> by harveechen@gmail.com
> *last modified 20-05-23*

2019

* 符号标注的题目存在争议

选择题

1	2	3	4	5	6	7	8	9	10
D	A	A	D	B	C	C	C	C	C

11	12	13	14	15	16	17	18	19	20
A	B	D	C	C	D	A	B	A	B

应用题

21

1)
队空条件：`rear == front`
队满条件：`(rear + 1) % m == front`
影响：数组不能存满元素，会牺牲一个存储单元
2)
元素个数 = `(rear - front + m) % m`
3)

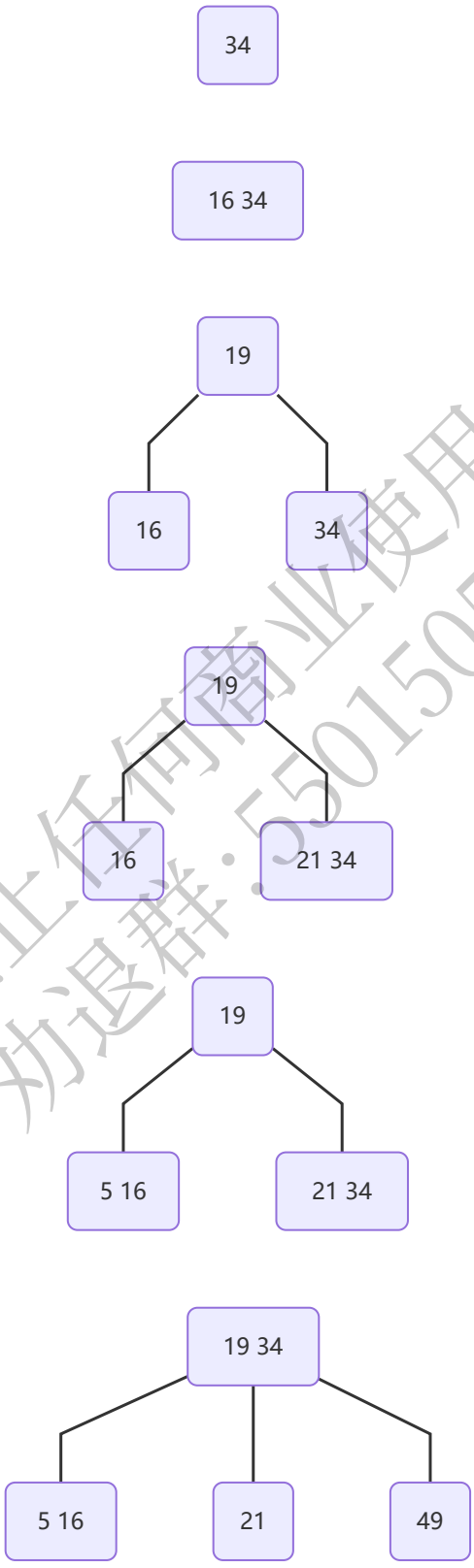
```
typedef struct Queue {
    int front, rear;
    ElemType Q[m]
} Queue*;

bool EnQueue(Queue q, ElemType x) {
    if ((q->rear+1)%m == front)
        return false;
    q->rear = (q->rear + 1) % m;
    q->Q[q->rear] = x;
    return true;
}

bool DeQueue(Queue q, ElemType *x) {
    if (rear == front)
        return false;
    q->front = (q->front + 1)%m;
    *x = q.Q[(q->front+1)%m];
```

禁止任何商业使用
劝退群：550150534

```
19      return true;
20  }
```

22

1)



禁止任何商业使用
劝退群：550150534

2)

```
1   struct Triple {
2       BtreeNode *r; // 节点地址指针
```

```
    int i; // 关键码序号
    int tag; // tag == 0, 查找成功; tag == 1, 查找失败
}

Triple Search(BtreeNode *root, ElemType x) {
    // P0, (K0, P1), (K1, P2), ... , (Km-2, Pm-1), (Km-1, Pm)
    // 数组 key 最后一个节点作为哨兵
    Triple result;
    GetNode(&root); // 从磁盘读取位于根root的节点
    int i;
    BtreeNode *p = root, *q == NULL;
    while(p != NULL) {
        i = 0;
        p->key[(p->n)] = MaxValue; //
        while(p->key[i] < x) i++;
        if (p->key[i] == x) {
            result.r = p;
            result.i = i;
            result.tag = 0;
            return result;
        }
        q = p; p = p->ptr[i];
        GetNode(p); // 从磁盘读取p节点
    }
    result.r = q; result.i = i-1; result.tag = 1; // x 落在[ki-1, ki]
    return result;
}
```

23

1)

```
int GetTreeHeight(BiTree *t) {
    Queue Q;
    InitQueue(Q);
    BiTree* p = t;
    int depth = 0;
    int height = -1;
    p->depth = depth
    EnQueue(Q, p)
    while (!IsEmpty(Q) {
        DeQueue(Q, &p);
        height = p->depth;
        if (p->lchild)
            Q.EnQueue(p->lchild, height+1);
        if (p->rchild)
            Q.EnQueue(p->rchild, height+1);
    }
    return height;
}
```

2)

```
int GetTreeDiameter(BiTree *t) {
    if (t == NULL) return 0;
    if (!t->lchild && !t->rchild) {
        t->height = 0; // 叶节点高度为0
```

禁止任何商业使用
劝退群：550150534

```
        return 0; // 叶节点直径为0
    }
    int lh,rh,ld, rd;
    ld = GetTreeDiameter(t->lchild); // 左子树的半径
    rd = GetTreeDiameter(t->rchild); // 右子树的半径
    if (t->lchild)
        lh = t->lchild->height +1;  // 左高度
    if (t->rchild)
        rh = t->rchild->height +1; // 右高度
    t->height = max(lh, rh);    // 本节点高度
    // 若最大路径包括本节点，则距离为 lh+rh
    // 否则为左右子树的最大路径值
    return max(lh+rh, ld, rd);
}
```

24

1)

记 a_i 为对应面值为 $d_i = 2^{i-1}$ 的硬币数目，则有 $\sum_{i=1}^{n} a_i d_i = N$

算法

```
int a[n], sum = 0;
for (i = n; i >= 1; i--) {
    a[i] = N/d[i];
    sum += a[i]
    N = N-a[i]*d[i]
}
```

对 n 进行数学归纳

1. 当面值集合为 $\{1\}$ 时，只有一种币值，显然 $a_1 = N$，算法成立
2. 假设当面值集合为 $\{1, 2, 2^2, \ldots, 2^{n-2}\}$ 时算法成立（即使得 $\sum_{i=1}^{n-1} a_i$ 最小）
3. 现加入面值 2^{n-1}
   若 $N < 2^{n-1}$，则 a_n 必定为 0，则问题等价于 N 由 $\{1, 2, 2^2, \ldots, 2^{n-2}\}$ 的面值集合兑换，由假设知算法成立；
   若 $N \geq 2^{n-1}$，记 $a = N/2^{n-1}$，假设存在不同于算法取值的 a'_i，有 $a'_n \leq a$；
   1. 若 $a'_n = a$，$N - a \cdot 2^{n-1}$ 将由 $\{1, 2, 2^2, \ldots, 2^{n-2}\}$ 兑换，由假设 $a_i = a'_i, i = 1, 2, \ldots, n-1$，算法成立；
   2. 若 $a'_n < a$，除了 2^{n-1} 币值兑换的部分，剩下 $N - a'_n \cdot 2^{n-1}$ 将由 $\{1, 2, 2^2, \ldots, 2^{n-2}\}$ 兑换，由假设要使剩下的金额兑换的硬币总数最小，有

$$\begin{aligned} a'_{n-1} &= (N - a'_n \cdot 2^{n-1})/2^{n-2} \\ &= (N - a \cdot 2^{n-1})/2^{n-2} + 2 \cdot (a - a'_n) \end{aligned}$$

   若令 $a_n = a$，则 $a_{n-1} = (N - a \cdot 2^{n-1})/2^{n-2}$，有

$$\begin{aligned} a'_n + a'_{n-1} &= (N - a \cdot 2^{n-1})/2^{n-2} + a + a - a'_n \\ &= a_{n-1} + a_n + a - a'_n \\ &\geq a_{n-1} + a_n + 1 \end{aligned}$$

   又 $a_i = a'_i, i = 1, 2, \ldots, n-2$，所以 $\sum_{i=1}^{n} a_i < \sum_{i=1}^{n} a'_i$，所以不存在比算法总硬币数还少的取值 a'_i

综上，算法得出的兑换方法的总硬币数最少。

2)

禁止任何商业使用 劝退群：550150534

```
int d[n+1]; // 存储面值的数组，d[0] 未用
int n, N;
ScanInput(d, &n, &N); // d[1] 一定为1，否则不能保证兑换总是可行的
int cost[n+1][N+1]; // cost[i1][j]  表示 {d1,d2,...,di} 的集合兑换金额 j 的最少硬币数
int i, j;
for (i = 0; i <= N; i++)
    cost[i][0] = 0;
for (j = 0; j <= N; j++) {
    cost[0][j] = 0;
    cost[1][j] = j;
}
for (i = 2; i <= n; i++) {
    for (j = 1; j <= N; j++) {
        if (j-d[i] < 0)
            cost[i][j] = cost[i-1][j]
        else
            cost[i][j] = min(cost[i][j-d[i]] + 1, cost[i-1][j]);
    }
}
```

25

```
semaphore a12=a13=a14=a25=a32=a45=0;
code begin
P1() { V(a12); V(a13); V(a14); }
P2() { P(a12); V(a25); }
P3() { P(a13); V(a32); }
P4() { P(a14); V(a45); }
P5() { P(a25); P(a45); }
code end
```

*26

1)

2KB/4B = 512

2)

页面大小为 2KB，按字节编址，所以页内偏移为 log(2KB/B) = 11

一张页表有512个页表项，所以页号（页表项偏移）为 log(512) = 9

可寻址位数为32，即逻辑空间，所以页目录号为 32-11-9 = 12

3)

反置页表最多为每一个物理页设置一个表项，所以最多包含页表项 2GB/2KB = 1024

4)

进程平均需要 2MB/2KB = 1K 的页表项，即需要 1K*4B/2KB = 2 页页表，故运行过程中会使用两次磁盘访问将进程全部页表调入内存，增加了IO操作和时间，故不合理

5)

page	2	1	4	2	3	1	3	1	5
5	2	2	2	2	2	2	2	2	5
10	_	1	1	1	3	3	3	3	3
20	_	_	4	4	4	1	1	1	1
miss	y	y	y		y	y			y

禁止任何商业使用
出退群：550150534

缺页中断次数 6 次
缺页中断率 6/9 = 66.7%
4104/2048 = 1
4014%2048 = 1966
1号页对应页框为20
故物理地址为 20*2048+1966 = 42926

27

1)
EBP 内容是 R[esp] - 4 = 0xbf0007fc
2)
缺失指令是 `leal -8(%ebp), %eax`
x的虚拟地址是 0xbf0007fc - 4 = 0xbf0007f8
y的虚拟地址是 0xbf0007fc - 8 = 0xbf0007f4
3)
返回值是 -12（截断）

scanf 将输入复制到 x，y 的值所指示的位置，而不是 x，y 的内存位置，x，y 实际未定值，可以是任何数。此时会出现：

1. 值 x，y 所指示的地址越界，scanf 执行出错，funct 函数没有运行完就被操作系统终止。
2. 值 x，y 所指示的地址没有越界，scanf 顺利执行，funct 函数执行结束返回非预期的结果。

4)
除数是32位的带符号除法，被除数将会被符号扩展到64位并且低高32位分别存放在 eax 和 edx 中
该右移指令是置被除数的高32位为0（非负整数）或1（负整数），即符号拓展。
5)
test 中的 result 是局部变量不在符号表中，main 中的 result 是全局弱符号且是唯一的，所以不会发生链接错误
main 中的 result 被分配在可读写数据段
test 中的 result 被分配在用户栈段
6)
因为 x，y 的地址分别为 0xbf0007f8、0xbf0007f4处于同一个主存块中
而在取 y 的值之前已经将 x 所在主存块缓存，所以不会发生数据缺失

cache地址结构如下
块内偏移为 log(64B/1B) = 6
组号为 log(64) = 6
标识号为 32 - 6 - 6 = 20
取 y 地址低12位 0111 1111 0100，得组号 0111 11，即第31组

28

1)
有6个C类网段
233.1.1/24
233.1.2/24
233.1.3/24
233.1.7/24
233.1.8/24
233.1.9/24
2)
A、B、C 类 IPv4 地址的划分大小固定，不能灵活适应各种不同大小的网段，容易造成网络地址浪费

禁止任何商业使用
划水群：550150534

3)
子网1: 223.1.1.0/26
子网2: 223.1.1.128/25
子网3: 233.1.1.192/28
4)

目的网段	下一节点
233.1.1.0/26	subnet1
233.1.1.128/25	subnet2
233.1.1.192/28	subnet3
233.1.9.0/24	R1
233.1.7.0/24	R1
233.1.3.0/24	R3
other	R2

5)

目的网段	下一节点
233.1.1.0/24	R1
233.1.8.0/24	R3
233.1.7.0/24	R3
233.1.3.0/24	R3
other	R2

路由汇聚是只用单一前缀向其他网络通告所拥有的多个网段的方法。
6)
CIDR 是只形如 a.b.c.d/x 的地址表示方式，x 代表网络段的长度。
该学校最少需要6个 CIDR 地址块
7)
NAT 对外部网络屏蔽内部网络的细节，而在内部网络中使用私有地址为内部设备分配 IP 地址。
当一个从内部设备发来的数据报到达 NAT 时，NAT 将改写该数据报网络层源地址和运输层端口号再按一般路由器的行为转发，添加相关条目记录，然后在改写后的端口号上监听此数据报的回复，把回复报文的网络层目的地址和运输层端口改写为原内部设备地址和端口号。
从而可以在不同的子网中复用私有地址，达到不需要申请新的网络地址而扩容的目的。
8)
候选路由器是 R2，因为 NAT 一般位于出子网的边界部分，边界路由器 R2 最符合此条件。

版权所有，禁止一切商业用途。

禁止任何商业使用
劝退群：550150534