# 希尔排序算法

## 算法

希尔排序又称为缩小增量排序，是对之前介绍的插入排序的一种优化版本，优化的地方在于：不用每次插入一个元素时，就和序列中有序部分中的所有元素进行比较。

该方法的基本思想是：设待排序元素序列有n个元素，首先取一个整数increment（小于序列元素总数）作为间隔，所有距离为increment的元素放在同一个逻辑数组中，在每一个逻辑数组中分别实行直接插入排序。然后缩小间隔increment，重复上述逻辑数组划分和排序工作。直到最后取increment=1，将所有元素放在同一个数组中排序为止。

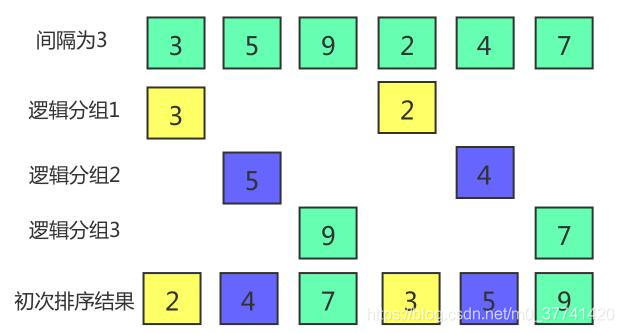
其实从上面的希尔排序的思想中也能看出希尔排序的实现步骤：

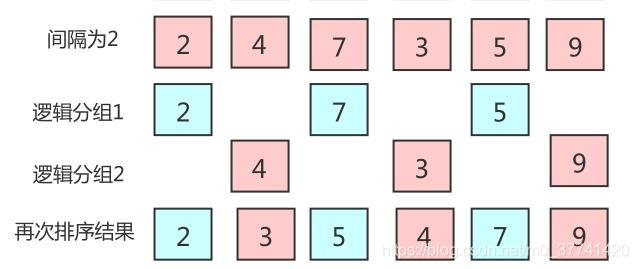
1>选increment，划分逻辑分组，组内进行直接插入排序。

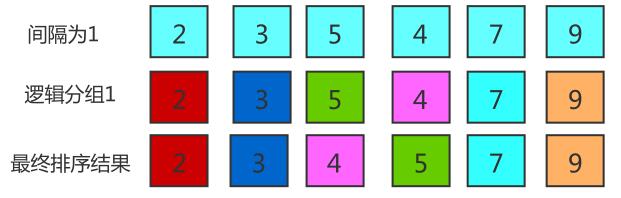
2>不断缩小increment，继续组内进行插入排序。

3>直到increment=1，在包含所有元素的序列内进行直接插入排序。

排序过程如下：

1>假如有[5,2,3,9,4,7]六个元素，第一趟取increment的方法是：6/3向下取整+1=3。将整个数据列划分为间隔为3的3个逻辑分组，然后对每一个逻辑分组执行直接插入排序，相当于对整个数组执行了部分排序调整。

2>第二趟将间隔increment= increment/3向下取整+1=2，将整个数组划分为2个间隔为2的逻辑分组，分别进行排序。

3>第3趟继续缩小间隔。间隔缩小为increment= increment/3向下取整+1=1，当增量为1的时候，实际上就是把整个数列作为一个逻辑分组进行插入排序。

从上述代码可以看出，希尔排序的代码与直接插入排序的代码十分相似。因为希尔排序本就是直接插入排序的优化版本，不同的地方就是间隔慢慢减成1，直接插入排序的间隔一直就是1。

## C语言设计

|  |
| --- |
| #include"stdio.h"  int arr[8] = {3,8,1,5,2,4,6,7};  void Shell\_Sort(int \*arr, int len)  {  /\*初始化划分增量\*/  int increment = len;  int temp;  int i,j;    while (increment > 1) /\*每次减小增量，直到increment = 1\*/  {  increment = increment/3 + 1; /\*增量下降法之一：除三向下取整+1\*/    /\*对每个按增量划分后的逻辑分组，进行直接插入排序\*/  for (j = increment; j < len; j++)  {  if (arr[j-increment] > arr[j]) // 这个if去掉不影响结果，加上简化计算  {  temp = arr[j];    for(i = j-increment;i >= 0 && arr[i] > temp;i = i- increment) // 满足条件就后移  {  arr[i+increment] = arr[i];  }  arr[i+increment] = temp; // 插入元素  }  }  }  }  int main()  {  int i;  // 显示排序前的序列  for(i=0;i<=7;i++)  printf("%d ",arr[i]);  printf("\n");  // 排序  Shell\_Sort(arr,8);  // 显示排序后的序列  for(i=0;i<=7;i++)  printf("%d ",arr[i]);  printf("\n");  } |

## 汇编语言设计

用LoongIDE新建一个工程。

移除core文件夹下的bsp\_start.c源文件。

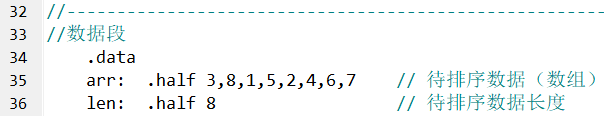
移除main.c源文件。

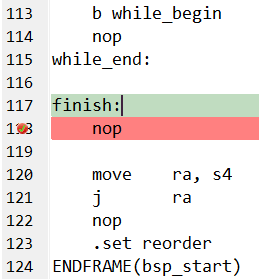
添加bsp\_start.S源文件。

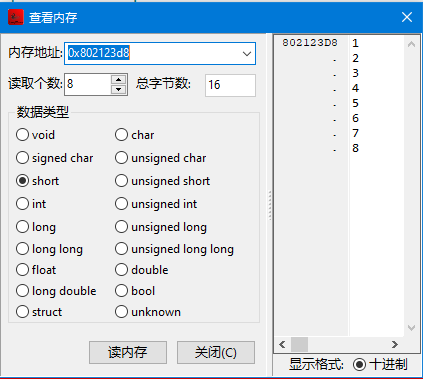
在bsp\_start.S下添加如下代码：

|  |
| --- |
| /\*  \* bsp\_start.S  \*  \* created: 2022/1/27  \* author: Li TianLing  \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*  \* v0 存储器寻址辅助寄存器  \* s0 待排序数据arr基地址  \* s1 increment  \* s2 参数传递-->len  \* s3 跳转辅助寄存器  \* t1 哨兵i  \* t2 哨兵i+increment  \* t3 哨兵i指向的值，即arr[i]  \* t4 哨兵i+increment指向的值，即arr[i+increment]  \* t5 哨兵j  \* t6 哨兵j指向的值，即arr[j]  \* t7 临时变量temp  \* t8 哨兵j-increment  \* t9 哨兵j-increment指向的值，即arr[j-increment]  \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include "regdef.h"  #include "cpu.h"  #include "asm.h"  //-----------------------------------------------------------------------------  //数据段  .data  arr: .half 3,8,1,5,2,4,6,7 // 待排序数据（数组）  len: .half 8 // 待排序数据长度  //-----------------------------------------------------------------------------  //代码段  .text  FRAME(bsp\_start,sp,0,ra)  .set noreorder  move s4, ra /\* 返回地址 \*/  la s5,3  la s0,arr // 加载待排序数据基地址（常量）  la s2,len // 加载、计算len(一个半字两个字节) 8\*2=16  lh s2,(s2)  sll s2,1  move s1,s2  while\_begin:  sub s3,s1,2 // increment<=2结束循环  blez s3,while\_end  nop  div s1,6 // increment=(increment/3)向下取整+1,但是半字占两字节，所以increment=(increment/6)向下取整\*2+2  mflo s1  sll s1,1  add s1,2  move t5,s1 // 哨兵j赋值  for\_1\_begin:  sub s3,t5,s2 // 参数校验，t5>=s2结束循环1,即j遍历到len-1结束  bgez s3,for\_1\_end  nop  if\_begin:  sub t8,t5,s1 // 计算j-increment  add v0,s0,t8 // 将arr[j-increment]加载到t9  lh t9,(v0)  add v0,s0,t5 // 将arr[j]加载到t6  lh t6,(v0)  sub s3,t9,t6 // 如果arr[j-increment]<=arr[j],不执行if  blez s3,if\_end  nop  move t7,t6 // 将arr[j]直接导入temp(t7)    move t2,t5 // 哨兵i初始化（哨兵j-increment）  sub t1,t2,s1  for\_2\_begin:  sub s3,t1,zero // t1<zero结束，即i<0结束循环2  bltz s3,for\_2\_end  nop  add v0,s0,t1 // 将arr[i]读入t3  lh t3,(v0)  sub s3,t3,t7 // arr[i]<=temp结束，即t3<=t7结束循环2  blez s3,for\_2\_end  nop  add v0,s0,t2 // 否则将arr[i]写入arr[i+increment]  sh t3,(v0)  sub t1,s1 // i-increment，开始循环2  sub t2,s1 // (i+increment)-increment  b for\_2\_begin  nop  for\_2\_end:  add v0,s0,t2 // 最后将temp插入arr[i+increment]  sh t7,(v0)  if\_end:  add t5,2 // j+2,开始循环1  b for\_1\_begin  nop  for\_1\_end:  b while\_begin  nop  while\_end:  finish:  nop  move ra, s4  j ra  nop  .set reorder  ENDFRAME(bsp\_start) |

## 调试运行

初始数据如下所示：

如下图所示，在finish后打上断点：

联合调试后，双击s0寄存器，查看对应存储器数据：

如上图所示，排序成功。