анализ. 2-е изд. – М. : Вильямс, 2005. – 1296 с.

2. Кнут Д. Искусство программирования. 3-е изд. – М. : Вильямс, 2005. – 720 с.

3. Седжвик Р. Фундаментальные алгоритмы на С++: Ч. 1-4: Анализ, структуры данных, сортировка, поиск. – К.: Diasoft, 2001. – 687 с.

4. Березин И.С., Жидков Н.П. Методы вычислений. – М.: Физматгиз, 1962. – Т.1. – 632 с.

5. Вержбицкий В.М. Основы численных методов. – М.: Высш. шк., 2002. – 840 с.

6. Волков Е.А. Численные методы. – М.: Наука, 1982. – 254 с.

7. Калиткин Н.Н. Численные методы. – М.: Наука, 1978. – 512 с.

8. Шилдт Г. С++: Руководство для начинающих. Пер. с англ., 2-е изд. – М.: Вильямс, 2005. – 1296 с.

9. Ален И. Голуб. Правила программирования на С и С++. – М.: Вильямс, 2001. – 241 с.

10. Страуструп Б. Язык программирования С++. – М.: Бином, 2005. – 1098 с.

11. Колмогоров А.Н. Основные понятия теории вероятностей. – М.: Наука, 1974. – 120 с.

12. Гмурман В.Е. Теория вероятности и математическая статистика. – М.: Высш. шк., 2005. – 479 с.

Поступила в редакцию 15.10.2007

Рецензент: канд. техн. наук С.А. Поливцев, Донецкий государственный институт проблем искусственного интеллекта, Донецк.