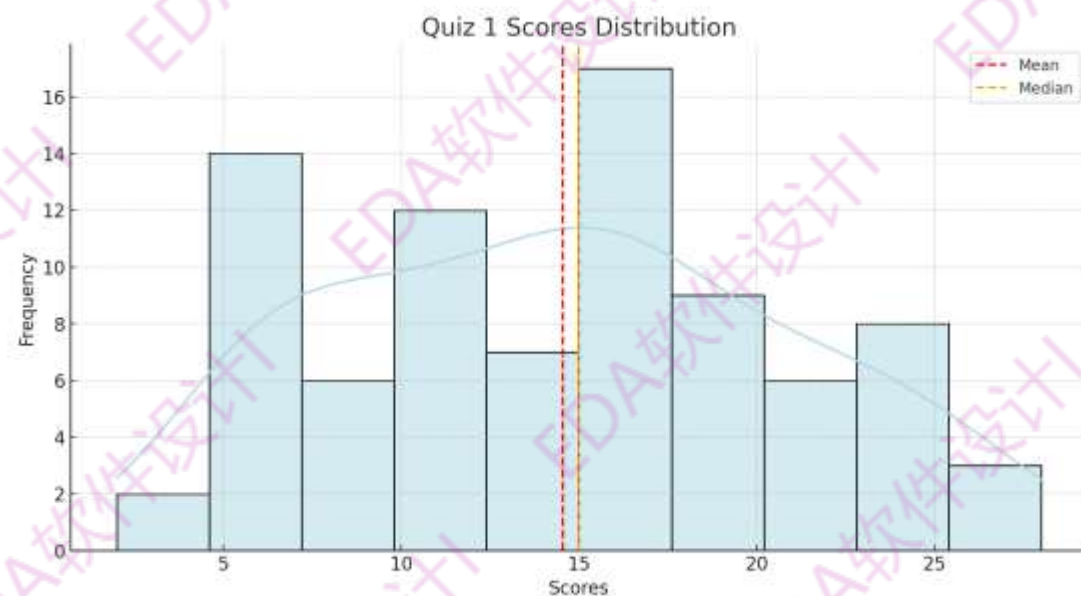
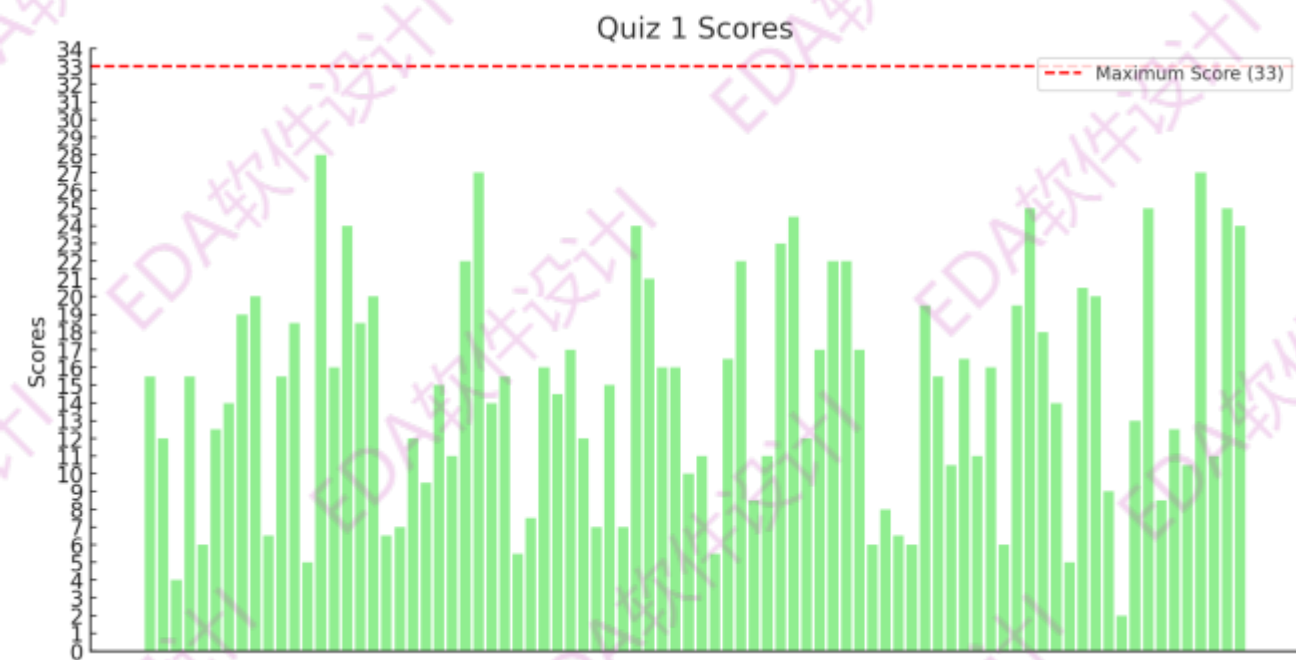


EDA 软件设计 I

Lecture 13

Check Course Site Frequently



附送概率论甜点：

中心极限定理：无论原始数据分布如何，只要样本数量足够大，样本均值的分布趋向于正态分布

Why Leetcode: 内化所学

1. Make sure you understand

2. 实践强化理解

- 算法知识只有在实际练习中才能巩固

3. 逐步提升思维能力，获得成就感

- LeetCode上的题目由浅入深，适合初学者到高手
- 提高逻辑思维能力和解决问题的技巧
- 逐步进阶的过程感受到自己的进步，获得成就感

4. 构建良好的问题解决习惯

- 每道题目通常需要思考时间、空间复杂度，这会让学生养成优化代码的习惯，有助于他们写出高效、可扩展的代码

• Leetcode题目来源:

1. 工业界实际遇到的问题，特别是技术面试中的经典问题
 - 数据结构、算法设计、系统设计
2. 计算机科学和算法领域的经典研究问题
3. 实际开发场景的抽象与简化
4. 来自编程竞赛的挑战题目
 - ICPC、Google Code Jam

The best way to learn something: use and spray it

使用法则：遵循由易到难的规律

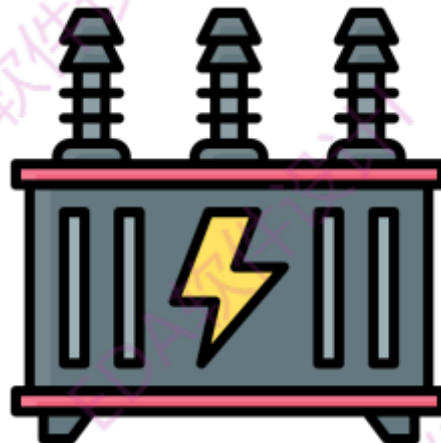
1. [两数之和](#)：easy
2. [104. 二叉树的最大深度](#)：easy, SAP面试题
 - SAP: 企业资源规划 (ERP) 软件和其他面向企业的解决方案
3. [45. 跳跃游戏 II](#)：medium, Google面试题
4. 挫败感management: 有一些easy题是披着easy皮的难题

“编程的挫折感正是思想与机器对话的磨砺，每一次卡顿都是在构建更深层的理解，每个错误都是通往优雅解法的必经之路”

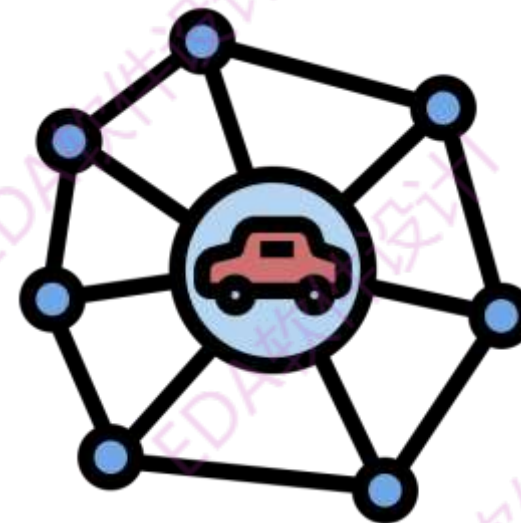
Review: 最小生成树 (MST) 的多种应用



设计最优的网络拓扑结构



最少的电缆长度覆盖所有的城市或建筑




最少的建设成本连接不同的城市或地点

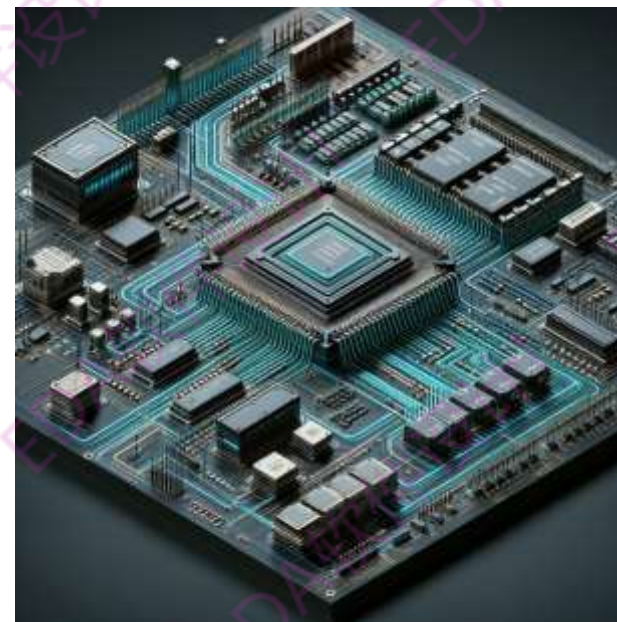
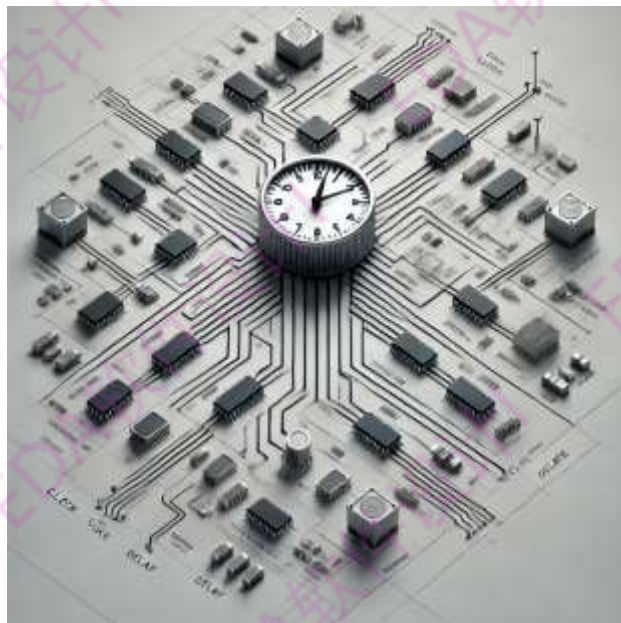
Review: 优化连接成本问题

最小生成树 (Minimum Spanning Tree) 的目标:
用最低的成本将所有节点连接起来





- ◆ 连通、连接
- ◆ 最优化 (min、max)、成本最小

Review: MST for EDA

 **时钟树综合目标:** 在保证时钟延迟最小和时钟抖动可控的前提下, 最小化时钟布线的总长度, 从而减少功耗并提高时钟网络的性能



布线目标:

-  减少信号线长度
-  降低功耗
-  避免布线拥塞
-  优化关键信号 (如时钟、数据、控制信号) 的传输路径

More MST Applications



生物学

应用：生成树和最小生成树可用于生物学构建系统发育树，以表示物种或基因之间的进化关系



物流和供应链管理

应用：在物流和供应链管理中，企业需要规划最优的运输网络以连接仓库、供应商和消费者



网络路由协议

应用：在计算机网络中，生成树协议（Spanning Tree Protocol, STP）是用于防止网络中出现环路的路由协议

Note: MST与TSP（旅行商问题）的区别

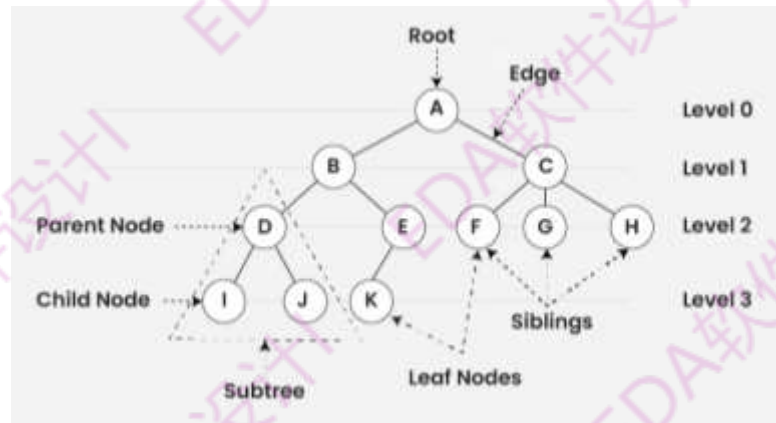
- MST: 寻找一棵连接图中所有节点的生成树，使总边权重最小，不需要回到起点，也没有闭环
- TSP: 在图中找到一个从起点出发、访问所有节点并回到起点的最短路径，形成闭环

最小生成树概念

■ 01

树

- 无环连通图
- n 个节点和 $n - 1$ 条边



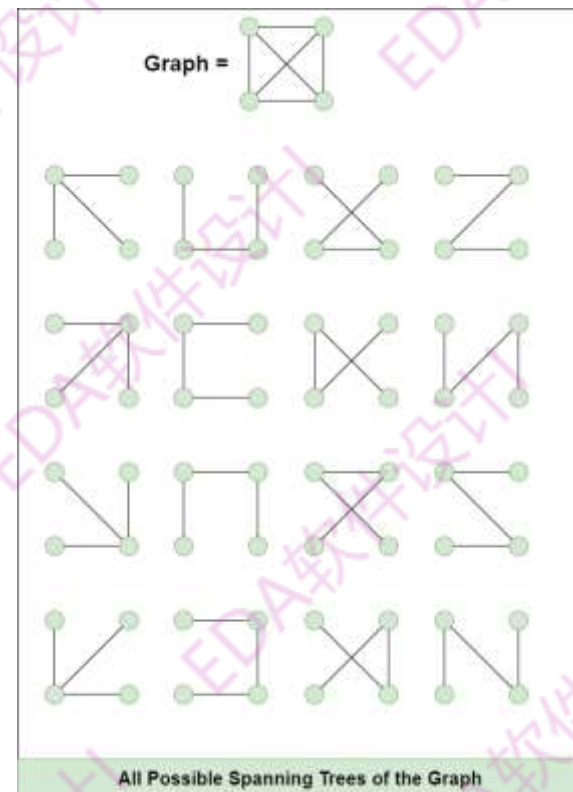
■ 02

生成树

- 一个**无向连通图**的子图

? 有向图有没有生成树?
? 不连通的图有没有生成树?

- 包含了图中所有顶点
- 但只包含能够**使图保持连通的最少数量的边**



■ 03

最小生成树

- 一棵生成树
- 生成树中的所有边的权重之和最小

? 最小生成树是唯一确定的吗?



寻找最小生成树的算法



1. Kruskal (克鲁斯卡尔) 算法

- ◆ Created in 1956 by Joseph Kruskal [美国数学家和统计学家]
- ◆ 由Joseph B. Kruskal 在1956年发表的论文《On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem》中首次提出

2. Prim (普林姆) 算法

- ◆ Created in 1957 by Robert C. Prim [美国数学家]
- ◆ Robert C. Prim在1957年提出, 并发表在论文《Shortest Connection Networks and Some Generalizations》中
- ◆ 虽然Prim最早提出了该算法, 但在1960年代, Edgar Dijkstra也独立发现了类似的算法, 于是该算法有时也被称为Dijkstra-Prim算法

Both are Greedy Algorithms!

贪心算法 (Greedy Algorithm)

- 贪心策略: 在**每一步骤都选择对当前最有利的选项 (局部最优)**, 希望通过局部最优解最终得到**全局最优解**
 - 每次只做出对当前问题的最优选择, 而不考虑未来可能发生的情况



找零问题：假设我们要找零27元，而手头上有的面值有20元、10元、5元和1元，目标是使用尽可能少的纸币来满足27元的找零需求



贪心算法解法

选择当前面额最大的纸币：从20元开始，因为这是小于等于27元且面额最大的纸币。

更新总额：用掉一个20元，还需找零 $27 - 20 = 7$ 元。

重复此过程直到找零总额为0

Start: Amount to change is 27

Choose max (20,10,5,1) that \leq remaining

Amount - 20 = 7

Choose max (20,10,5,1) that \leq remaining amount

Amount - 5 = 2

Choose max (20,10,5,1) that \leq remaining amount

Amount - 1 = 0

Done: Amount is zero

水果收集问题:

假设你去果园苹果、橙子、葡萄，
每种水果的价值不同
苹果每个价值5元（重1斤），橙子
每个价值3元（重0.5斤），葡萄每
串价值8元（重2斤）。

你带的篮子只能装10公斤的水果，
不超过这个重量限制。

**目标是在不超过10公斤的前提下，
收集尽可能多的总价值**

使用贪心策略解决问题

在这个问题中，贪心策略可以帮我们
找到一个不错的解决方案，虽然
不一定是最优解，但通常会比较接
近最优解。我们可以这样考虑：每
次都挑选“价值密度”（价值/重量）
最高的水果，因为这意味着这个水
果的单位重量提供了最高的收益。

1. 计算每种水果的价值密度:

1. 苹果: $5\text{元}/1\text{斤} = 5\text{元/斤}$
2. 橙子: $3\text{元}/0.5\text{斤} = 6\text{元/斤}$
3. 葡萄: $8\text{元}/2\text{斤} = 4\text{元/斤}$

贪心选择:

**按照价值密度从高到低选择水果，
直到篮子装满为止**

1. 第一步: 选择橙子（6元/斤），
因为它的价值密度最高。
2. 如果篮子容量更小，我们装完橙
子后，再依次选择密度第二高的苹
果（5元/斤），最后选择葡萄（4
元/斤）。

Start: Basket capacity
is 10 kg

Choose
Orange if
available and
space allows

If space remains after Oranges

Choose Apple if
space allows

If space remains after Apples

Choose Grape if
space allows

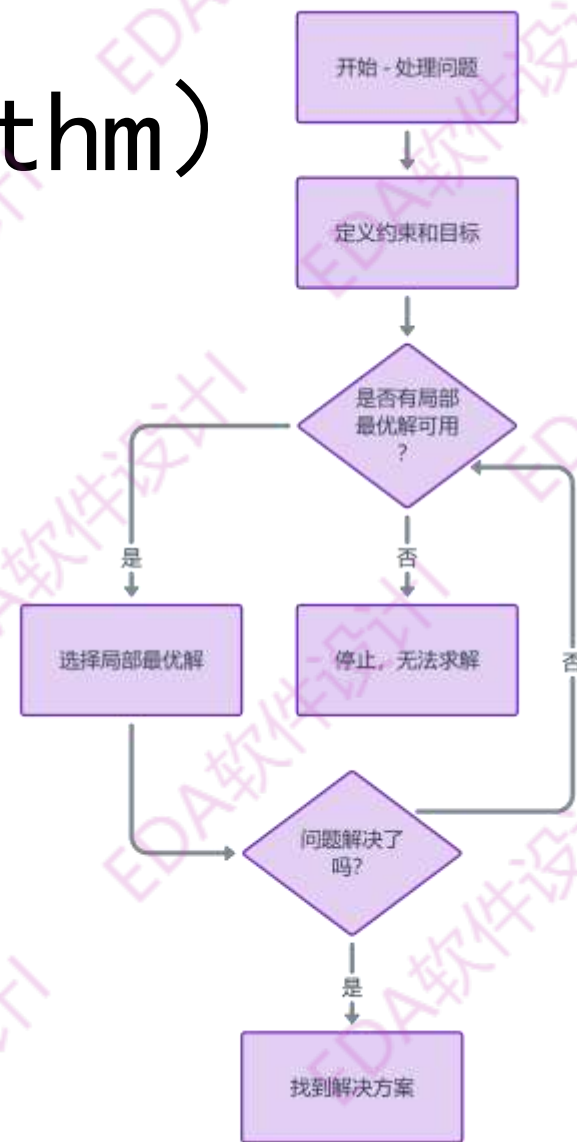
No space left or no more Grapes

Done: Basket full or no
more fruits

贪心算法 (Greedy Algorithm)

- **不可回溯性**：一旦做出某个选择，就不能回溯修改
- 步骤：
 - ① **初始化**：从问题的初始状态出发
 - ② **贪心选择**：在每一步中，做出当前最优选择（即局部最优解）
 - ③ **检查是否结束**：当问题被解决，或者没有更多选择时，结束算法

❓ 局部最优一定会导致全局最优吗？什么情况下局部最优能保证实现全局最优？



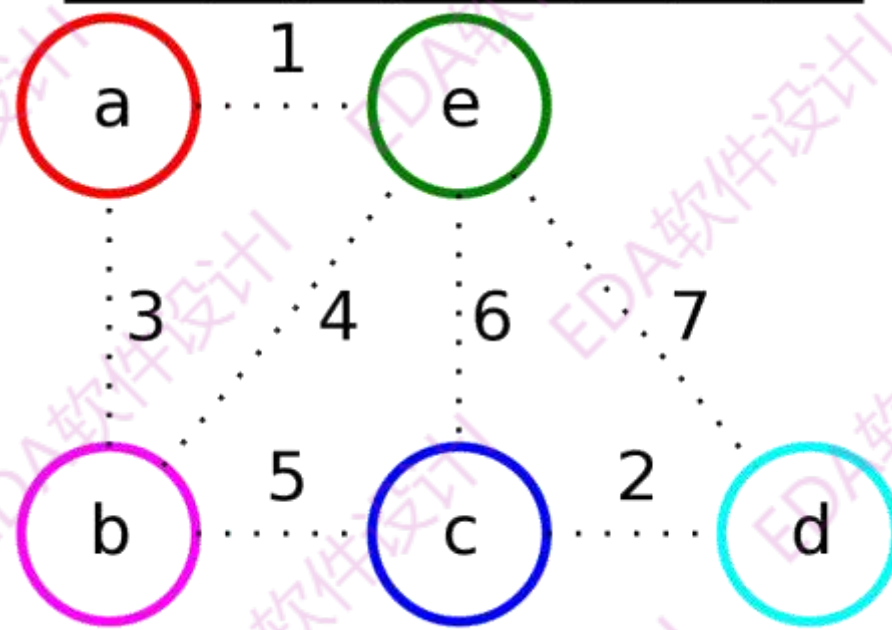
Kruskal 算法

- **贪心策略**：从所有边中，按照权重从小到大的顺序，依次选择边，如果选择的边不构成环路，则将其加入生成树

- **步骤**：

- ① **排序所有边**：首先将图中的所有边按照权重从小到大排序
- ② **逐步选择最小的边**：每次选择当前权重最小的边，并判断加入这条边是否会形成环。如果不会形成环，则将该边加入最小生成树中
- ③ **直到构建完成**：重复上述步骤，直到所有顶点都连接起来，构成最小生成树

Edge	ab	ae	bc	be	cd	ed	ec
Weight	3	1	5	4	2	7	6

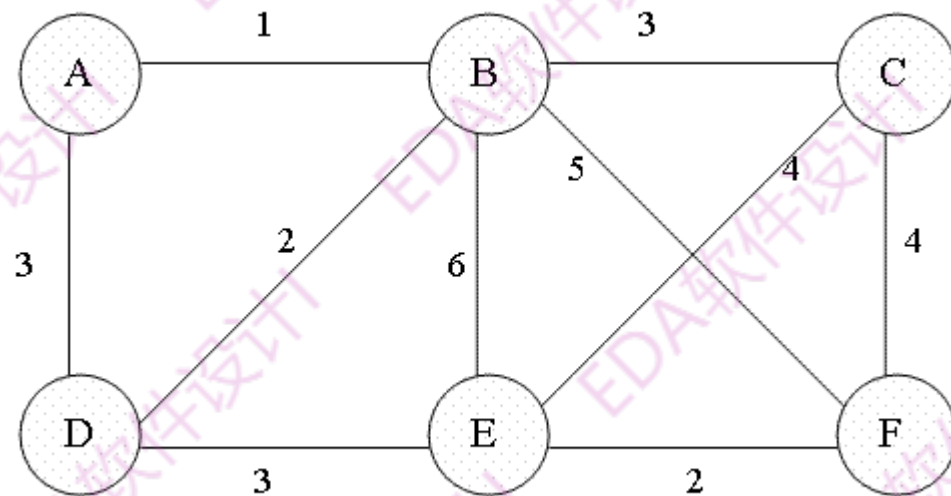


Prim算法

- **贪心策略**：从一个初始顶点开始，不断选择距离已构造生成树最近的顶点，并将其加入生成树中

- **步骤**：

- ① **从任意顶点开始**：从图中任意一个顶点开始，初始化生成树（此时只有一个顶点）
- ② **逐步扩展生成树**：在已加入生成树的顶点集合中，找到一条连接该集合与未加入集合之间的最小边
- ③ **加入新顶点**：将这条边及其连接的顶点加入生成树，重复此过程，直到生成树包含所有顶点



MST Done

Question?

Progress So Far



BFS



DFS



Topological Sort



MST



Next?