四/7

0	1	2	3	4	5	6	7	8	9	10	11	12	13
78		15	3		57	45	20	31		23	36	12	

$$ASL_{sure} = \frac{1+1+1+1+1+1+1+4+1+2}{10} = \frac{7}{5}$$

$$ASL_{unsure} = \frac{2+1+3+2+1+5+4+3+2+1+5+4+3}{13} = \frac{36}{13}$$

(2)

0	1	2	3	4	5	6	7	8	9	10	11	12	13
78	15	3			57	45	20	31	36	23		12	

(3, 5)

$ASL_{sure} = \frac{5+3+8\times1}{10} = \frac{8}{5}$

四/9. (1) $\alpha = \frac{8}{m} \le 0.5$ $m \ge 16$ $m = 4k+3$, m为质数

$\frac{1}{1-\alpha} \le 2$ $1-\alpha \ge 0.5$

$m = 19$ $Hash(X) = X\%19$

(2) 25: 6 1

40: 2 1

11: 11 1

97: 2→3 2

59: 2→3→6→11→18 5

30: 11→12 2

87: 11→12→15 3

73: 16 1

$ASL_{sure} = \frac{1+1+1+2+5+2+3+1}{8} = 2$

五/3

```
// 填充位向量
void fillBitVector(Node* head, std::vector<bool>& bitVector, int M) {
    for (int i = 0;i < M + 1;i++){
        bitVector.push_back(false);
    }
    while (head != nullptr) {
        bitVector[head->data] = true;
        head = head->next;
    }
}

// 并集
void unionSet(const std::vector<bool>& bitA, const std::vector<bool>& bitB, std::vector<bool>& bitC) {
    for (int i = 0; i < bitA.size(); ++i) {
        bitC[i] = bitA[i] || bitB[i];
    }
}
```

```
// 交集
void intersectSet(const std::vector<bool>& bitA, const std::vector<bool>& bitB, std::vector<bool>& bitC) {
    for (int i = 0; i < bitA.size(); ++i) {
        bitC[i] = bitA[i] && bitB[i];
    }
}

// 差集
void differenceSet(const std::vector<bool>& bitA, const std::vector<bool>& bitB, std::vector<bool>& bitC) {
    for (int i = 0; i < bitA.size(); ++i) {
        bitC[i] = bitA[i] && !bitB[i];
    }
}
```

```
// 从位向量构建链表
Node* buildListFromBitVector(const std::vector<bool>& bitVector) {
    Node* head = nullptr;
    Node* tail = nullptr;
    for (int i = 0; i < bitVector.size(); i++) {
        if (bitVector[i]) {
            Node* newNode = new Node(i);
            if (!head) {
                head = newNode;
                tail = newNode;
            } else {
                tail->next = newNode;
                tail = newNode;
            }
        }
    }
    return head;
}
```

五/5

```
template<typename T>
Node<T>* search(Node<T>* head, Node<T>*& p, const T& key) {
    // 如果链表为空或p为空，从头开始搜索
    if (!head || !p) {
        p = head;
    } else {
        // 从p开始向前搜索
        while (p && p->data > key) {
            p = p->prev;
        }
        // 从p开始向后搜索
        while (p && p->data < key) {
            p = p->next;
        }
    }
    // 如果找到关键码，返回p；否则返回nullptr
    if (p && p->data == key) {
        return p;
    }
    return nullptr;
}
```