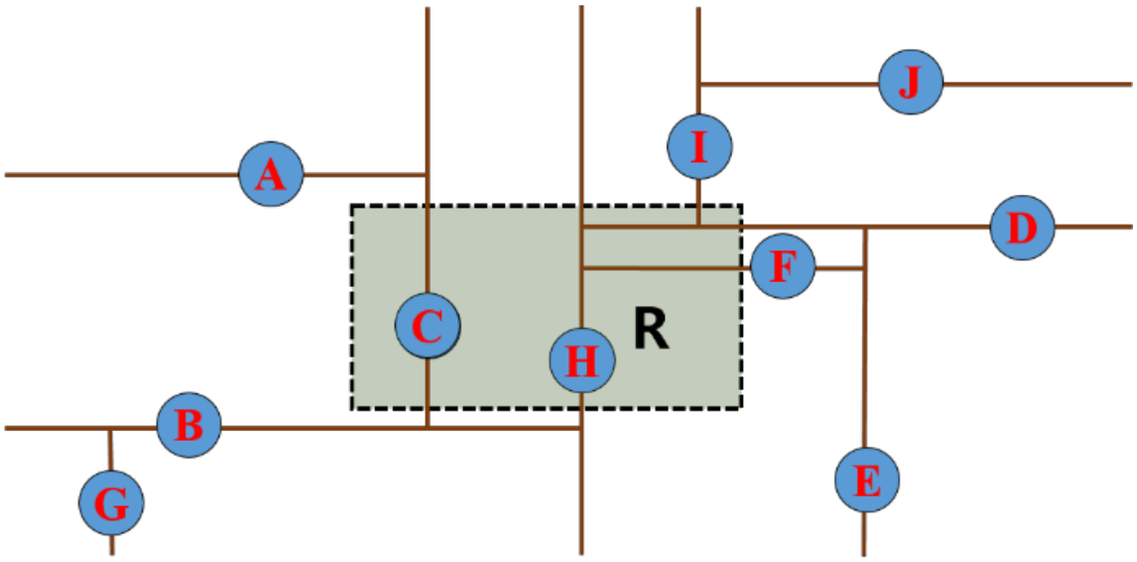
一、单选题

1．在如图所示的 2d-树（根节点为 H）中进行范围 R 的查询，则哪些节点因为减枝而没有访问到？（C）



A．E

B．J

C．G

D．A

2．一个栈的输入序列为 a,b,c,d,e,f，则下列序列中不可能是输出序列的是（D）

A．b,c,e,d,f,a

B．d,c,e,f,b,a

C．a,d,c,f,e,b

D．c,d,e,a,b,f

3．在无向图中，边用邻接表存储，点数 N 边数 E，要获取两两之间的最短距离（边值皆为正），考虑到复杂度影响，当 E ~ N 时采用 a 算法复杂度较低；当 E ~ N^2 时，采用 b 算法，复杂度较低并且实现简易。则 a,b 分别为（B）。

A．dijkstra, dijkstra

B．dijkstra, floyd

C．floyd, dijkstra

D．floyd, floyd

4．从 n 个未排序的数中寻找中位数，采用平均复杂度最低的算法，复杂度为（B）

A．$O(1)$

B．$O(n)$

C．$O(nlogn)$

D．$O(n^2)$

5．下面的序列中，（D）是堆

A．1，5，10，6，7，8，9，2

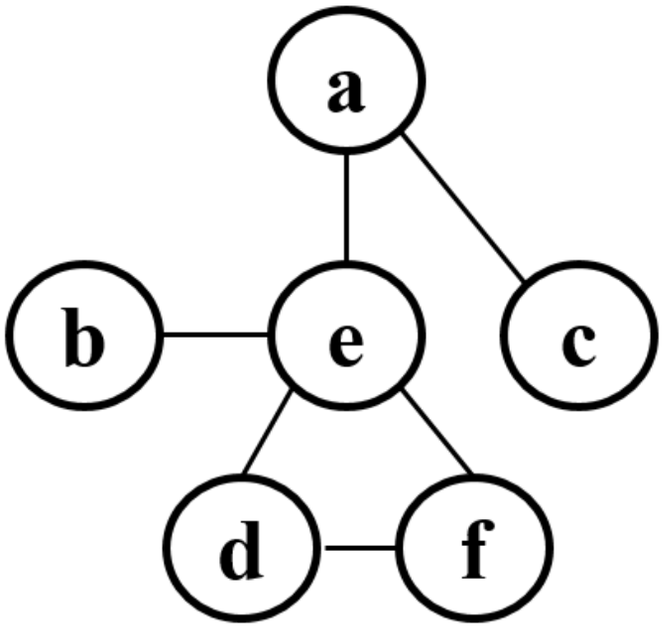
B．9，8，7，6，4，8，2，1

C．9，8，7，6，5，4，3，7

D．1，2，8，4，3，9，10，5

6．由权值分别为 11，8，6，2，5 的叶子结点生成一棵哈夫曼树，它的带权路径长度为（B）

A．73
B．71
C．48
D．53
7．具有 60 个结点的二叉树，其叶子结点有 12 个，则度为 1 的结点数为（D）
A．11
B．13
C．48
D．37
8．对给定的关键字序列 110，119，007，911，114，120，122 进行基数排序，则第 2 趟分配收集后得到的关键字序列是（B）
A．007,110,114,119,120,122,911
B．007,110,911,114,119,120,122
C．007,110,119,114,911,120,122
D．110,120,911,122,114,007,119
E．911,122,120,119,114,110,007
9. 对下面无向图进行深度优先遍历，在序列 edfabc, acefdb, caedfb, aedfcb, aebcdf 中，符合遍历的结果数目为 (B) 个



A．1 个
B．2 个
C．3 个
D．4 个
10．若序列的原始状态为（1，2，3，4，5，10，6，7，8，9），则以下排序方法（从小到大）比较次数最少的为（A）。
A．插入排序
B．希尔排序
C．冒泡排序
D．快速排序

二、填空题

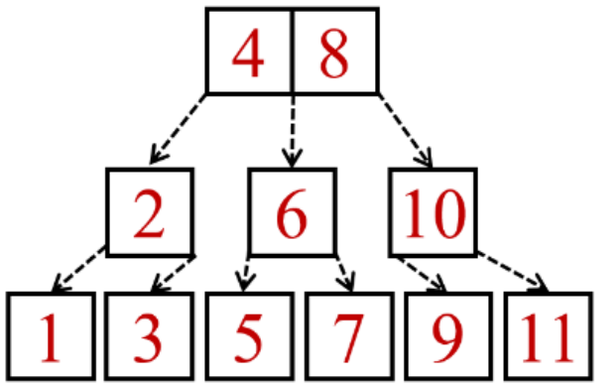
1．若一个算法的时间复杂度由以下递推式决定，且其时间复杂度为$T(n)=2n^2$（为简单起见，假设 n 为 2 的幂次），k>0。那么，k=__2__。

$$T(n)=\begin{cases}1 & (n=1)\\ kT\left(\frac{n}{2}\right)+n^2 & (n>1)\end{cases}$$

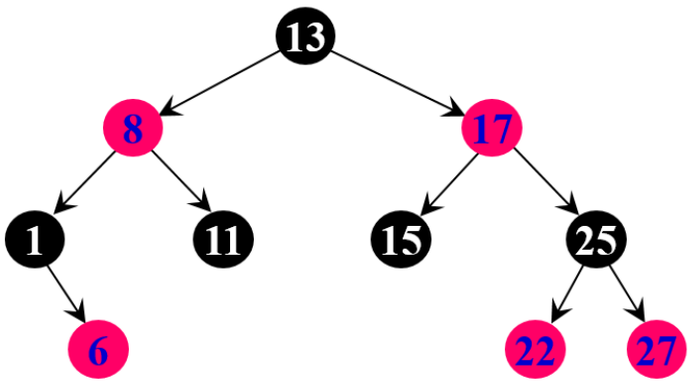
2．前缀表达式为 x,+,x,2,5,9,+,3,4 的表达式值为__133__，后缀表达式为 0,!,3,+,1,2,!,4,+,^,×,5,!,9,-,8,6,+,-,+的表达式值为__101__。

3．以{1, 10, 3, 5, 2, 4, 9, 8, 6, 7}的输入顺序建立 AVL 树，根节点的值为__5__，依次删除 7,5,3,1 后（删除时，用中序直接前驱替代），根节点的值为__8__。

4．以下 3 阶 B 树，删除节点 4 后，则关键码 8 的左子树节点所包含的关键码是（替换使用直接后继，左右皆可合并时选取左边合并）__2__，__5__。



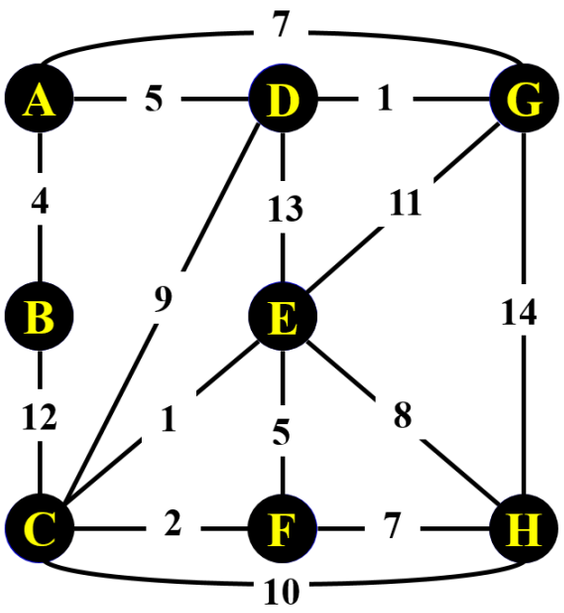
5．下图的红黑树，转化为对应的四阶 B 树后，删除关键码 15，则根节点的关键码为__8__，__13__，__22__。该红黑树插入节点 28 后，根节点为___13___。



6．字符串“aabaabaaa”的 next 数组是（采用改进后的）_-1-11-1-11-1-15_。【数字之间不要加空格，负号紧接前一数字，例如“-1,1,0,-1”写成”-110-1”】

7．设哈希表长 m=14，哈希函数为 H(key)=key MOD 11。表中已有 5 个节点 H(37)=4，H(38)=5，H(61)=6，H(84)=7，H(20)=9，其余地址为空。如用双向平方探测法处理冲突，则关键字为 49 的节点地址是__1__。

8. 对以下无向图求解从 A 出发的迪杰斯特拉最短路径树，采用优先级队列方式，则在提取完 A,B 和 D 节点之后，优先级队列中的数据元素个数为__5____。



本题可参考以下代码:

```
int Dij(int Star, int End){
      Node P, Pn;
      P.end = Star; P.weight = 0;
      priority_queue<Node> Q;
      Q.push(P);              //把顶点放入优先级队列
      while (!Q.empty()){
          P = Q.top();  Q.pop(); //提取优先级最高顶点
          if (visited[P.end]) continue;  // 若该顶点被访问过，则返回
          visited[P.end] = true; // 设置该顶点访问标记
          int nEdge = G[P.end].size(); // 该顶点的邻域表个数
          for (int i = 0; i< nEdge; i++){
              Pn.end = G[P.end][i].end; // 取出第i个邻域顶点的秩
              Pn.weight = G[P.end][i].weight + P.weight; // 对应权重修改
              if (!visited[Pn.end])    // 若该邻域顶点未被访问，则放入队列
                     Q.push(Pn);
          }
      }
      return -1;
}
```

三、主观题

1．代码补充：对于一个链表，请设计一种思路，判断是否有环。

```
bool hasCycle(ListNode *head) {
    if (head == NULL || head->next == NULL) {
        return false;
    }
    ListNode* fast = head;
    ListNode* slow = head;
    while (fast->next != NULL && fast->next->next != NULL) {
        ______(1)______ fast = fast -> next -> next ;
        ______(2)______ slow = slow -> next ;
        if (slow == fast) {
            return true;
        }
    }
    return false;
}
```

2. 代码补充：带头节点单链表，L 为指向头节点的指针，P 不为首元素，以下代码删除 P 的直接前驱及 P 本身，请完成代码。（注：以下 Q 和 K 为指向链表节点的指针）

```
Q = P; P = L;
________(1)_________; while (P->next->next != Q) P = P->next ;
K = P->next;
P->next = Q->next;
free(K); free(Q);
```

3．代码补充：给定一个无向连通图，图中每个节点代表一个地点，每条边代表两个地点之间的一条道路，边的权重代表道路的最大负重。现在一辆卡车要从一个地点开往另一个地点，保证卡车的载重要不大于其所经过的每一条道路的最大负重（例如，边 AB 的最大负重为 3，边 BC 的最大负重为 5，则从 A 经过 B 再到 C，卡车的最大负重为 3）。求卡车的最大载货重量是多少。输入：第一行两个数 N 和 M，其中 N 代表图中的地点个数，M 代表查询的个数。N 和 M<=100。接下来一个 N*N 的对称矩阵，矩阵的第 I 行和第 J 列代表地点 I 和 J 之间道路的最大负重，其值小于 10000，若 I 和 J 之间不存在道路，则为-1。接下来 M 行，每行两个数 x 和 y，代表一个卡车要从地点 x 运输到地点 y。输出：M 行，每行输出卡车的最大载重。

```
#include <iostream>
using namespace std;

int min(int a, int b)
{
    if (a>b) return b;
    else return a;
}

int main()
{
    int N, M;
    cin >> N >> M;
    int e[N][N];//储存负重
    for (int i = 0; i<N; i++)
        for (int j = 0; j<N; j++)
            cin >> e[i][j];//输入各边负重，-1为不存在
    for (int k = 0; k < N; k++)
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                if (_____(1)_____)//若有更大的负载则更新，注意e[i][i]不更新
                    _____(2)_____
    /*查询结果*/
    for (int k = 0; k < M; k++)
    {
        cin >> i;
        cin >> j;
        cout << _____(3)_____ e[i][j];
    }
    return 0;
}
```

i!=j && min(e[i][k], e[k][j]) > e[i][j]

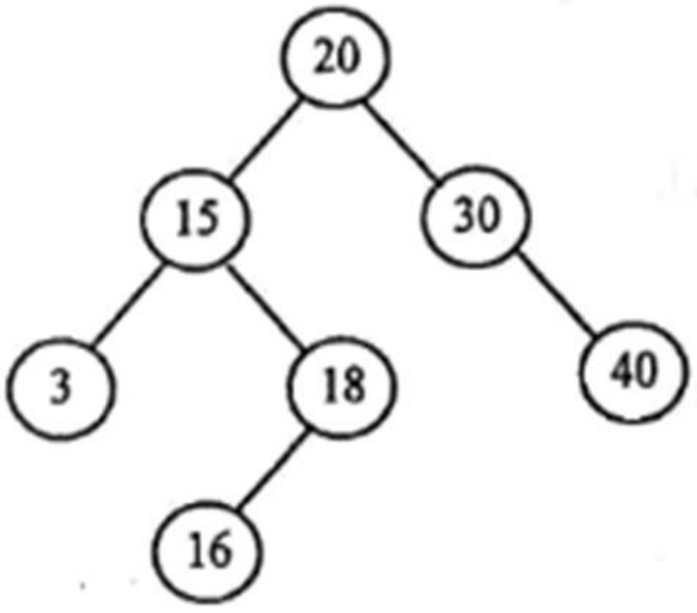
e[i][j] = min(e[i][k], e[k][j]);

4．分析回答题。二叉树的存储结构类型定义如下，阅读算法函数 A33，1）说明该算法的功能。2）给出对下图二叉树执行 A33(root,6,100,0)的输出结果。其中，root 指向该树的根节点。

1) 输出二叉树中序遍历序列中第1个大于 k_2 的元素之前所有大于等于 k_1 的元素，若存在大于k_2的元素，则end在运行后为1，否则为0.

2) 15 16 18 20 30 40

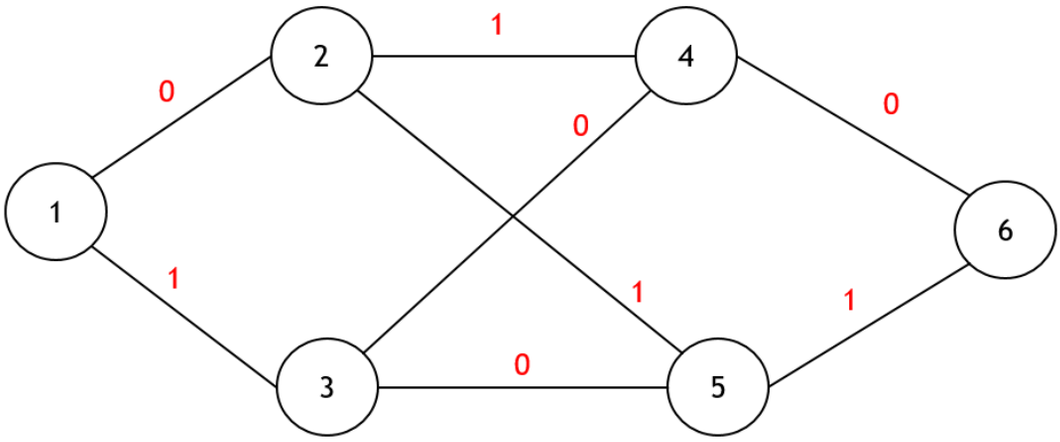
```
typedef int DataType;
typedef struct node{
    DataType key;
    struct node* lchild;
    struct node* rchild;
}BinTNode;
typedef BinTNode* BinTree;

void A33(BinTree root, int k1, int k2, int& end){
    if(root==NULL) return;
    A33(root->lchild,k1,k2,end);
    if(end) return;
    if(root->key>k2){
        end = 1; return;
    }
    if(root->key>=k1) printf("%d ",root->key);
    A33(root->rchild,k1,k2,end);
}
```



5．算法设计题：给定一棵 n 个节点的满二叉树 T，所有的叶子节点除了叶子节点 x 都具有相同的权值 w，叶子节点 x 具有另一个权值 w2，w !=w2，每个非叶子节点的权值为它的孩子的权值之和。现在给定这颗二叉树 T，遍历到一个节点可以获得该节点的权值，请从根节

点出发，通过尽量少的遍历次数找到叶子结点 x。注意：w 和 w2 未知，只知道 w!=w2，以及每个节点的权值。要求：在算法正确的前提下，尽量是的复杂度达到最优。【请写出复杂度最小的算法设计思路，分析复杂度，并尽可能写出代码】

6．算法设计：给定一个无向图 G，图上的每条边由 0 和 1 进行标号，若从 A 到 B 的路径上依次经过的边为 01010，那么该路径的长度为 01010 代表的二进制串的值，即长度为 10。现在给定起点 A 和终点 B，请用尽量快的方式求出 A 到 B 的最短路径。要求：在算法正确的前提下，尽量使得复杂度最优。【请写出复杂度最小的算法设计思路，分析复杂度，并尽可能写出代码】下面是一种情况示例，从 1 到 6 的最短路径长度为 2，走法为 1->2->4->6，路径二进制表示为 010，值为 2。具体值的计算忽略前导零。



5. 思路：该满二叉树深度 $h \geq 2$（至少有3层）。

比较 root → lchild → lchild，root → lchild → rchild，root → rchild → lchild，root → rchild → rchild 的权值，确定 w 与 w_2 的大小关系（假设 $w > w_2$）

设当前查找到的节点为 i，x在以i根的子树中。

1）若 i 为叶子节点，则 x = i，结束

2）否则，若 i->lchild->key < i->rchild->key，则 i = i->lchild，否则 i = i->rchild。

i 从根节点开始递归即可

复杂度：递归层数为 h 的层数，每次递归复杂度 O(1)，因此总的复杂度为 O(log n)

代码：略。

6. 思路：修改 Dijkstra 算法，e[i][j] 表示从 i 到 j 边上的权值。

用 d[i] 表示从 A 到 i 的最短路径长，p[i] 表示从 A 到 i 的最短路径中 i 的前驱节点。

初始化：d[A] = 0，d[i] = inf（某个不可能到的路径长，我用 -1 作为标记）。

用优先队列维护数组 d，每次找出未遍历的最小的 d[i]，激活 i，用 2×d[i] + e[i][j] 与 d[j] 比较，若更小，则更新 d[j]，p[j]，直到所有点被激活（或节点 B 被激活）。

复杂度：与 Dijkstra 算法相同，为 $O(e \log n)$，其中 n 是 G 的顶点数，e 是 G 的边数。
$O(n^2)$（不用优先队列）

代码：略。