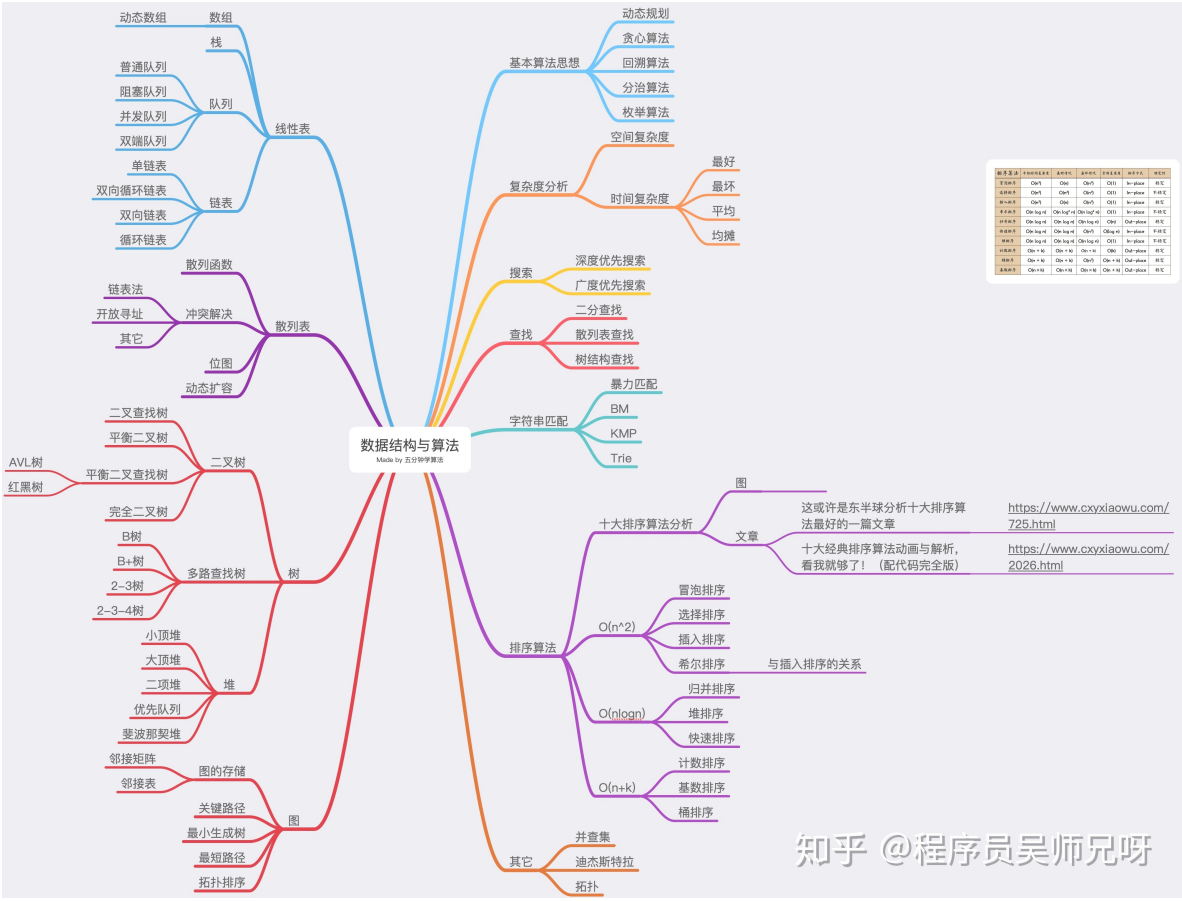
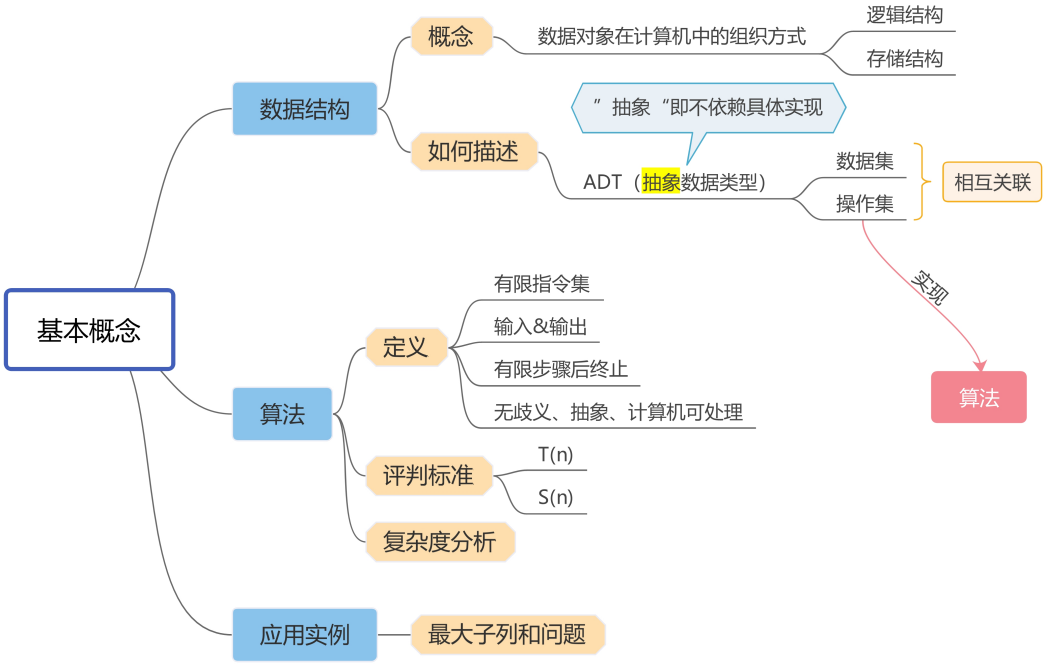
Data Structure

知识网络



基本概念



复杂度渐进表示法

$$T(n) = O(f(n)) =\gg T(n) \leq Cf(n)$$

$$T(n) = \Omega(g(n)) =\gg T(n) \geq Cg(n)$$

$$T(n) = \Theta(h(n)) =\gg T(n) = O(h(n))\&\&T(n) = \Omega(n)$$

T(n) = n! 的算法不可用

复杂度分析技巧

- 若两段算法分别有复杂度$T_1(n) = O(f_1(n))$和$T_2(n) = O(f_2(n))$，则：

$$T_1(n) + T_2(n) = max(O(f_1(n)), O(f_2(n)))$$

$$T_1(n) \times T_2(n) = O(f_1(n) \times f_2(n))$$

- 若$T(n)$是关于n的k阶多项式，那么

$$T(n) = \Theta(n^k)$$

- 一个for循环的时间复杂度等于循环次数乘以循环体代码的复杂度
- if-else结构的复杂度取决于if的条件判断复杂度和两个分枝部分的复杂度，总体复杂度取三者中最大的

应用实例：最大子列和问题

给定n个整数的序列$A_1, A_2, ..., A_n$, 求函数$f(i, j) = \max(0, \sum_{k=i}^{j} A_k)$的最大值。

法一

遍历法，找出所有子列的和，求最大值

```
int MaxSubSeqSum_1(vector<int> A) { // 遍历法，找出所有子列的和，求最大值
    int maxSum = 0, tempSum;
    for (int i = 0; i < A.size(); i++)
    {
        for (int j = i; j < A.size(); j++)
        {
            tempSum = 0;
            for (int k = i; k < j; k++)
            {
                tempSum += A[k];
            }
            if (tempSum > maxSum) maxSum = tempSum;
        }
    }
    return maxSum;
}
```

法二

优化法一，去掉k的循环

```
int MaxSubSeqSum_2(vector<int> A) { //遍历法改进，去掉k的循环
    int maxSum = 0, tempSum;
    for (int i = 0; i < A.size(); i++)
        {
            tempSum = 0;
            for (int j = i; j < A.size(); j++)
            {
                tempSum += A[j];
                if (tempSum > maxSum) maxSum = tempSum;
            }
        }
    return maxSum;
}
```

法三

分而治之（递归式）

```
int MaxSubSeqSum_3(vector<int> A) {  //分而治之，递归式
    return divideAndConquer(A, 0, A.size() - 1);
}

int divideAndConquer(vector<int>& A, int left, int right) {
    int maxLeftSum = 0, maxRightSum = 0,
        maxLeftBorderSum = 0, maxRightBorderSum = 0, maxBorderSum = 0;
    int tempSumL = 0, tempSumR = 0;
    if (left == right)  // 递归出口
    {
        if (A[left] > 0) return A[left];
        else return 0;
    }
    int center = (left + right) / 2;
    maxLeftSum = divideAndConquer(A, left, center); // 左区域子列和最大值
```

```
    maxRightSum = divideAndConquer(A, center + 1, right);  // 右区域子列和最大值
    for (int i = center; i >= left; i--)  // 由分界线向左扫描找最大值
    {
        tempSumL += A[i];
        if (tempSumL > maxLeftBorderSum) maxLeftBorderSum = tempSumL;
    }
    for (int i = center + 1; i <= right; i++)  // 由分界向右扫描找最大值
    {
        tempSumR += A[i];
        if (tempSumR > maxRightBorderSum) maxRightBorderSum = tempSumR;
    }
    maxBorderSum = maxLeftBorderSum + maxRightBorderSum;
    return maxLeftSum > maxRightSum ? (maxLeftSum > maxBorderSum ? maxLeftSum : maxBorderSum)
        : (maxRightSum > maxBorderSum ? maxRightSum : maxBorderSum);
}
```

法四

在线处理

```
int MaxSubSeqSum_4(vector<int> A) {  // 在线处理
    int tempSum = 0, maxSum = 0;
    for (int i = 0; i < A.size(); i++)
    {
        tempSum += A[i];
        if (tempSum > maxSum) maxSum = tempSum;
        else if (tempSum < 0) tempSum = 0;
    }
    return maxSum;
}
```