严锦 1700011049

1.

(1).

0	1	2	3	4	5	6	7	8	9
7	14		8		11	30	18	9	

(2).

检索成功的平均长度: (1+1+1+1+3+3+2) / 7 = 12/7, 检索失败的平均长度: (3+2+1+2+1+5+4) / 7 = 18/7.

2.

```
vector<int> Intersection(const vector<int>& A, const vector<int>& B) {
    sort(A.begin(), A.end());
    sort(B.begin(), B.end());
    vector<int> ans;
    int i = 0, j = 0;
    while(i < A.size() && j < B.size()) {
        if (A[i] == B[j]) {
            int tmp = A[i];
            ans.push_back(tmp);
            while (i < A.size() && A[i] == tmp) i++;
            while (j < B.size() && B[i] == tmp) j++;
        }
        else if (A[i] < B[j])
            i++;
        else if (A[i] > B[j])
            j++;
    }
    return ans;
}
```

排序时间复杂度为$\mathcal{O}(n\log(n))$，归并时间复杂度为$\mathcal{O}(n)$，总时间复杂度为$\mathcal{O}(n\log(n))$

3.

对当前找到的所有连续整数集合，分别以集合中的最小元素和最大元素建立hash索引。遍历到新元素k时，如果k-1为末位数的集合和k+1为首位数的集合均位于hash表中，则将hash[k-1], k, hash[k+1]合并成新集合，再从hash表中删除hash[k-1], hash[k+1]，并更新索引。最终在hash表中找出最大的集合即可。时间复杂度为$\mathcal{O}(n)$

4.

使用数组a[L]储存文本数据。

MOVE k即定位到a[k]，时间复杂度为$\mathcal{O}(1)$.

PRINT n即输出a[k]到a[k+n]的数据，时间复杂度为$\mathcal{O}(n)$.

PREV即定位到a[k-1]，时间复杂度为$\mathcal{O}(1)$.

NEXT即定位到a[k+1]，时间复杂度为$\mathcal{O}(1)$

5.

使用块状链表。将长度为L的数据分为$\sqrt{L}$个长度为$\sqrt{L}$的线性表，再将这些线性表链接形成链表，同时使用额外的sum数组记录每个从起点到链结的总数据长度。

MOVE k，先从sum数组中查询到小于k的最大上界，再从相应的链结的线性表中继续定位到k的位置，时间复杂度为$\mathcal{O}(\sqrt{L})$

PRINT n，同**4**，时间复杂度为$\mathcal{O}(n)$

PREV, NEXT, 同**4**，时间复杂度为$O(1)$

INSERT n, s, 若插入s后，仍然不超过链结的最大长度，则直接在线性表中插入。若插入s后超出了链结的最大长度，则将超出的部分单独建立新的链结，插入在当前链结的后面。时间复杂度为$\mathcal{O}(n)$

DELETE n，若链结中的元素数大于n，则直接从线性表中删除。若链结的线性表中的元素不够删除，则删除该链结，再从该链结的前一个链结的线性表的尾部删除剩余的元素。时间复杂度为$\mathcal{O}(n)$

每次操作完，都要重新更新sum数组。同时可以合并元素数量不足$\sqrt{L}$的结点，以减少检索长度。