2020/04/10_第9课_二维数组、排序算法的优劣

笔记本: C
创建时间: 2020/4/10 星期五 15:28
作者: ileemi
标签: 二维数组

排序算法的优劣

- 存储介质
- 数据结构
- 样本分布

选择法需要支持随机访问（CD切歌、链表）（访问数据的代价均等）在这个场合下，选择法优于冒泡
冒泡是在顺序访问（磁带切歌、数组）的场合时候，这个时候冒泡优于选择

可靠、稳定、海量、廉价（如医院肺部图像—个人10多G）-->磁带

RAM：随机存储器
ROM：只读存储器

数组支持随机访问：
选择法

数组的访问：
(int)ary + sizeof(type) * n
(int)ary + sizeof(type) * 6
(int)ary + sizeof(type) * 500

链表不支持随机访问：
冒泡法

数学归纳法

从基本功练起，不要上来取技巧
四两拨千斤：
1、对手犯了致命错误
2、你的水平远高于你的对手

基本功：
1、设计思路清晰，写代码快
2、分析思路清晰，读代码快

大数运算

减少运算次数

二维数组

二维数组和多维数组的本质还是一维数组，是一个特殊的一维数组，
数组的数组，这个数组的元素不是基本数据类型，而是数组。

把一维数组当作元素的称之为二维数组，把二维数组当作元素的称之为三维数组
数组的数组就是多维数组

作文本的单元格看成元素，每一行相当于单元格的数组
单元格 ary[10];
单元格 [10] 一面[20] 一本有50面 本[50]： 单元格 本[50][20][10]

```
double ary[5];  //基本类型的数组
int ary[2][3] = {
    {1, 2, 3},      //ary[0]
    {4, 5, 6}       //ary[1]
}
//ary 有 2 个元素，每个元素为 int 5
```

二维数组寻址公式

二维数组的元素为一维数组，三维数组的元素为二维数组

type ary[N][M] = ...
//N个一维数组，每个一维数组的规格为 M个type的一维数组

ary[x] address is：
(int)ary + sizeof(type[M]) * x
一次下标运算得到其元素一维数组的首地址

type ary[N][M] = ...
ary[x][y] address is：
(int)ary + sizeof(type[M]) * x + sizeof(type) * y

第二次下标运算针对一维数组做运算

对寻址公式进行优化：
(int)ary + sizeof(type) * M * x + sizeof(type) * y

(int)ary + sizeof(type) * (M * x * y)

```
    int x = 3;
    int y = 7;
    int ary[2][3] = {
        {1, 2, 3},      //ary[0]
        {4, 5, 6}       //ary[1]
    };

    int* p = &ary[0][0];
    //ary 有 2 个元素，每个元素为 int 5
    printf("%08x\r\n", &ary[x]);
    printf("%08x\r\n", (int)ary + sizeof(int[3]) * x);  //sizeof(int[3])
= sizeof(int) * 3

    printf("%08x\r\n", &ary[x][y]);
    printf("%08x\r\n", &p(3 * x * y));  //根据优化后的公式
    printf("%08x\r\n", (int)ary + sizeof(int[3]) * x + sizeof(int) * y);
```

三维数组寻址公式

type ary[L][N][M] = ...
ary[x][y][z] address is :

```
    //第 1 次下标运算等到 L 个 type[N][M]
    (int)ary + sizeof(type[N][M]) * x
    //第 2 次下标运算等到 N 个 type[M]
    (int)ary + sizeof(type[M]) * x
    //第 3 次下标运算等到 M 个 type
    (int)ary + sizeof(type) * x
```

推导公式:
1、(int)ary + sizeof(type[N][M]) * x + sizeof(type[M]) * y + sizeof(type) * z
2、(int)ary + sizeof(type) * (N * M * x + M * y + z)
3、(int)ary + sizeof(type) * (M* (N * x + y) + z)

三维数组的初始化

```
    int x = 3;
    int y = 7;
    int z = 9;
```

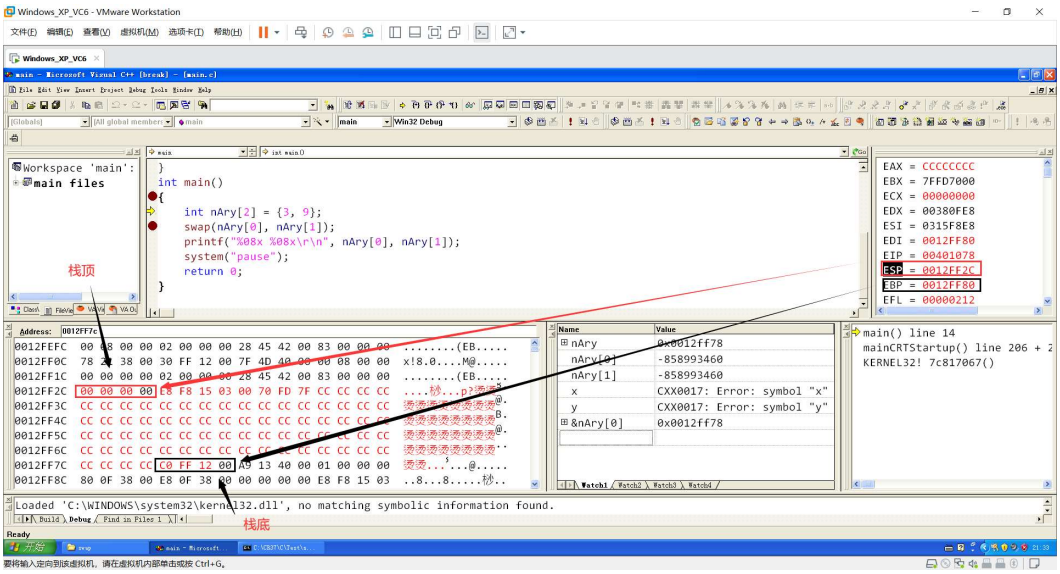
```
    int ary[2][3][4] = {
        {   //ary[0]
            {1, 2, 3, 4},    //ary[0][0]
            {5, 6, 7, 8},    //ary[0][1]
            {9, 0, 1, 2}     //ary[0][2]
        },
        {   //ary[1]
            {10, 20, 30, 40},    //ary[1][0]
            {50, 60, 70, 80},    //ary[1][1]
            {90, 80, 10, 20}     //ary[1][2]
        }
    };
    //等价
    printf("%08x\r\n", &ary[x][y][z]);
    printf("%08x\r\n", (int)ary + sizeof(int[3][4]) * x + sizeof(int[4])
* y + sizeof(int) * z));
    printf("%08x\r\n", (int)ary + sizeof(int) * ( 3 * 4 * x + 4 * y +
z));
    printf("%08x\r\n", (int)ary + sizeof(int) * ( 4 * (3 * x +  y ) +
z));
```

数组作为函数的参数

交换两个值

```
#include <stdio.h>
#include <stdlib.h>
void swap(int x, int y) //形参 copy
{
//      int nTemp = x;
//      x = y;
//      y = nTemp;
    x = x - y;
    y = x + y;
    x = y - x;
}
int main()
{
    int nAry[2] = {3, 9};
    swap(nAry[0], nAry[1]); //实参
    // 00000003, 00000009
    printf("%08x %08x\r\n", nAry[0], nAry[1]);
    //对数组元素nAry[0], nAry[1]做下标运算
    system("pause");
```

```
    return 0;
}
```




```
//
void swap(int nAry[])
{
    int nTemp = nAry[0];
    nAry[0] = nAry[1];
    nAry[1] = nTemp;

    nAry++; //可以++操作，数组名作为参数的时候不是常量
}

int nTemp(int x)
{
    //常量当作参数传递可以++
    x++;    //可以++
}

int main()
{
    int nAry[2] = {3, 9};
    swap(nAry); //数组名为地址常量

    nTemp(3);

    //这里会交换数组两个元素的值，存在间接访问
    printf("%08x %08x\r\n", nAry[0], nAry[1]);
    system("pause");
    return 0;

    nAry++; //不可以，nAry是第0项元素的地址常量
}
```

是否影响实参，主要看函数内部对实参的地址是否进行了**间接访问**，如果存在间接访问就会影响到函数外，同时也会影响到实参。下标运算就属于间接访问：nAry[0]、nAry[1]

下标运算的两个步骤：
1、取出地址
2、取值（根据地址按照数据类型取值）