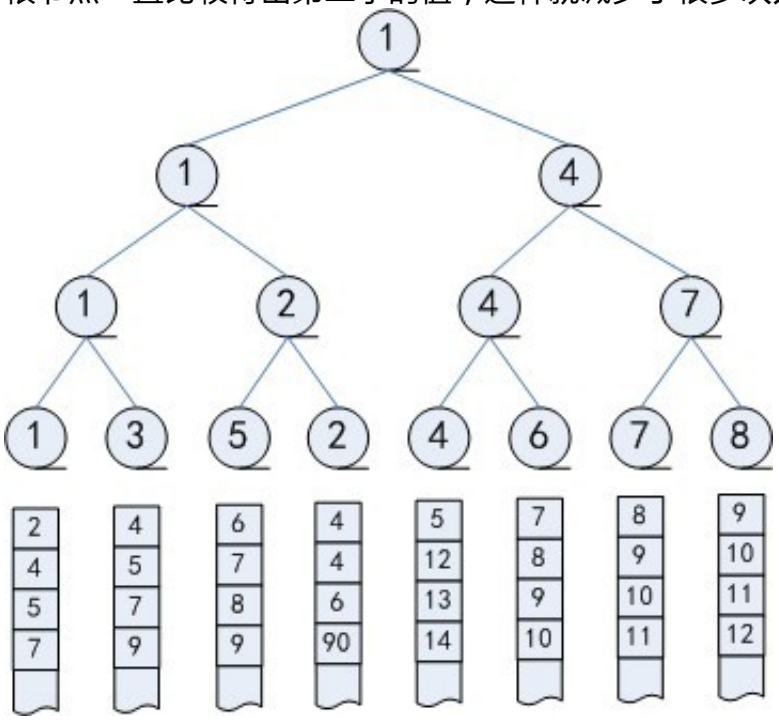
数据结构：胜者树与败者树

假设有k个称为顺串的有序序列，我们希望将他们归并到一个单独的有序序列中。每一个顺串包含一些记录，并且这些记录按照键值的大小，以非递减的顺序排列。令n为k个顺串中的所有记录的总数。并归的任务可以通过反复输出k个顺串中键值最小的记录来完成。键值最小的记录的选择有k种可能，它可能是任意有一个顺串中的第1个记录。并归k个顺串的最直接的办法就是进行k-1次比较确定下一个输出的记录。对k>2，我们可以通过使用选择数这种数据结构来降低寻找下一个最小元素所需要进行的比较次数。有两个种选择树：胜利树和失败树。

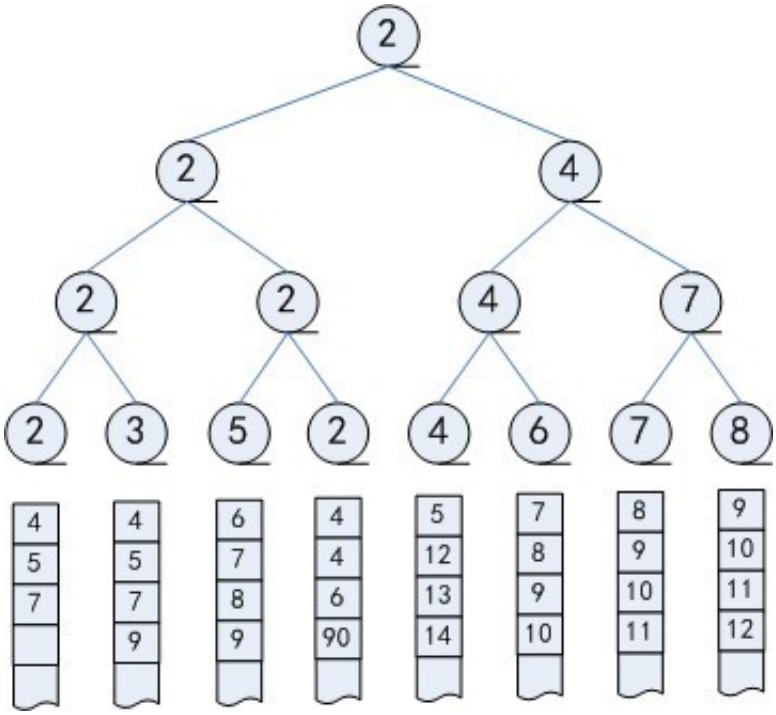
胜者树与败者树是完全二叉树。就像是参加比赛一样，每个选手有不同的实力，两个选手PK,实力决定胜负，晋级下一轮，经过几轮之后，就能得到冠军。不同的是，胜者树的中间结点记录的是胜者的标号；而败者树的中间结点记录的败者的标号。 胜者树与败者树可以在log(n)的时间内找到最值。任何一个叶子结点的值改变后，利用中间结点的信息，还是能够快速地找到最值。在k路归并排序中经常用到。

一、胜者树

胜者树的一个优点是，如果一个选手的值改变了，可以很容易地修改这棵胜者树。只需要沿着从该结点到根结点的路径修改这棵二叉树，而不必改变其他比赛的结果。下面是选择一个最小的数字为最胜利者（见图1所示），第一次把各个数组里面的第一个元素都放进去，这是根据胜利树的规则两两比较，得到最小的值，第一次弄完之后，就得出1数字是胜利的，也就是1是最小的。在下一次输出第二小的数字时候，只需要把1所在的数组里面的元素加进去，然后从叶子节点到根节点一直比较得出第二小的值，这样就减少了很多次无用的比较（见图2所示）。



图一



图二

问题：有20个有序数组，每个数组有500个uint变量，降序排序。要求从这10000个元素中选出最大的500个。
程序：

```
#include <iostream>
#include <vector>
#include <cmath>
#include <ctime>
#include <algorithm>

/**
*
* Author: w397090770
* Data  : 2012.10.15
* Email : wyphao.2007@163.com
* 转载请注明出处，谢谢。
*
*/
#define N 10
#define INF (1 << 31) - 1
using namespace std;
typedef struct Node{
  int which; //记录是哪个子数组
```

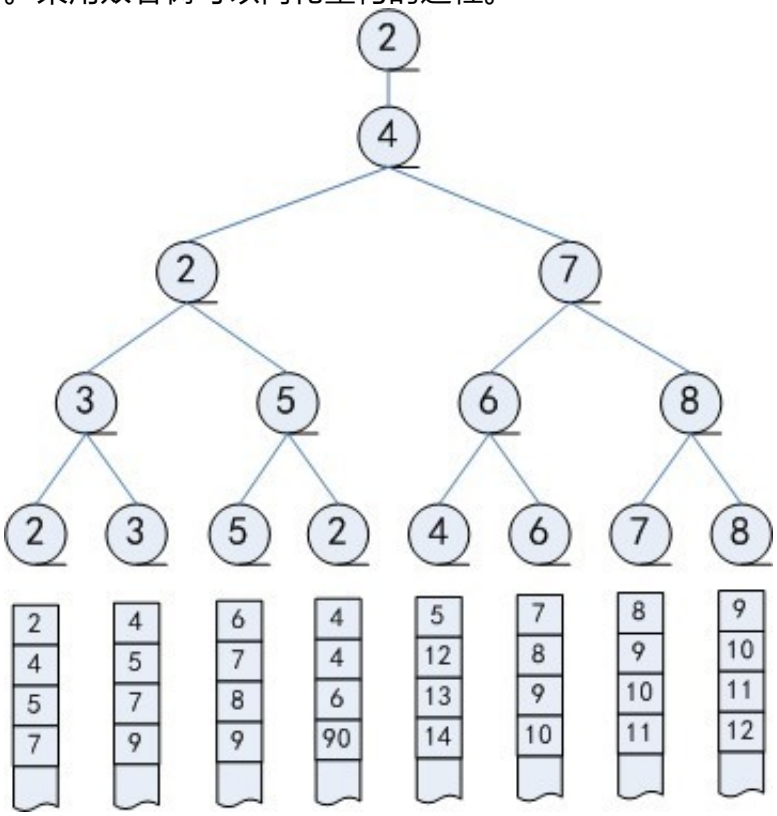
```
  int index; //记录是上个标记数组的第几个元素了
  int data; //记录数组的元素
}Node;
int com(const void *a, const void *b){
  if(*(int *)a > *(int *)b){
  return 1;
 }else if(*(int *)a < *(int *)b){
   return -1;
 }
  return 0;
}

void adjustTreeForFirst(Node *tempArray, int len){
  int i = len / 2;
  int j = 0;
  while(i > 1){
  for(j = i; j < 2 * i - 1; j += 2){
   if(tempArray[j].data > tempArray[j + 1].data){
    tempArray[j / 2] = tempArray[j + 1];
   }else{
    tempArray[j / 2] = tempArray[j];
   }
  }
  i /= 2;
 }
}

//col 是列
//row 是行
//len 是选择出前多少个元素
void winTree(int **a, int row, int col, int len){
 int *result = (int *)malloc(len * sizeof(int));
 Node tempArray[row * 2];
 int index = 0;
 int i = 0, j = 0;
 memset(tempArray, 0, sizeof(struct Node) * 2 * row);

 for(i = 0; i < row; i++){
  tempArray[row + i].data = a[i][0];
  tempArray[row + i].which = i;
  tempArray[row + i].index = 0;
 }

 for(i = 0 ; i < len; i++){
  adjustTreeForFirst(tempArray, 2 * row);//为了代码减少代码量，我把每一次调用都调整整个树
，其实是不必要的，有兴趣的用户可以自己写写
```

```
 result[i] = tempArray[1].data;
 if(tempArray[1].index + 1 < col){
  tempArray[row + tempArray[1].which].data = a[tempArray[1].which][tempArray[1].index + 1];
  tempArray[row + tempArray[1].which].index = tempArray[1].index + 1;
  tempArray[row + tempArray[1].which].which = tempArray[1].which;
 }else{//如果某个数组里面的数据处理完了，就把那个节点赋值为无穷大
  tempArray[row + tempArray[1].which].data = INF;
  //tempArray[row + tempArray[1].which].index = tempArray[1].index + 1;
  //tempArray[row + tempArray[1].which].which = tempArray[1].which;
 }
}

for(i = 0; i < len; i++){
 cout << result[i] << endl;
}
   free(result);
}

int main(){
 /*int a[N - 2][N] = {
  3, 4, 5, 6, 10, 11, 12, 13, 20, 21,
   1, 2, 7, 8, 9, 30, 31, 32, 33, 34,
     14, 15, 16, 17, 18, 19, 22, 23, 24, 25,
     26, 27, 28, 29, 35, 36, 37, 38, 39, 40,
     50, 51, 52, 54, 55, 65, 66, 67, 68, 69,
     53, 56, 57, 58, 59, 60, 61, 62, 63, 64,
     41, 42, 43, 44, 45, 46, 47, 48, 49, 2222,
     1, 2, 2, 4, 5, 6, 12, 13, 20, 21
 };*/
 const int size = 500;
 const int del = 20;

 int *a[del];
 int i = 0, j = 0;
 //分配内存空间
  for(i = 0; i < del; i++){
   a[i] = (int *)malloc(size * sizeof(int));
  }

  //初始化数组
 srand( time(NULL) );
 for(i = 0; i < size; i++){
  for(j = 0; j < del; j++){
   a[j][i] = rand();
  }
 }
```

```
 //排序
 for(i = 0; i < del; i++){
  qsort(a[i], size, sizeof(int), com);
 }

 //利用胜利树进行处理
 winTree(a, del, size, size);
 return 0;
}
```

二、败者树

败者树是胜者树的一种变体。在败者树中，用父结点记录其左右子结点进行比赛的败者，而让胜者参加下一轮的比赛。败者树的根结点记录的是败者，需要加一个结点来记录整个比赛的胜利者。采用败者树可以简化重构的过程。



本博客文章除特别声明，全部都是原创！
原创文章版权归过往记忆大数据（过往记忆）所有，未经许可不得转载。
本文链接: 【】（）