acm.mipt.ru

олимпиады по программированию на Физтехе

Раздел «Язык Си» . CoffeeSort :

Разные сортировки.

Простые сотрировки

Часто возникают задачи поиска в массиве определенных значений или наличия тех или иных элементов.

Данные же, оказываются записанными случайным образом. Из-за этого поиск может занять продолжительное время.

Гораздо удобнее искать если данные отсортированны по определенному критерию.

Сортировка "пузырьком".

Рассмотрим массив целых чисел из 6 элементов.

```
3 58 -9 7 12 33
```

Для того, чтобы найти минимальный или максимальный элемент, нужно просмотреть весь массив. А поиск среднего элемента (относительно которого почти половина других элемнтов меньше по значению, а другая половина половина – больше) вообще оказывется несколько сложнее ожидаемого.

Попробуем применить самый простой алгоритм сортировки к нашему массиву: будем брать два соседних элемента, сравнивать их и ставить больший элемент левее меньшего. Повторим эту операцию до конца, а затем еще 5 раз:

```
#include <stdio.h>
#include <stdlib.h>
// печать массива
void printM(const int * mas, int n){
    int i;
  for(i = 0; i < n; i++){
     printf("%d ", mas[i]);
  }
  printf("\n");
};
// Читать массив из файла
void getM(FILE* fin, int * m, int n){
   int i;
  for(i = 0; i < n; i++){
     fscanf(fin,"%d", m + i);
  }
};
// Поменять два элемента местами
void swp(int * a, int *b){
   int c =*b;
   *b = *a;
   *a =c;
};

int main(){
 int *m;
  int n;
  FILE *f;
  f=fopen("t.dat","r");
  fscanf(f, "%d", &n);
// выделили память под массив
```

```
  m = (int*)calloc(n, sizeof(int));
// получили массив из файла
  getM(f,m,n);
// напечатали
  printM(m,n);

  int i,j;
// сортируем массив "пузырьком"
  for(j = 0; j < n-1; j++){
   printf("j=%d\n", j);
    for(i = 0; i < n - 1 ; i++)
      if( m[i] < m[i+1]){
       swp(m + i, m + i + 1);
       printM(m,n);
      }
  }
 printM(m,n);
 // освобождаем выделенную память!!!
   free(m);
   return 0;
}
```

Наша программа выведет следующее:

```
3 58 -9 7 12 33
j=0
58 3 -9 7 12 33
58 3 7 -9 12 33
58 3 7 12 -9 33
58 3 7 12 33 -9
j=1
58 7 3 12 33 -9
58 7 12 3 33 -9
58 7 12 33 3 -9
j=2
58 12 7 33 3 -9
58 12 33 7 3 -9
j=3
58 33 12 7 3 -9
j=4
58 33 12 7 3 -9
```

Массив прекрасно отсортировалася. И теперь мы знаем, что максимальный элемент – первый, а минимальный, последний.

Сортировка "выбором".

Можно попробовать еще один простой метод сортировки. Напишем функцию сортировки "выбором". Здесь мы ищем максимальный элемент сначала во всем массиве, и меняем его с первым элемнтом местами. Затем ищем максимальный элемент в подмассиве начиная со второго элемента и меняем его местами со вторым. И так далее.

```
#include <stdio.h>
#include <stdlib.h>
// печать массива
void printM(const int * mas, int n){
    int i;
  for(i = 0; i < n; i++){
     printf("%d ", mas[i]);
  }
  printf("\n");
};
// Читать массив из файла
void getM(FILE* fin, int * m, int n){
   int i;
  for(i = 0; i < n; i++){
     fscanf(fin,"%d", m + i);
  }
};
// Поменять два элемента местами
void swp(int * a, int *b){
   int c =*b;
   *b = *a;
   *a =c;
};
// Найдем номер максимального элемнта
int max(int * mass, int n){
    int i;
    int maxe = mass[0], kmax = 0;
    for(i = 0; i < n; i++){
       if(maxe < mass[i]){
         maxe = mass[i];
         kmax = i;
       }
```

```
    }
   return kmax;
};
// функция сортировки "выбором"
void sortS(int *mass, int n){
  int mn = 0;
  int j;
  for(j = 0; j< n - 1 ; j++){
     printf("j=%d\n",j);
// находим максимальный в подмассиве с номера j
     mn = max(mass + j, n - j);
// печатаем его и номер
     printf("max[%d]=%d\n",mn+j,mass[mn+j]);
// меняем местамми с элемнтов под номерм j
     swp(mass + j,mass + mn+j);
     printM(mass, n);
  }
};
int main(){
 int *m;
  int n;
  FILE *f;
  f=fopen("t.dat","r");
  fscanf(f, "%d", &n);

  m = (int*)calloc(n, sizeof(int));

  getM(f,m,n);
  printM(m,n);
// Сортировка
  sortS(m,n);
// печать
  printf("ready:\n");
  printM(m,n);
  free(m);
  return 0;
}
```

Вывод на экран нашей программы:

```
3 58 -9 7 12
j=0
max[1]=58
58 3 -9 7 12
j=1
max[4]=12
58 12 -9 7 3
j=2
max[3]=7
58 12 7 -9 3
j=3
max[4]=3
58 12 7 3 -9
ready:
58 12 7 3 -9
```

Обе эти сортировки предполагают в худшем случае два прохода по массиву. То есть количество действий будет сравнимо с n^2 от количества элемнтов.

Задачи.

Задача 1.

Вставить отладочную печать в программу с *Qsort* так, чтобы был понятен механизм выбора медианы и сортировки.

Задача 2. Бинарный поиск

Дан сортированный массив из *N* элементов.

Написать программу, которая ищет номер элемента равного заданному или выясняет, что такого элемента нет. Для этого данный элемент сравнивается со средним элементом массива. Если они равны, то номер найден. Если элемент меньше, то такой же поиск происходит в левой части массива. Если больше, то – в правой.

Задача 3. Сортировка "вставками".

Дан массив из *N* целых чисел.
1. Массив из одного числа – отсортированный
2. Исходный массив делится на две части: отсортированную и неотсортированную. Новый элемент добавляется из неотсортированной части в отсортированную, и этим самым отсортированная часть

увеличивается на один элемент, а неотсортированная уменьшается
3. Новый элемент вставляется в отсортированную часть между меньшим элементом и тем, который больше или равен новому. Если это необходимо, часть отсортированнх элементов сдвигается на один до того места, где был новый
4. Эти действия продолжаются до тех пор, пока не останется элементов в неотсортированной части массива.
5. Поиск места, в которое нужно вставить новый элемент осуществляется бинарным поиском <\ol>

```
//исходный массив
3 58 -9 7 12 33
// отсортированная часть отделяется ||,
// новый элемент - |
3 || 58 | -9 7 12 33
//1. вставляем перед 3 и двигаем на место 58
58| 3 ||  -9 7 12 33
//2. вставляем перед -9 за 3. Двигать не нужно
58 3 | -9|| 7 12 33
//4. вставляем 7 перед 3. Двигаем на одну позицию
58 |7| 3 -9|| 12 33
//5. вставляем 12 перед 7. Двигаем на одну позицию
58 | 12| 7 3 -9|| 33
//6. вставляем 33 перед 12. Двигаем на одну позицию
58 | 33| 12 7 3 -9||
// массив отсортирован
```

Задача 4.

Дан массив из ***N*** положительных и отрицательных чисел. Написать программу, которая преобразует его так. Все отрицательные числа находятся слева массива, остальные справа. При этом отрицательные отсортированны по убыванию, а положительные по возрастанию

```
33 -4 0 8 -1 -5
// преобразование
-1 -4 -5 0 8 33
```

Задача 5.

Есть ***N*** шариков. Вес каждого представлен дробным числом. Написать программу, которая ищет вес шарика наиболее соответствующий весу некоего идеального "среднего" шарика. То есть такого, что разница в количестве шариков тяжелее этого и легче минимальна.

Сортруем все подряд

Указатели на функцию

В языке С можно описать указатель на любую функцию. Собственно, вызов функции по имени, это уже обращенре к функции о адресу, где она расположена в памяти.

Можно описать ***интерфейс*** функции, которую мы ходим использовать. При этом может быть множество раздичных функций с разным содержанием, но интерфейс у них будет одинаковый. Тогда формальный вызов функций меняться не будет, а будут производиться разные действия.

Рассмортим как использовать указатель на некоторую функцию, которая получает в качестве параметров два целых числа и возвращает целое число.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// точка
typedef struct P{
  int x,y;
}Point;
// Прямоугольник
typedef struct R{
  Point lh;
  Point rd;
}Rectangle;
// Функция поворота на 90 вокруг центра прямоугольника
void rotate90(Rectangle* a){
   Point center;
   center.x = (a->lh.x+a->rd.x)/2;
   center.y = (a->lh.y+a->rd.y)/2;
   int deltx,delty;
   deltx = (a->lh.y-a->rd.y)/2;
   delty = (a->rd.x-a->lh.x)/2;
   a->lh.x = center.x - deltx;
   a->lh.y = center.y + delty;
   a->rd.x = center.x + deltx;
   a->rd.y = center.y -delty;
};
// Перенос прямоугольника в точку (0,0)
void moveToZero(Rectangle* a){
```

```
  Point center;
  center.x = (a->lh.x+a->rd.x)/2;
  center.y = (a->lh.y+a->rd.y)/2;
  a->lh.x -= center.x;
  a->lh.y -= center.y;
  a->rd.x -= center.x;
  a->rd.y -= center.y;
};
// печать прямоугольника
void printR(Rectangle a){
  printf("(%d,%d)-(%d,%d)\n",a.lh.x,a.lh.y,a.rd.x,a.rd.y);
};
// написать функции для чтения данных о прямоугольнике
int main(){
   Rectangle *m;
  int n;
  FILE *f;
  f=fopen("rec.dat","r");
  fscanf(f, "%d", &n);
// Здесь получим прямоугольник

// Описание указателя на функцию
// Функция должна возвращать void
// параметр должен быть Rectangle*
// Указатель на функцию называется act
  void (*act)(Rectangle*);
// присвоим указателю act адрес функции rotate90
  act =rotate90;
// вызовем функцию act(m)
// а работать будет rotate90
  act(m);
   printRM(m,1);
// присвоим указателю на функцию act
   act = moveToZero;
// вызов функции по адресу в act
// работает moveToZero
   act(m);
   printRM(m,1);
}
```

Библиотечная функция ***qsort*** требует следующие параметры: указатель на массив для сортировки, количество элементов, размер одного элемента массива и указатель на функцию сравнения элементов массива.

Рассмотрим несколько примеров.

Сортировка массива целых чисел

```
#include <stdio.h>
#include <stdlib.h>
// печать массива
void printM(const int * mas, int n){
    int i;
  for(i = 0; i < n; i++){
     printf("%d ", mas[i]);
  }
  printf("\n");
};
// получить массив
void getM(FILE* fin, int * m, int n){
   int i;
  for(i = 0; i < n; i++){
     fscanf(fin,"%d", m + i);
  }
};
// функция сравнения элементов массива
int cmpInt(const void* a, const void* b){
 // всегда нужны указатели на void
// эти указатели здесь преобразовываем к нужному типу
    return (*(int*)a) - (*(int*)b);
};

int main(){
   int *m;
  int n;
  FILE *f;
  f=fopen("t.dat","r");
  fscanf(f, "%d", &n);
// выделение памяти под массив
  m = (int*)calloc(n, sizeof(int));

  getM(f,m,n);
  printM(m,n);
// вызов функции qsort
  qsort(m, n, sizeof(int),cmpInt);
```

```
   printM(m,n);
}
```

Сортировка массива строк

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Массив строк состоит из указателей на начало каждой строки
// Печать массива строк
void printM(char ** mas, int n){
    int i;
  for(i = 0; i < n; i++){
     printf("%s\n", mas[i]);
  }
  printf("\n");
};
// получить строки из файла
void getM(FILE* fin, char ** m, int n){
   int i;
   int len;
   char str[100];
  for(i = 0; i < n; i++){
     fscanf(fin,"%s", str);
     len = strlen(str);
     m[i] = (char*)calloc(len, sizeof(char));
// копируем буфер в элемент массива
     strcpy(m[i],str);
  }
};
// Сравнение строк
int cmpStr(const void* a, const void* b){
    char *s1;
    char *s2;
// преобразовываем void к указателю на массив char
// и разыменовываем (строка - это один элемент массива)
    s1 = (*(char**)a);
    s2 = (*(char**)b);
// ппросто сравниваем строки
    return strcmp(s1,s2);
};

int main(){
   char **m;
  int n;
  FILE *f;
  f=fopen("str.dat","r");
  fscanf(f, "%d", &n);

  m = (char**)calloc(n, sizeof(char*));

  getM(f,m,n);
  printM(m,n);
// вызов qsort для сортировки массива строк
  qsort(m, n, sizeof(char**),cmpStr);
   printM(m,n);

}
```

Сортировка массива прямоугольников (по площади)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct P{
  int x,y;
}Point;

typedef struct R{
  Point lh;
  Point rd;
}Rectangle;
// печать одного прямоугольника
void printR(Rectangle a){
  printf("(%d,%d)-(%d,%d)\n",a.lh.x,a.lh.y,a.rd.x,a.rd.y);
};

// печать массива прямоугольников
void printRM(Rectangle *mr, int n){
    int i;
  for(i = 0; i < n; i++){
```

```
     printR(mr[i]);
  }
};
// получить весь массив
void getR(FILE* fin, Rectangle * m, int n){
   int i;
  for(i = 0; i < n; i++){
     fscanf(fin,"%d%d%d%d",&(m[i].lh.x),&(m[i].lh.y),&(m[i].rd.x),&(m[i].rd.y) );
   }
};
// сравнение прямоугольников по площади
int cmpRec(const void* a, const void* b){
    Rectangle r1 = *(Rectangle*)a;
    Rectangle r2 = *(Rectangle*)b;
    int s1 = (r1.rd.x - r1.lh.x)*(r1.rd.y - r1.lh.y);
    int s2 = (r2.rd.x - r2.lh.x)*(r2.rd.y - r2.lh.y);
    return s1-s2;
};

int main(){
   Rectangle *m;
  int n;
  FILE *f;
  f=fopen("rec.dat","r");
  fscanf(f, "%d", &n);

  m = (Rectangle*)calloc(n, sizeof(Rectangle));

  getR(f,m,n);
  printRM(m,n);
// вызов функции qsort для сортировки массива прямоугольников
  qsort(m, n, sizeof(Rectangle),cmpRec);
   printRM(m,n);


}
```

Задача 6.

Написать функцию поиска минимального прямоугольника, которая использует указатель на функцию сравнения прямоугольников. Прямоугольники могут сравниваться по площади, периметру и "самый узкий" (тот, у которого расстояние от центра пересечения диагоналей до ближайшей стороны – миниально).

```
// massRec - массив прямоугольников
// n - количество элементов массива
// int (*cmp)(Rectangle*, Rectangle*) - указатель на функцию сравнения прямоугольников.
Rectangle minRec(Rectangle* masRec, int n, int (cmp)(Rectangle, Rectangle*));
```

Провериить работу функции minRec с разными функциями сравнения прямоугольников

Задача 7.

Множество содержит только разные элементы (в зависимости от критерия) в отсортированном виде.
Два прямоугольника считаются одинаковыми:
1. первый критерий – если площади равны
2. второй критерий – если равны соотвественно их ширина и высоты

Написать функцию добавления прямоугольника в множество, с использованием указателя на функцию сравнения

```
//rec - прямоугольник, который нужно вставить
// massRec - массив прямоугольников
// n - количество прямоугольников во множестве
// maxn - максимальный размер массива под множество
// int (*cmp)(Rectangle*, Rectangle*) - указатель на функцию сравнения прямоугольников.
void insertRecSet(Rectangle rec, Rectangle* recSet, int n, int maxn, int (cmp)(Rectangle, Rectangle*));
```

Задача 8.

В файле записана информация о городах: название города, год создания, количество жителей, расстояние от нас Написать программу, которая
1. ищет самый древний город
2. ищет город с данным названием
3. ищет самый маленький город (по количеству жителей)
4. ищет все города, растояние до которых не менее *m* километров и не более *n* километров.

Применить сортировку qsort -- TatyanaOvsyannikova2011 - 29 Nov 2016