Assignment Two

1951976 李林飞

2021 年 5 月 8 日

1. Given an array $A = \{13, 15, 124, 28, 44, 28, 27, 5, 71\}$.

(1) Arrang A in descending order by insertion sort.

插入排序主要代码如下：

```
// 降序排序
void InsertionSort(ElemType A[], int n){
  int i,j;
  int temp;
  for(i=2; i<n;i++){   // 依次将A[2]-A[n]插入到前面已排序序列
    if(A[i] < A[i-1]){  // 若A[i]大于前驱，将A[i]插入有序表
      temp = A[i];    // 复制哨兵,A[0]不存放元素
      for(j=i-1; temp > A[j];j--)  // 从后面往前查找待插入位置
        A[j+1] = A[j];    // 向后挪位
      A[j+1] = temp;        // 复制到插入位置
    }
  }
}
```

手工模拟插入排序过程：

13	15	124	28	44	28	27	5	71
15	13	124	28/	44	28	27	5	71
15	124	13	28	44	28	27	5	71
124	15	13	28	44	28	27	5	71
124	15	28	13	44	28	27	5	71
124	28	15	13	44	28	27	5	71
124	28	15	44	13	28	27	5	71
124	28	44	15	13	28	27	5	71
124	44	28	15	13	28	27	5	71
124	44	28	15	28	13	27	5	71
124	44	28	28	15	13	27	5	71
124	44	28	28	15	27	13	5	71
124	44	28	28	27	15	13	5	71
124	44	28	28	27	15	13	71	5
124	44	28	28	27	15	71	13	5
124	44	28	28	27	71	15	13	5
124	44	28	28	71	27	15	13	5
124	44	28	71	28	27	15	13	5
124	44	71	28	28	27	15	13	5
124	71	44	28	28	27	15	13	5

其中蓝色表示有序序列；红色表示待考虑元素；紫色表示待交换两元素。

(2) Arrang A in descending order by quick sort.

快速排序划分方案采用两端法，主要代码如下：

```
// 双向快速排序划分操作 —— 降序排序
int Partition(ElementType A[], int low, int high){   // 一趟划分
  ElementType pivot = A[low];  // 将表中第一个元素作为枢轴，对表进行划分
  while(low < high){     // 循环跳出条件
    while(low < high && A[high] <= pivot) high--;

    A[low] = A[high];    // 将比枢轴小的元素移动到左端

    while (low < high && A[low] >= pivot) low++;

    A[high] = A[low];    // 将比枢轴小的元素移动到右端
  }

  A[low] = pivot; //枢轴放在最终位置
  return low;      // 返回枢轴最终位置
}

void quickSort(int A[], int low, int high){
  if(low < high){
    // Partition()划分操作
    int pivotpos = Partition(A, low, high);
    quickSort(A, low, pivotpos-1);  // 子表递归排序
    quickSort(A, pivotpos+1, high);
  }
}
```

手工模拟快速排序过程如下：

先从右边开始，再从左边。初始条件：pivot=A[low]

13	15	124	28	44	28	27	5	71
71	15	124	28	44	28	27	5	71
71	15	124	28	44	28	27	5	5
71	15	124	28	44	28	27	13	5
124	15	124	28	44	28	27	13	5
124	15	15	28	44	28	27	13	5
124	71	15	28	44	28	27	13	5
124	71	27	28	44	28	27	13	5
124	71	28	28	44	28	15	13	5
124	71	28	28	44	27	15	13	5
124	71	44	28	44	27	15	13	5
124	71	44	28	28	27	15	13	5

示意图：灰色表示该值赋给 pivot,值可认为空；紫色表示此时该值为 pivot；蓝色表示此时 pivot 在最终位置上。

(3) Describe the basic idea of binary search for decrement arrays and give a non-recursive algorithm and also the recursive version.

二分查找基本思想：

假设需查找的值为 x. 先将降序序列 $A[0...n-1]$ 分为两个等长的子序列, 比较 $A[n/2]$ 与 x 的大小：如果相等，则停止；如果 $x > A[n/2]$, 则在 A 的左半部分 $A[0...n/2-1]$ 中查找；如果 $x < A[n/2]$, 则在 A 的右半部分 $A[n/2+1...n-1]$ 中查找。直到在子序列中查到为止。

递归版本：具体细节如下：

```
// 二分查找降序序列 —— 递归
int RbinarySearch(ElemType A[], int low, int high, ElemType x){
  int mid = (low + high) / 2;
  if(A[mid] == x)     // 找到，返回
    return mid;
  else if (A[mid] < x)  // 查找前半部分
    return RbinarySearch(A, low, mid - 1, x);
  else                  // 查找后半部分
    return RbinarySearch(A, mid + 1, high, x);
}
```

非递归版本：具体细节如下：

```
// 二分查找降序序列 —— 非递归
int BinarySearch(ElemType A[], int len, ElemType x){
  int low = 0, high = len - 1, mid;

  while(low <= high){
    mid = (low + high) / 2;  // 中间位置
    if(A[mid] == x)          // 找到x
      return mid;
    else if(A[mid] < x)      // 进入左半部分查找
      high = mid -1;
    else                     // 进入右半部分查找
      low = mid + 1;
  }

  return -1;    // 查找失败
}
```

(4) Use above algorithms to find the elements (i.e. 13, 124) and provide necessary details of the searching process.

Eg1:查找 13

0	1	2	3	4	5	6	7	8
124	71	44	28	28	27	15	13	5

124	71	44	28	28	27	15	13	5

124	71	44	28	28	27	15	13	5

Eg2:查找 124

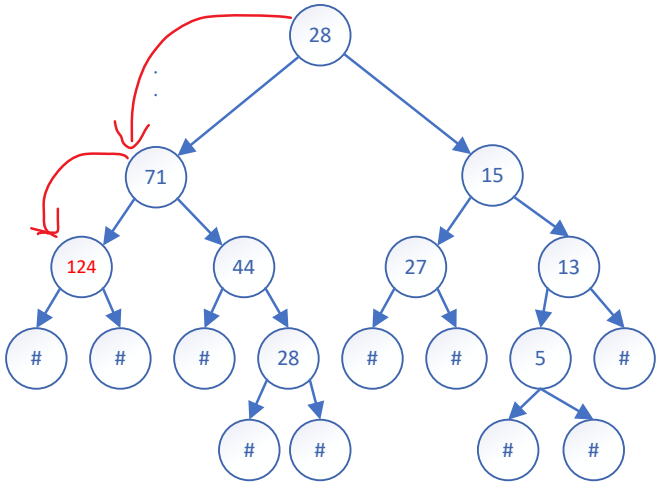
0	1	2	3	4	5	6	7	8
124	71	44	28	28	27	15	13	5

124	71	44	28	28	27	15	13	5

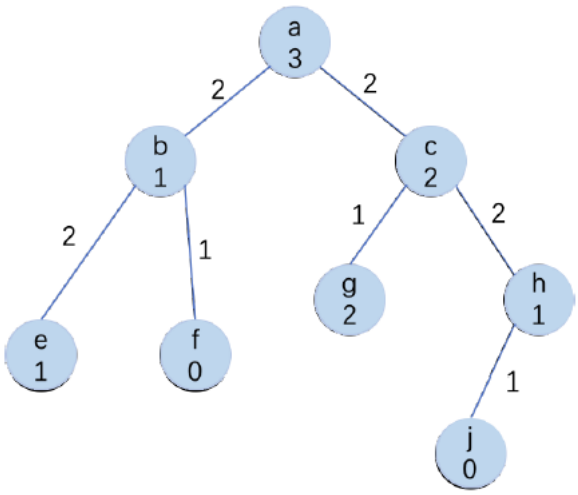
124	71	44	28	28	27	15	13	5

注：蓝色表示目前查找的有序序列；紫色表示目前的分界值；红色表示找到目标。

二分查找判定树查找过程(#表示查找失败)



2. Consider the minimal cost search problem represented in the figure, where a is the start node and there are goal nodes at f and j. For each node, the heuristic cost is indicated on the node, and for each arc, the arc cost is indicated along the arc. What is the upper bound when only the start node has been explored? Which goal node is found first by Branch and Bound? What is the upper bound immediately after the first goal node is found? Is the second goal found by Branch and Bound?



答案:

(1) 当仅探索起始节点 a 时，其上界是无穷，即 $UB(a) = \infty$。

(2) 最先找到的目标结点是 f。

(3) 当找到第一个目标节点 f 时，上界值为 3。

(4) 不能找到第二个目标结点，因为到达 j 的路径在搜索过程中被剪枝。

解析:

为使用优先队列分支限界法解决此问题,先给出以下定义:$UB(x)(upper\ bound)$ 为结点 x 的上界函数，$LB(x)(lower\ bound)$ 为结点 x 的下界函数。对于剪

枝问题，如果还未找到目标结点，则当 $LB(p) > UB$ 时进行剪枝；如果已找到一个目标结点，则当 $LB(p) \geq UB$ 时进行剪枝。

理解 1：将 heuristic cost 视为优先级

搜索过程如下：

(1) 将 a 结点加入优先队列中时，由于 a 不是目标结点，则上界为 $UB(a) = \infty$, 而 $cost(a) = 0$，故 $LB(a) = cost(a) = 0$。扩展 a 结点到 b 和 c 结点，两个都是可行结点，将其加入队列中，并将 a 结点抛弃。

(2) 由于 b 结点的优先级大于 c 结点，因此将其作为下一个扩展结点。由于其不是目标结点，故上界函数 $UB(b) = \infty$，下界函数为 $LB(b) = cost(b) = 2$。扩展 b 结点到其子结点 e 和 f, 将其加入到优先级队列中。

(3) 此时队列中有 e、f、c 三个结点，f 结点的优先级最高且是目标结点，找到目标结点 f. 此时上界函数为找到的最优解，即 $UB(f) = 2+1 = 3$, 下界函数为 $LB(f) = cost(f) = 2+1 = 3$. 接着判断 e 结点，不是可行结点，直接舍去。

(4) 判断 c 结点，其下界为 $LB(c) = cost(c) = 2 < UB = 4$, 故可将其作为新的扩展结点。扩展 c 结点，将子结点 g 和 h 加入到队列中，并将 c 结点舍去。

(5)g 和 h 结点中,g 直接舍去，h 结点的下界函数 $LB(h) = cost(h) = 2 + 2 = 4 > UB(f) = 3$，故对 h 分支进行剪枝，找不到第二个目标结点 j。

理解 2：将 heuristic cost 视为启发式成本

根据题意，设 $h(x)$ 为结点 x 的 heuristic cost,$cost(x)$ 为结点的 arc cost。

在 branch and bound 中上界函数可表示为目前找到解决方案的最小代价，如果还未找到可行的解决方案，则是无穷大 (infinite)。对于计算一条路径的下界函数：$LB(p) = f(p) = cost(p) + h(p)$。

搜索过程如下：

(1) 将 a 结点加入优先队列中时，由于 a 不是目标结点，则上界为 $UB(a) = \infty$, 而 $cost(a) = 0, h(a) = 3$, 故 $LB(a) = cost(a) + h(a) = 0 + 3 = 3$。扩展 a 结点到 b 和 c 结点，两个都是可行结点，将其加入队列中，并将 a 结点抛弃。

(2) 由于 b 结点的 heuristic cost 小于 c 结点，因此将其作为下一个扩展结点。由于其不是目标结点，故上界函数 $UB(b) = \infty$，下界函数为 $LB(b) = cost(b) + h(b) = 2 + 1 = 3$。扩展 b 结点到其子结点 e 和 f, 将其

加入到优先级队列中。

(3) 此时队列中有 e、f、c 三个结点，f 结点的 heuristic cost 最小且是目标结点，找到目标结点 f. 此时上界函数为找到的最优解，即 $UB(f) = 2 + 1 = 3$, 下界函数为 $LB(f) = cost(b) + h(b) = 2 + 1 + 0 = 3$. 接着判断 e 结点，不是可行结点，直接舍去。

(4) 判断 c 结点，其下界为 $LB(c) = cost(c) + h(c) = 4$。此时，由于 $LB(c) \geq UB(f) = 3$，因此对 c 分支进行剪枝，故找不到第二个目标结点 j。