Задача 1

Полностью объяснить задачу из второго задания № 15. Под «полностью объяснить» подразумевается все — от свойств используемых структур данных до строчек кода.

Формат сдачи — предоставленный исходный код плюс устная беседа по нему.

Задача 2

Разобрать алгоритм бинарного поиска с определением ближайших узлов (см. ниже). Оценить асимптотическую сложность алгоритма.

Формат сдачи — названная асимптотическая сложность (как «О большое от чего-то») плюс устная беседа по алгоритму.

Задача 3

На базе любой удобной задачи второго задания реализовать структуру данных и интерфейс к ней (указаны ниже). Программа при запуске должна читать команды из файла commands.txt, выполнять их, печатать результат на экран и завершаться. Никаких интерактивных действий от пользователя не предполагается.

Структура данных: бинарное дерево

Тип данных: строка не более 15 символов

Состав инструкций в commands.txt:

INSERT A — добавить элемент со значением A, ничего не печатать. Если такой элемент уже есть — добавить еще одну копию.

FIND A — найти элемент со значением A. Если элемент найден, напечатать FOUND. Если не найден — напечатать NOT FOUND.

DELETE A — найти и удалить элемент со значением A. Если элемента нет — ничего не делать. Если элементов несколько — удалить одну из копий.

DELETE_ALL A — найти и удалить все вхождения элемента со значением A.

Пример:

Инструкции в файле:

INSERT alice

INSERT alice

INSERT alice

DELETE alice

FIND alice

DELETE_ALL alice

FIND alice

Вывод программы:

FOUND (так как две alice еще остались)

NOT FOUND (так как теперь все alice удалены)

Формат сдачи — предоставленный исходный код плюс устная беседа по нему.

Быстрый поиск с определением ближайших узлов

В ряде случаев (в частности, в задачах интерполяции) приходится выяснять, где по отношению к заданному упорядоченному массиву действительных чисел располагается заданное действительное число. В отличие от поиска в массиве целых чисел, заданное число в этом случае чаще всего не совпадает ни с одним из чисел массива, и требуется найти номера элементов, между которыми это число заключено.

Задача ставится так. Дан упорядоченный массив действительных чисел array размерности n, проверяемое значение value и начальное приближение узла old. Требуется найти номер узла res массива array, такой, что array[res]<=value<array[res+1]

Алгоритм работает следующим образом.

1. Определяется, лежит ли проверяемое value за пределами массива array. В случае value<array[0] возвращается -1, в случае value>array[n-1] возвращается n-1.

2. Иначе проверяется: если значение old лежит за пределами индексов массива (т.е. old<0 или old>=n, то переходим к обычному бинарному поиску, установив левую границу left=0, правую right=n-1.

3. Иначе переходим к выяснению границ поиска. Устанавливается left=right=old, inc=1 -- инкремент поиска.

4. Проверяется неравенство value>=array[old]. При его выполнении переходим к следующему пункту (5), иначе к пункту (7).

5. Правая граница поиска отводится дальше: right=right+inc. Если right>=n-1, то устанавливается right=n-1 и переходим к бинарному поиску.

6. Проверяется value>=array[right]. Если это неравенство выполняется, то левая граница перемещается на место правой: left=right, inc умножается на 2, и переходим назад на (5). Иначе переходим к бинарному поиску.

7. Левая граница отводится: left=left-inc. Если left<=0, то устанавливаем left=0 и переходим к бинарному поиску.

8. Проверяется value<array[left]. При выполнении правая граница перемещается на место левой: right=left, inc умножается на 2, переходим к пункту (7). Иначе к бинарному поиску.

9. Проводится бинарный поиск в массиве с ограничением индексов left и right. При этом каждый раз интервал сокращается примерно в 2 раза (в зависимости от четности разности), пока разность между left и right не достигнет 1. После этого возвращаем left как результат, одновременно присваивая old=left.

Пример реализации:

```
/* Search within a real ordered array.
   Parameters:
   value -- the sample,
   old -- previous result,
   array,n -- the array and its dimension.
   Returns: Left node of the array segment matching the sample.
   In case the sample is out of the array, returns -1 (less than
   left boundary) or (n-1) (more than the right boundary).
*/
```

```
int fbin_search(float value,int *old,float *array,int n) {
  register int left,right;
  /* проверка позиции за пределами массива */
  if(value < array[0]) return(-1);
  if(value >= array[n-1]) return(n-1);
  /* процесс расширения области поиска. Вначале проверяется валидность
     начального приближения */
  if(*old>=0 && *old<n-1) {
    register int inc=1;
    left = right = *old;
    if(value < array[*old]) {
      /* область расширять влево */
      while(1) {
        left -= inc;
        if(left <= 0) {left=0;break;}
        if(array[left] <= value) break;
        right=left; inc<<=1;
      }
    }
    else {
      /* область расширять вправо */
      while(1) {
        right += inc;
        if(right >= n-1) {right=n-1;break;}
        if(array[right] > value) break;
        left=right; inc<<=1;
      }
    }
  }
  /* начальное приближение плохое -
  за область поиска принимается весь массив */
  else {left=0;right=n-1;}
  /* ниже алгоритм бинарного поиска требуемого интервала */
  while(left<right-1) {
    register int node=(left+right)>>1;
    if(value>=array[node]) left=node;
```

```
    else right=node;
  }
  /* возвращаем найденную левую границу,
  обновив старое значение результата */
  return(*old=left);
}
```

Замечание: для успешной работы алгоритма целесообразно при первичном запуске положить *old=-1 или другое число, гарантирующее бинарный поиск на первом проходе.