# 哈夫曼树与和哈夫曼编码

## 基本概念

路径：在一棵树中，从一个结点往下可以达到的结点之间的通路，称为路径。

路径长度：某一路径所经过的“边”的数量，称为该路径的路径长度。

节点的带权路径长度：若将树中结点赋给一个带有某种含义的数值，则该数值称为该结点的权。从根结点到该结点之间的路径长度与该结点的权的乘积，称为该结点的带权路径长度。

树的带权路径长度：树的带权路径长度规定为所有叶子结点的带权路径长度之和，记为WPL。

哈夫曼树：给定n个权值作为n个叶子结点，构造一棵二叉树，若该树的带权路径长度达到最小，则称该二叉树为哈夫曼树，也被称为最优二叉树。

根据树的带权路径长度的计算规则，我们不难理解：树的带权路径长度与其叶子结点的分布有关。

即便是两棵结构相同的二叉树，也会因为其叶子结点的分布不同，而导致两棵二叉树的带权路径长度不同。

那如何才能使一棵二叉树的带权路径长度达到最小呢？

根据树的带权路径长度的计算规则，我们应该尽可能地让权值大的叶子结点靠近根结点，让权值小的叶子结点远离根结点，这样便能使得这棵二叉树的带权路径长度达到最小。

## 哈夫曼树的构建

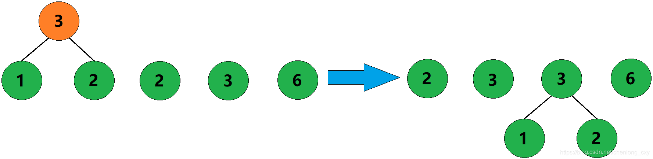
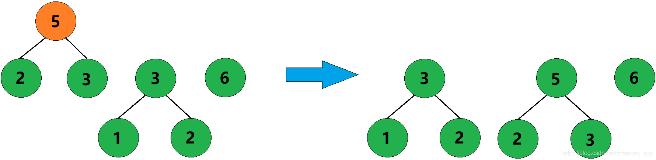
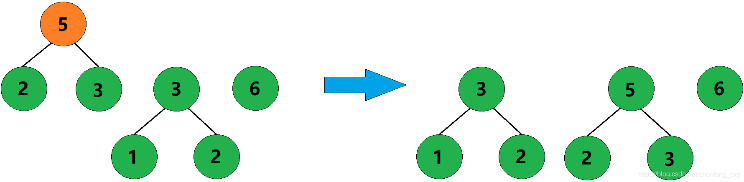
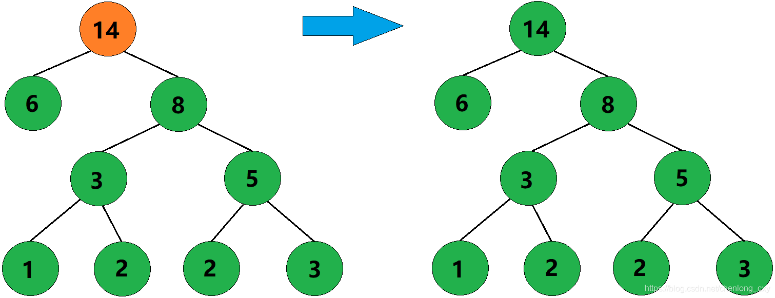
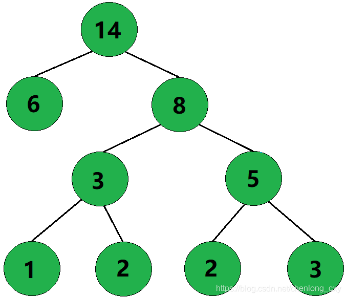
构建思路：下面给出一个非常简洁易操作的算法，来构造一棵哈夫曼树。

1>初始状态下共有n个结点，结点的权值分别是给定的n个数，将他们视作n棵只有根结点的树。

2>合并其中根结点权值最小的两棵树，生成这两棵树的父结点，权值为这两个根结点的权值之和，这样树的数量就减少了一个。

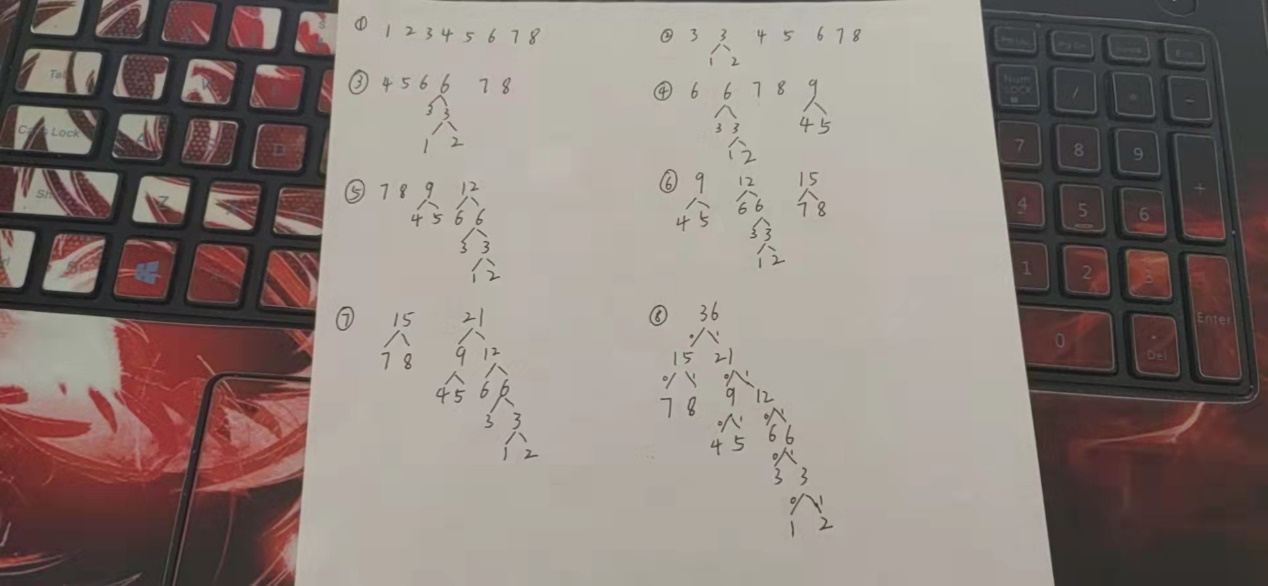
3>重复操作2，直到只剩下一棵树为止，这棵树就是哈夫曼树。

构建过程如下：

1. 初始状态：有5棵只有根结点的树。
2. 合并权值为1和2的两棵树，生成这两棵树的父结点，父结点权值为3。
3. 合并权值为2和3的两棵树，生成这两棵树的父结点，父结点权值为5。
4. 合并权值为3和5的两棵树，生成这两棵树的父结点，父结点权值为8。
5. 合并权值为6和8的两棵树，生成这两棵树的父结点，父结点权值为14。
6. 此时只剩下一棵树了，这棵树就是哈夫曼树。

观察这棵哈夫曼树，我们还可以发现，哈夫曼树不存在度为1的结点。因为我们每次都是选择两棵树进行合并，自然不存在度为1的结点。

由此我们还可以推出，若给定n个数要求构建哈夫曼树，则构建出来的哈夫曼树的结点总数为2n-1，因为对于任意的二叉树，其度为0的叶子结点个数一定比度为2的结点个数多1。

代码构建1到8的哈夫曼树过程图解如下：

## 哈夫曼编码

哈夫曼编码：对于任意一棵二叉树来说，把二叉树上的所有分支都进行编号，将所有左分支都标记为0，所有右分支都标记为1。

那么对于树上的任何一个结点，都可以根据从根结点到该结点的路径唯一确定一个编号。

代码实现1到8的哈夫曼编码过程如下：

从需要被编码的节点（叶子节点）向上编码并存放在编码缓存区code，直到节点无父节点。最后从编码缓存区拷贝出哈夫曼编码，以上图中的“1”为例编码过程如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 当前节点 | 父节点 | 编码缓存区code | | | | |
| 1 | 3 |  |  |  |  | 0 |
| 3 | 6 |  |  |  | 1 | 0 |
| 6 | 12 |  |  | 1 | 1 | 0 |
| 12 | 21 |  | 1 | 1 | 1 | 0 |
| 21 | 36 | 1 | 1 | 1 | 1 | 0 |
| 36 | 无 | 1 | 1 | 1 | 1 | 0 |

## C语言设计

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  // 四个单位为一组：权值偏移量为0，父节点坐标偏移量为1，左孩子节点坐标偏移量为2，右孩子节点偏移量为3  //在下标为1到i-1的范围找到权值最小的两个值的下标，其中坐标s1的权值小于坐标s2的权值  void Select(int\* HF\_Tree, int len, int& s1, int& s2)  {  int min;  // 找第一个最小值  for (int j = 1; j <= len; j++) // 先给min一个初始值，数组中第一个无父节点的坐标  {  if (HF\_Tree[j\*4+1] == 0)  {  min = j;  break;  }  }  for (int j = min + 1; j <= len; j++) // 找最小值点坐标  {  if (HF\_Tree[j\*4+1] == 0 && HF\_Tree[j\*4] < HF\_Tree[min\*4])  min = j;  }  s1 = min; // 第一个最小值点坐标给s1    // 找第二个最小值  for (int j = 1; j <= len; j++) // 先给min一个初始值，数组中第一个无父节点的坐标，并且不能是已经用过的s1  {  if (HF\_Tree[j\*4+1] == 0 && j != s1)  {  min = j;  break;  }  }  for (int j = min + 1; j <= len; j++) // 找除s1之外的最小值点坐标  {  if (HF\_Tree[j\*4+1] == 0 && HF\_Tree[j\*4] < HF\_Tree[min\*4]&&j != s1)  min = j;  }  s2 = min; // 第二个最小值点坐标给s2  }  //构建哈夫曼树（构建出来的树，待编码数据，个数）  void CreateHuff(int\* HF\_Tree, int\* basic\_date, int number)  {  int m = 2 \* number - 1; //哈夫曼树总结点数  for (int i = 1; i <= number; i++)  {  HF\_Tree[i\*4] = basic\_date[i - 1]; // 将权值赋给n个叶子结点  }  for (int i = number + 1; i <= m; i++) // 构建哈夫曼树，从number+1到m把辅助节点补充完毕  {  // 选择权值最小的s1和s2，生成它们的父结点  int s1, s2;  Select(HF\_Tree, i - 1, s1, s2); // 在下标为1到i-1的范围找到权值最小的两个值的下标，其中s1的权值小于s2的权值  HF\_Tree[i\*4] = HF\_Tree[s1\*4] + HF\_Tree[s2\*4]; //i的权重是s1和s2的权重之和  HF\_Tree[s1\*4+1] = i; // s1的父亲是i  HF\_Tree[s2\*4+1] = i; // s2的父亲是i  HF\_Tree[i\*4+2] = s1; // 左孩子是s1  HF\_Tree[i\*4+3] = s2; // 右孩子是s2  }  // 打印哈夫曼树中各结点之间的关系  printf("哈夫曼树为:>\n");  printf("本节点坐标 本节点权值 父结点坐标 左孩子坐标 右孩子坐标\n");  printf("0 \n");  for (int i = 1; i <= m; i++)  {  printf("%-4d %-4d %-6d %-6d %-6d\n", i, HF\_Tree[i\*4], HF\_Tree[i\*4+1], HF\_Tree[i\*4+2], HF\_Tree[i\*4+3]);  }  printf("\n");  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*  \* HF\_Code是指向指针的指针：开辟(number+1)个大小为(char\*)的数据空间，HF\_Code[i]中存放的是指向字符类型的指针  \* HF\_Code[i]是指向字符类型的指针：在该指针指向的地址下，开辟(number-start)个大小为char的数据空间，存放字符  \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  //生成哈夫曼编码  void HuffCoding(int\* HF\_Tree, char\* HF\_Code, int number)  {  char code[8]={0}; // 辅助空间，编码最长为n(最长时，前n-1个用于存储数据，最后1个用于存放'\0')  code[number - 1] = '\0'; // 辅助空间最后一个位置为'\0'  for (int i = 1; i <= number; i++) // 对number个数据依次编码  {  int start = number - 1; // 每次生成数据的哈夫曼编码之前，先将start指针指向'\0'  int c = i; // 正在进行的第i个数据的编码  int p = HF\_Tree[c\*4+1]; // 找到该数据的父结点  while (p) // 直到父结点为0，即父结点为根结点时，停止  {  if (HF\_Tree[p\*4+2] == c) // 如果该结点是其父结点的左孩子，则编码为0，否则为1  {  start--;  code[start] = '0';  }  else  {  start--;  code[start] = '1';  }  c = p; // 继续往上进行编码  p = HF\_Tree[c\*4+1]; // 继续找c的父结点  }  for(int j=0;code[start]!=NULL;j++)  {  HF\_Code[i\*8+j] = code[start];  start++;  }    }  }  //主函数  int main()  {  /\* 待编码数据数组w \*/  int number = 8;  int basic\_date[8]={1,2,3,4,5,6,7,8};    /\* 构建哈夫曼树 \*/ // 哈夫曼节点2\*number-1  int HF\_Tree[64]={0}; // 因为下标为0的组不存储数据，所以开节点个数+1组（每组四个）即（（2\*8-1）+1）\*4  CreateHuff(HF\_Tree, basic\_date, number);  /\* 构建哈夫曼编码 \*/  char HF\_Code[72]={0}; // 8个为一组i\*8+j 9\*8  HuffCoding(HF\_Tree, HF\_Code, number);  /\* 打印哈夫曼编码 \*/  for (int i = 1; i <= number; i++)  {  printf("数据<%d>的编码为:", HF\_Tree[i\*4], HF\_Code[i]);  for(int j=0;j<6;j++)  printf("%c", HF\_Code[i\*8+j]);  printf("\n");  }  return 0;  } |

## 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。

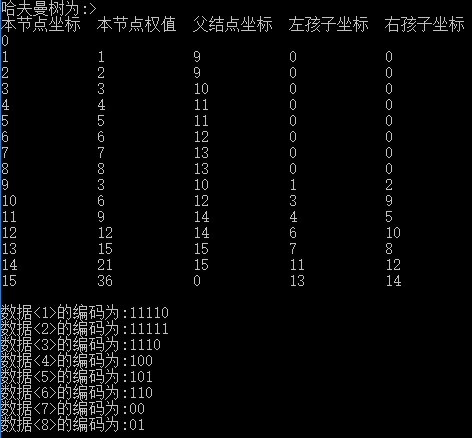
移除main.c源文件。

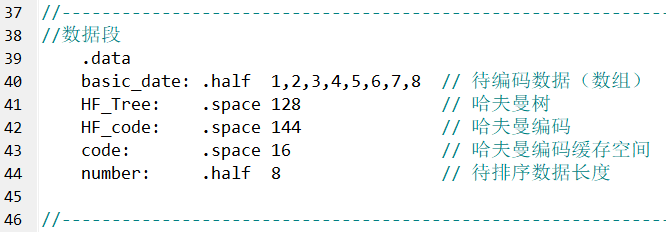
添加bsp\_start.S源文件。

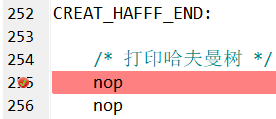
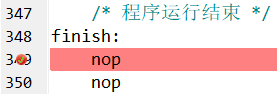
在bsp\_start.S下添加如下代码：

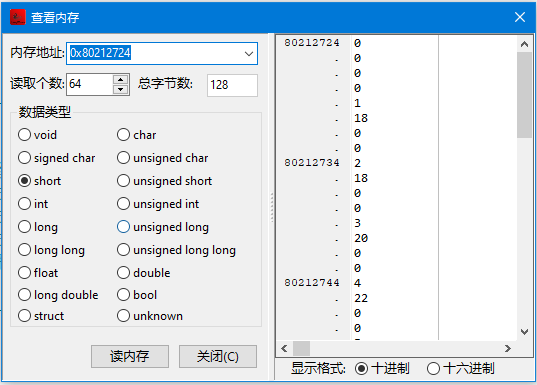
|  |
| --- |
| /\*  \* bsp\_start.S  \*  \* created: 2022/2/10  \* author: Li TianLing  \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*  \* v0 存储器寻址辅助寄存器  \* v1 跳转辅助寄存器  \*  \* s0 待编码数据basic\_date基地址 （常量）  \* s1 哈夫曼树HF\_Tree基地址 （常量）  \* s2 哈夫曼编码HF\_code基地址 （常量）  \* s3 编码缓存空间code基地址 （常量）  \* s4 待编码数据个数 （常量）  \*  \* t0 哈夫曼树总节点数m （half类型\*2）  \* t1 循环哨兵i  \* t2 循环哨兵j  \* t3 最小值点坐标s1  \* t4 最小值点坐标s2  \* t5 最小值辅助筛选min  \* t6 参数start  \* t7 参数c  \* t8 参数p  \* t9 参数len  \*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include "regdef.h"  #include "cpu.h"  #include "asm.h"  //-----------------------------------------------------------------------------  //数据段  .data  basic\_date: .half 1,2,3,4,5,6,7,8 // 待编码数据（数组）  HF\_Tree: .space 128 // 哈夫曼树  HF\_code: .space 144 // 哈夫曼编码  code: .space 16 // 哈夫曼编码缓存空间  number: .half 8 // 待排序数据长度  //-----------------------------------------------------------------------------  //代码段  .text  FRAME(bsp\_start,sp,0,ra)  .set noreorder  move k0, ra /\* 返回地址 \*/  la s0,basic\_date // 将basic\_date基地址加载到s0（常量）  la s1,HF\_Tree // 将HF\_Tree基地址加载到s0（常量）  la s2,HF\_code // 将HF\_code基地址加载到s0（常量）  la s3,code // 将code基地址加载到s0（常量）  la v0,number // 将个数numer加载到s4（常量）  lh s4,(v0)  sll s4,1  /\* 开始构建哈夫曼树 \*/  CREAT\_HAFFF\_BEGIN:  mul t0,s4,2 // m=number\*2-2  sub t0,2  la t1,2 // i=2  for\_leaf\_copy\_begin: // 将权值赋给n个叶子节点  sub v1,t1,s4 // i>number结束  bgtz v1,for\_leaf\_copy\_end  nop    sub v0,t1,2 // 将basic\_date[i-2]加载到s5  add v0,s0  lh s5,(v0)  mul v0,t1,4 // 将s5保存到HF\_Tree[i\*4]  add v0,s1  sh s5,(v0)    add t1,2  b for\_leaf\_copy\_begin  nop  for\_leaf\_copy\_end:    add t1,s4,2 // i=number+2  for\_creat\_hafftree\_begin: // 构建哈夫曼树  sub v1,t1,t0 // i>m结束  bgtz v1,for\_creat\_hafftree\_end  nop    /\* 开始挑选最小值 \*/  SELECT\_BEGIN:  sub t9,t1,2 // len=i-2  la t2,2 // j=2  for\_initial\_min\_begin\_1: // 初始化min，数组中第一个无父节点的坐标  sub v1,t2,t9 // j>len结束  bgtz v1,for\_initial\_min\_end\_1  nop  if\_no\_father\_begin\_1: // 如果无父节点  mul v0,t2,4 // 将该节点的父节点坐标HF\_Tree[j\*4+2]加载到s5  add v0,2  add v0,s1  lh s5,(v0)    bne s5,zero,if\_no\_father\_end\_1 // 父节点不为0就不执行此if  nop    move t5,t2 // 否则min=j  b for\_initial\_min\_end\_1 // 并跳出for循环  nop  if\_no\_father\_end\_1:  add t2,2 // j=j+2  b for\_initial\_min\_begin\_1  nop  for\_initial\_min\_end\_1:  add t2,t5,2 // j=min+2  for\_select\_min\_begin\_1: // 找最小点坐标  sub v1,t2,t9 // j>len结束  bgtz v1,for\_select\_min\_end\_1  nop  if\_is\_min\_begin\_1:  mul v0,t2,4 // 将该节点的父节点坐标HF\_Tree[j\*4+2]加载到s5  add v0,2  add v0,s1  lh s5,(v0)  bne s5,zero,if\_is\_min\_end\_1 // 存在父节点就结束  nop    mul v0,t2,4 // 将该节点的父节点坐标HF\_Tree[j\*4]加载到s5  add v0,s1  lh s5,(v0)  mul v0,t5,4 // 将该节点的父节点坐标HF\_Tree[min\*4]加载到s6  add v0,s1  lh s6,(v0)  sub v1,s5,s6  bgez v1,if\_is\_min\_end\_1 // HF\_Tree[j\*4]>=HF\_Tree[min\*4]就结束  nop    move t5,t2 // 否则min=j  if\_is\_min\_end\_1:  add t2,2 // j=j+2  b for\_select\_min\_begin\_1  nop  for\_select\_min\_end\_1:  move t3,t5 // 第一个最小值选取结束s1=min  la t2,2 // j=2  for\_initial\_min\_begin\_2: // 初始化min，数组中第一个无父节点的坐标  sub v1,t2,t9 // j>len结束  bgtz v1,for\_initial\_min\_end\_2  nop  if\_no\_father\_begin\_2: // 如果无父节点  mul v0,t2,4 // 将该节点的父节点坐标HF\_Tree[j\*4+2]加载到s5  add v0,2  add v0,s1  lh s5,(v0)  bne s5,zero,if\_no\_father\_end\_2 // 父节点不为0就不执行此if  nop  beq t2,t3,if\_no\_father\_end\_2 // j=s1也不执行此if  nop  move t5,t2 // 否则min=j  b for\_initial\_min\_end\_2 // 并跳出for循环  nop  if\_no\_father\_end\_2:  add t2,2 // j=j+2  b for\_initial\_min\_begin\_2  nop  for\_initial\_min\_end\_2:  add t2,t5,2 // j=min+2  for\_select\_min\_begin\_2: // 找最小点坐标  sub v1,t2,t9 // j>len结束  bgtz v1,for\_select\_min\_end\_2  nop  if\_is\_min\_begin\_2:  mul v0,t2,4 // 将该节点的父节点坐标HF\_Tree[j\*4+2]加载到s5  add v0,2  add v0,s1  lh s5,(v0)  bne s5,zero,if\_is\_min\_end\_2 // 存在父节点就结束  nop  mul v0,t2,4 // 将该节点的父节点坐标HF\_Tree[j\*4]加载到s5  add v0,s1  lh s5,(v0)  mul v0,t5,4 // 将该节点的父节点坐标HF\_Tree[min\*4]加载到s6  add v0,s1  lh s6,(v0)  sub v1,s5,s6  bgez v1,if\_is\_min\_end\_2 // HF\_Tree[j\*4]>=HF\_Tree[min\*4]就结束  nop  beq t2,t3,if\_is\_min\_end\_2 // j=s1也结束  nop  move t5,t2 // 否则min=j  if\_is\_min\_end\_2:  add t2,2 // j=j+2  b for\_select\_min\_begin\_2  nop  for\_select\_min\_end\_2:  move t4,t5 // s2=min  SELECT\_END:  mul v0,t3,4 // 将HF\_Tree[s1\*4]加载到s5  add v0,s1  lh s5,(v0)  mul v0,t4,4 // 将HF\_Tree[s2\*4]加载到s6  add v0,s1  lh s6,(v0)  add s7,s5,s6 // s7=s5+s6  mul v0,t1,4 // 将s7保存到HF\_Tree[i\*4]  add v0,s1  sh s7,(v0)    mul v0,t3,4 // 将i保存到HF\_Tree[s1\*4+2]（s1的父节点）  add v0,2  add v0,s1  sh t1,(v0)  mul v0,t4,4 // 将i保存到HF\_Tree[s2\*4+2]（s2的父节点）  add v0,2  add v0,s1  sh t1,(v0)  mul v0,t1,4 // 将s1保存到HF\_Tree[i\*4+4]  add v0,4  add v0,s1  sh t3,(v0)  mul v0,t1,4 // 将s2保存到HF\_Tree[i\*4+6]  add v0,6  add v0,s1  sh t4,(v0)    add t1,2 // i=i+2  b for\_creat\_hafftree\_begin  nop  for\_creat\_hafftree\_end:  /\* 打印哈夫曼树——省略 \*/  CREAT\_HAFFF\_END:  nop  nop    /\* 开始哈夫曼编码 \*/  HAFF\_CODING\_BEGIN:  la s5,2 // 用0和1编码，2表示结束  sub v0,s4,2 // code[number-2]=2  add v0,s3  sh s5,(v0)    la t1,2 // i=2  for\_coding\_begin:  sub v1,t1,s4 // i>number时结束  bgtz v1,for\_coding\_end  nop    sub t6,s4,2 // start=number-2  move t7,t1 // c=i  mul v0,t7,4 // p=HF\_Tree[c\*4+2]  add v0,2  add v0,s1  lh t8,(v0)    while\_begin:  beq t8,zero,while\_end // p=0结束  nop    if\_left\_child\_begin:  mul v0,t8,4 // 将HF\_Tree[p\*4+4]加载到s5  add v0,4  add v0,s1  lh s5,(v0) // HF\_Tree[p\*4+4]!=c跳转到else  bne s5,t7,else\_right\_child\_begin  nop    sub t6,2 // start=start-2  la s6,0 // s6=0  add v0,s3,t6 // 将s6保存到code[start]  sh s6,(v0)  b if\_child\_end  nop  else\_right\_child\_begin:  sub t6,2 // start=start-2  la s6,1 // s6=1  add v0,s3,t6 // 将s6保存到code[start]  sh s6,(v0)  if\_child\_end:    move t7,t8 // c=p  mul v0,t7,4 // p=HF\_Tree[c\*4+2]  add v0,2  add v0,s1  lh t8,(v0)    b while\_begin  nop  while\_end:    la t2,0 // j=0  la s5,2 // s5寄存器赋值为2  for\_copy\_code\_begin:  add v0,s3,t6 // 将code[start]加载到s6  lh s6,(v0)  beq s5,s6,for\_copy\_code\_end // s6为2复制完成结束  nop  mul v0,t1,8 // 将s6保存到HF\_code[i\*8+j]  add v0,t2  add v0,s2  sh s6,(v0)  add t6,2 // start=start+2  add t2,2 // j=j+2  b for\_copy\_code\_begin  nop  for\_copy\_code\_end:    /\* 补上一个编码结束标志 \*/  la s6,2 // 结束标志2  mul v0,t1,8 // 将s6保存到HF\_code[i\*8+j]  add v0,t2  add v0,s2  sh s6,(v0)    add t1,2 // i=i+2  b for\_coding\_begin  nop  for\_coding\_end:  HAFF\_CODING\_END:  /\* 程序运行结束 \*/  finish:  nop  nop  move ra, k0  j ra  nop  .set reorder  ENDFRAME(bsp\_start) |

## 调试运行

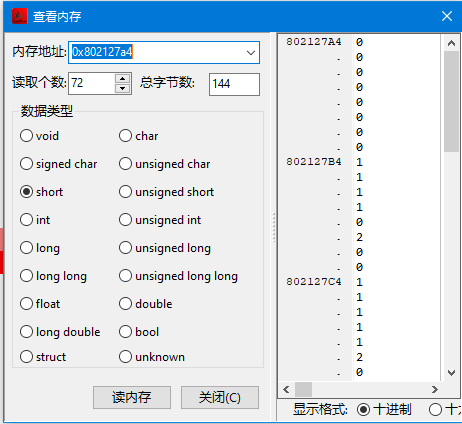
C语言运行结果：

初始数据如下所示：

如下图所示，在构建哈夫曼树结束的位置与finish后打上断点：

联合调试后，双击s1寄存器，查看构建完成的哈夫曼树：

【注】存储器中数据为“半字”类型，一个数据占两个字节，所以与C语言实现的结果而言，坐标值回翻倍。

继续运行后，双击s2寄存器，查看哈夫曼编码（以2结束）：

如上图中，“1”的哈夫曼编码是“11110”，“2”的哈夫曼编码是“11111”，结果正确。