Агеева 110Б-21

**Теория IX курсового проекта.**

**I. Двоичный поиск. Оценка эффективности метода.**

Двоичный поиск заключается в том, что на каждом шаге множество объектов делится на две части и в работе остаётся та часть множества, где находится искомый объект. Или же, в зависимости от постановки задачи, мы можем остановить процесс, когда мы получим первый или же последний индекс вхождения элемента. Последнее условие — это левосторонний/правосторонний двоичный поиск.

*Идея поиска* заключается в том, чтобы брать элемент посередине, между границами, и сравнивать его с искомым. Если искомое больше (в случае правостороннего — не меньше), чем элемент сравнения, то сужаем область поиска так, чтобы новая левая граница была равна индексу середины предыдущей области. В противном случае присваиваем это значение правой границе. Проделываем эту процедуру до тех пор, пока правая граница больше левой более чем на 1.

Выполняется на упорядоченном одномерном массиве. Производит самый быстрый поиск при таких условиях.  
Максимальное количество сравнений(проходов) log2n. Работает следующим образом: смотрим середину первоначального интервала - больше, меньше, равна ли искомому значению? Если равна – то искомый элемент найден, если меньше, то делаем новый интервал, который начинается с середины предыдущего интервала и заканчивается той же верхней границей предыдущего, т.е. это вторая половина исходного интервала, далее вся процедура повторяется. Если середина - больше искомого значения, то новый интервал будет первой половиной исходного интервала. И так итеративно сужаем наш интервал до тех пор, пока искомый элемент не будет найден или не дойдем до пустого интервала.

while((low <= high)&&(!found)) {  
mid=(low+high)/2;  
if(keys[mid] == key) found = 1;  
else if(key > keys[mid]) low = mid + 1;  
else high = mid - 1;  
}  
if (found==1) return mid;  
if(found==0) {  
printf("Record not found!\n");  
return -1;  
}

В случае, когда первое значение mid совпадает с ключом, тогда считается, что алгоритм выполнился за свое лучшее время O(1). В среднем и худшем случае время работы алгоритма составляет O(logn), что значительно быстрее, чем у линейного поиска, требующего линейного времени.

Бинарный алгоритм поиска применяется к отсортированному массиву для поиска элемента. Бинарный алгоритм поиска применяется к отсортированному массиву для поиска элемента. Поиск начинается со сравнения целевого элемента со средним элементом массива.

В случае, если целевой элемент меньше среднего элемента (учитывая, что массив следует по возрастанию) массива, вторая половина массива отбрасывается, и поиск продолжается путем деления первой половины.

**Плюсы и минусы бинарного поиска на C**

**Преимущества:**

* Довольно простой алгоритм
* Намного быстрее по сравнению с линейным поиском. Линейный поиск требует N / 2 и N сравнений для средних и наихудших сценариев. Двоичный поиск просто требует в общей сложности log2 (N) и log 2 (N) сравнений, соответственно, для среднего и наихудшего сценариев. Проще говоря, линейный поиск в среднем требует 500 000 сравнений для набора миллионов элементов. Бинарный поиск, с другой стороны, требует всего 20 сравнений.
* Часто доступно как уже реализованная библиотечная процедура

**Недостатки:**

* Сложнее, чем линейный поиск
* Большая потеря эффективности, если список не поддерживает произвольный доступ
* Работает только для отсортированных и сохраненных списков

**II Модульное многоязычие. Директива extern.**

Если некоторый объект объявлен внутри блока, то он видим в этом блоке, и во всех внутренних блоках. Если объект объявлен на внешнем уровне, то он видим от точки его объявления до конца данного исходного файла.  
Объект может быть сделан глобально видимым с помощью соответствующих объявлений во всех исходных файлах, образующих программу.  
Спецификатор класса памяти в объявлении переменной может быть auto, register, static или extern. Если класс памяти не указан, то он определяется по умолчанию из контекста объявления.  
Переменная, объявленная локально с классом памяти extern, является ссылкой на переменную с тем же самым именем, определенную глобально в одном из исходных файлов программы. Цель такого объявления состоит в том, чтобы сделать определение переменной глобального уровня видимым внутри блока.  
Пример:  
/\* объявления переменной i, являющейся именем внешнего  
массива длинных целых чисел, на локальном уровне \*/  
/\* исходный файл file1.c \*/  
main()  
{ ...  
}  
fun1()  
{ extern long i[]; ...  
}  
/\* исходный файл file2.c \*/  
long i[MAX]={0};  
fun2()  
{ ...  
}  
fun3()  
{ ...  
}  
Объявление переменной i[] как extern в приведенном примере делает ее видимой внутри функции fun1. Определение этой переменной находится в файле file2.c на глобальном уровне и должно быть только одно, в то время как объявлений с классом памяти extern может быть несколько.  
Объявление с классом памяти extern требуется при необходимости использовать переменную, описанную в текущем исходном файле, но ниже по тексту программы, т.е. до выполнения ее глобального определения. Следующий пример иллюстрирует такое использование переменной с именем st.  
Пример:  
main()  
{ extern int st[]; ...  
}  
static int st[MAX]={0};  
fun1()  
{ ...  
}  
Объявление переменной со спецификатором extern информирует компилятор о том, что память для переменной выделять не требуется, так как это выполнено где-то в другом месте программы.