> CH1

- 数学模型不是 ODE、PDE，而是 linear list, tree, graph
- Time Complexity：算法中操作重复的次数 / 算法所需时间
  - $O(1) < O(\log n) < O(n) < O(n^{const}) < O(const^n) < O(n!) < O(n^n)$
  - 和输入有关，因此通常指最坏情况
- Space Complexity：算法所需空间
- 作业 easy

> CH2

- topology structure and storage structure
  - topology structure → list, tree, graph
  - storage structure → sequence（地址连续）, Link（地址不连续）

```
typedef struct {
    int data
    * p_next
}
```

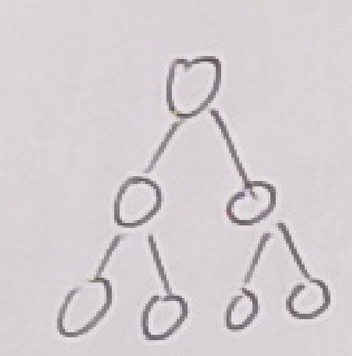
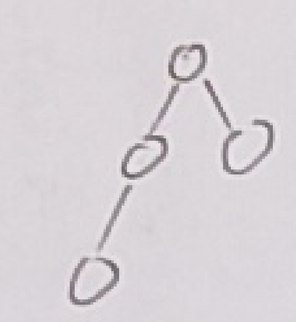
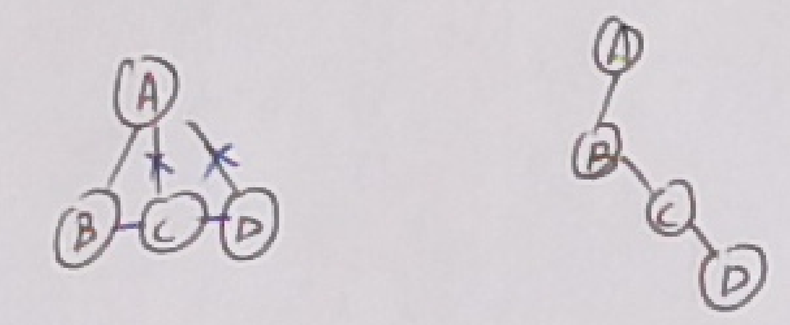
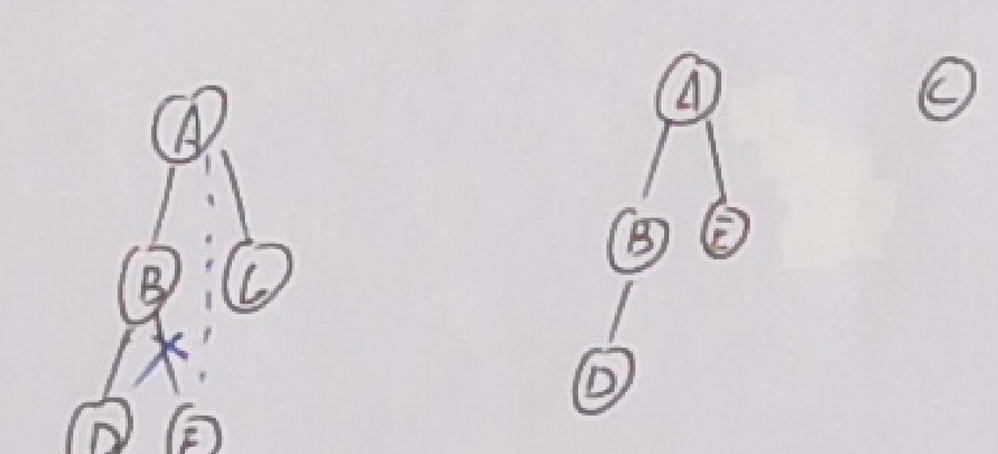
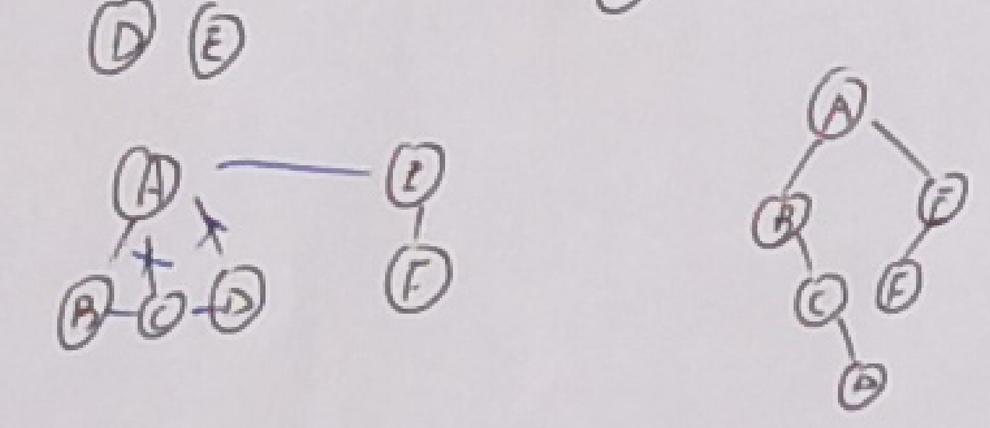
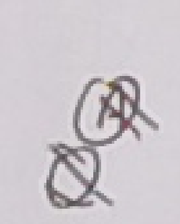
- sequence：insert 和 delete 是 O(n)
- Link：头结点是为了让对第1个节点的操作和之后的统一。

> CH3

- stack：typedef struct 里指定了 size

```
typedef struct {
    * top
    * base
    size
}
```

> CH6

- 度：结点的子树个数（二叉树中，0.1.2 三种情况）
- 树的度：最多的次数
- 根为第一层
- 满 binary tree  完全 binary tree 
- 树 → 二叉树  // left-child right-sibling
- 二叉树 → 树 
- forest → binary tree 
- binary tree → forest  易
- traverse：定义为每个节点只被访问一次
  - // 深度优先：递归
  - 广 ：queue
- Pre Order：printf → 左 → 右
- In Order：左 → pr → 右
- Post Order：左 → 右 → pr

> CH7

- 边表

```
typedef struct edge_node {
    int adjacent_vertex;
    edge_node * p_next,
    int weight;
} edge_node;
```

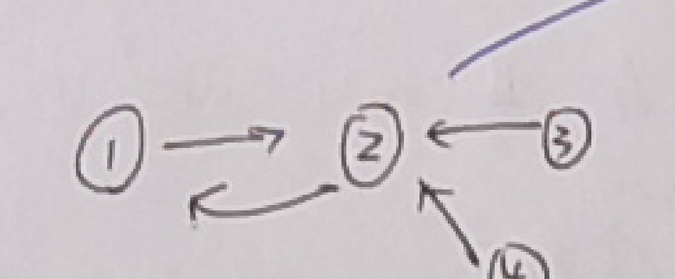
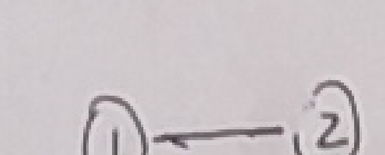
- 顶点表

```
typedef struct vertex_node {
    int information;
    edge_node * p_first;
} vertex_node, adjacency_List [100];
```

（100 → MAX_NUMBER_OF_VERTEX）

- 图

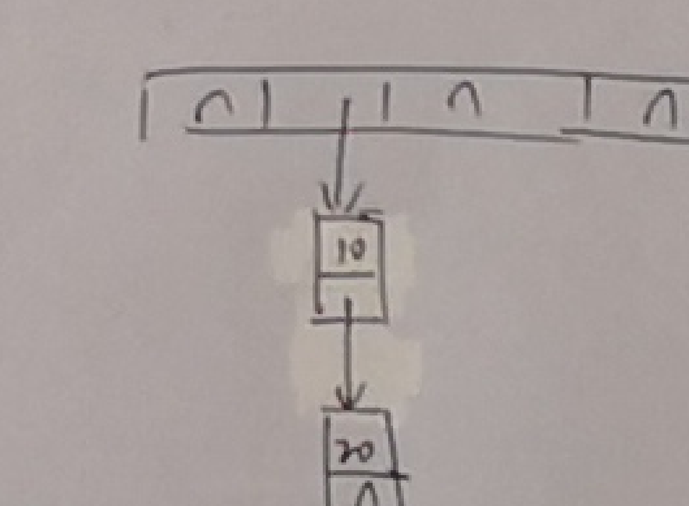
```
typedef struct My_Graph {
    adjacency_List adjacency_List;
    int number_of_vertex, number_of_edge;
    int flag_direct;
} My_Graph;
```

- 有向  ②的入度为3，出度为1　$n \cdot (n-1)$
- 无向  $\frac{n(n-1)}{2}$
- 连通：$\forall (i,j)\ \exists\ i \to j$
- 强连通：$\forall (i,j)\ \exists\ i \to j$ 且 $\exists\ j \to i$

- Activity On Vertex ✓　　微积分 → 复变 → PDE
- Activity On Edge
  - 最长 path ✓　　○ —10h→
  - 最短 path：Dijkstra
    - Visited　　Unvisited = [A B C D E]
    - 到点了更新

> CH9：Search Table 是待查找的集合；ASL 是 $\langle C \rangle = \sum p_i c_i$（比较次数）

- 静、动
  - 顺序：$ASL = \frac{1}{n}\sum_{i}^{n} i = \frac{n+1}{2}$　// 哨兵：找不到时 return 0
  - binary search：Search Table 必须↑或↓
  - binary search 的判定树：易
- 动态 binary search Tree：左 < 根 ≤ 右　// 中序遍历可得↑
  - 分块：第1块 < 第2块 < 第3块
- Hash
  - hash：关键字 ↦ 地址
  - 解决冲突：
    - Ⅰ：线性 $d = 1, 2, 3 \cdots m-1$
    - 二次 $d = 1^2, -1^2, 2^2, -2^2 \cdots k^2, -k^2$
  - 解决 Ⅱ：链表法



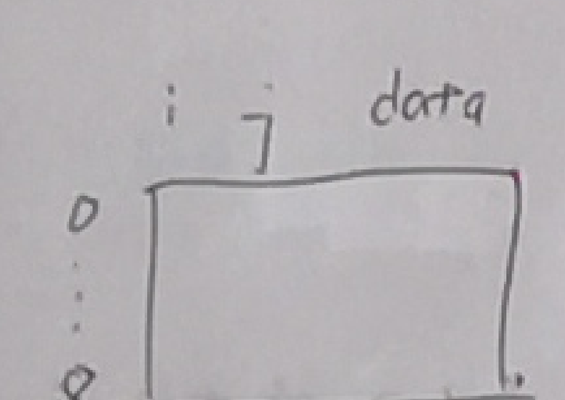
2

- 中缀 a+b
- 前缀/波兰式 +ab
- 后缀/逆波兰式 ab+

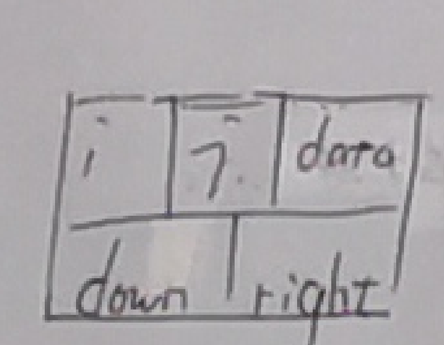
- recursion: 时空消耗大, 但是什
- 循环队列
  - 入队 Q.rear = (Q.rear+1) % M
  - 出队 Q.front = (Q.front+1) % M → 可考不看
  - 队空 Q.front == Q.rear
  - 队满 (Q.rear+1) % M == Q.front

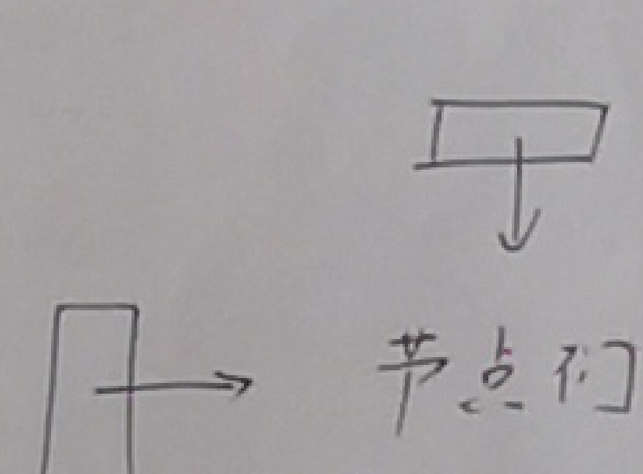
> CH4
- '\0' 串长为?
- '' 串长为0 → // 结尾的 '\0' 不算长度

> CH5
- C 以行序为主序: a_{00} a_{01} … $a_{0,n-1}$ a_{10} … $a_{1,n-1}$ … $a_{m-1,n-1}$
- sparse matrix
  - 三元



十字链表





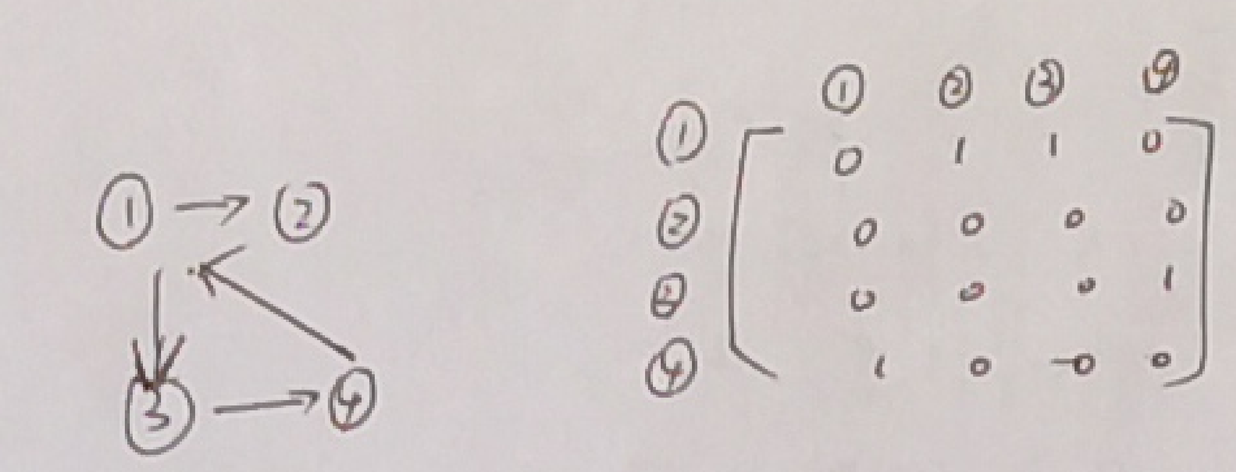
节点们

4

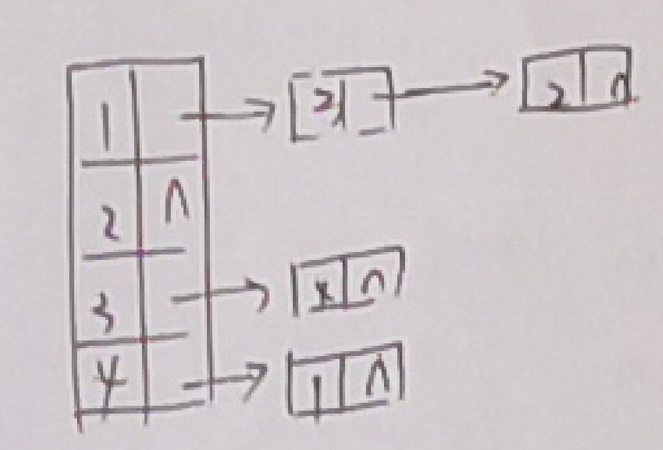
- Huffman
  - wpl = $\sum_{k=1}^{n} w_k l_k$ 最小 (只算叶子)
- 森林的遍历
  - TH
    - 先 = 先序对应的二叉树
    - 中/后 = 中~

6.

- 邻接矩阵

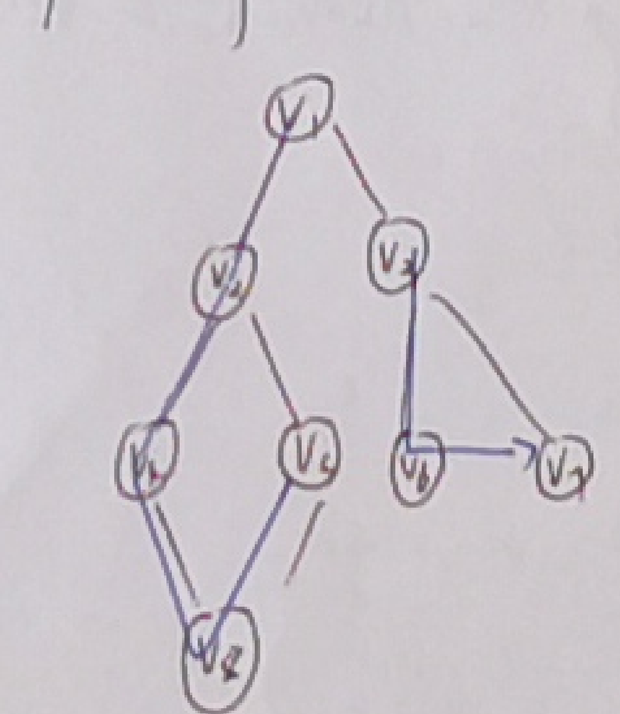


- 邻接表

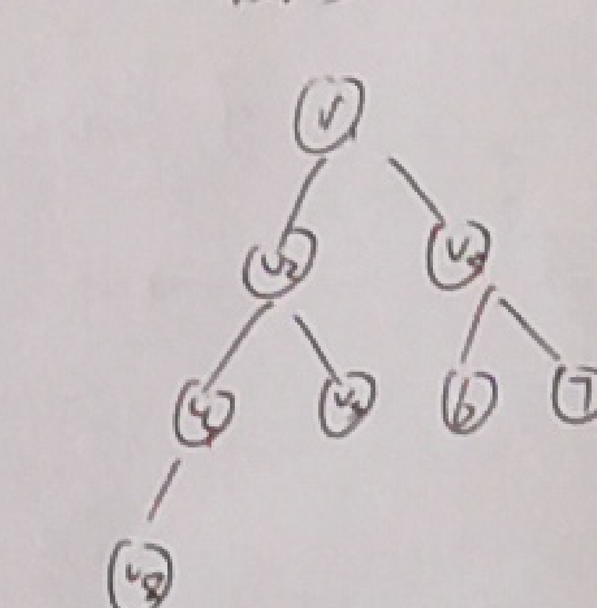


- spanning tree

DFS



BFS

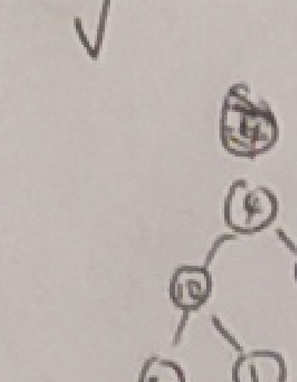


- minimum spanning tree: n个城市, 只需 (n-1) 条边
  1. Prim: 适用于边多。与边数无关
     - 每次都选权值最小的边 且 u∈U v∈V-U
  2. Kruskal: 适用于顶点多。与顶点数无关
     - 每次~ 且不能有回路

8

> CH 10 // Hill、quick、heap

- insert: 扑克牌
- Shell: 军训分组: 5 3 1
- Bubble: ✓
- quick: ① 选Pivot 且划分 ② 快排左 ③ 快排右 — 2 1 1̄ 3 → 1̄ 1 2 3
- select: ✓ — 0 1 2 3 4 / 4 10 3 5 1
- heap:



4 10 4̄ 5 → 10 5 4̄ 4

- merge: 49 38 → 38 49; 65 97 → 65 97