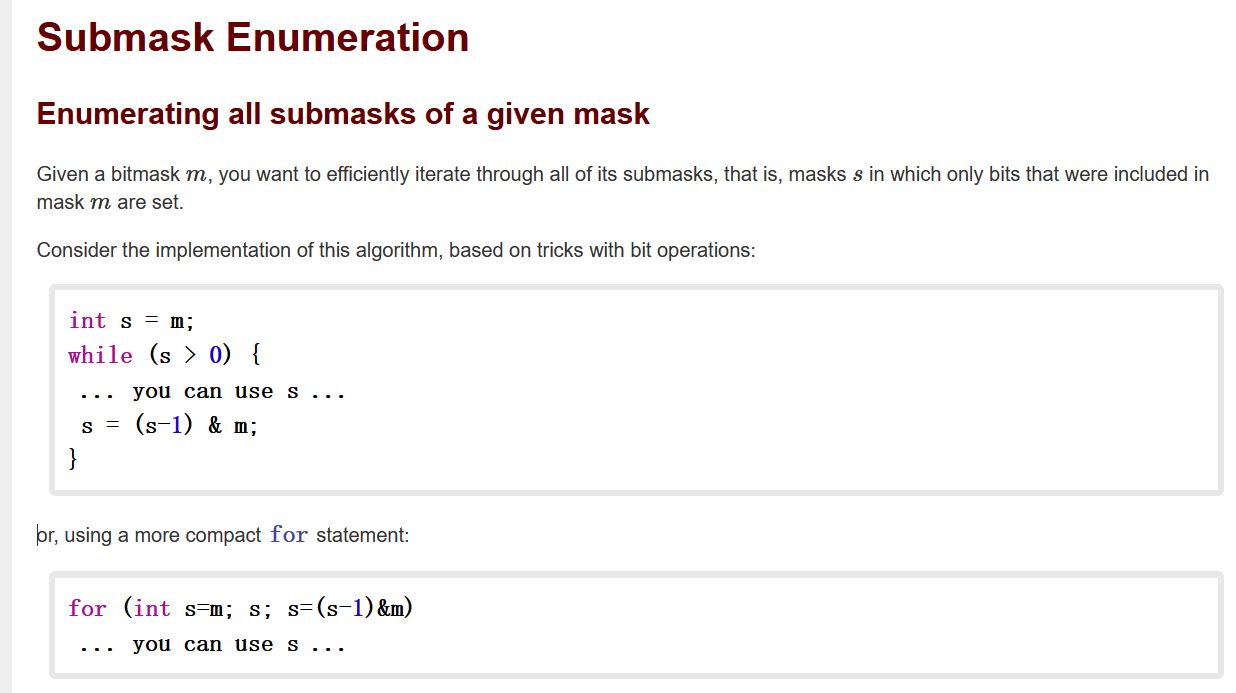
**位运算技巧**

枚举子集



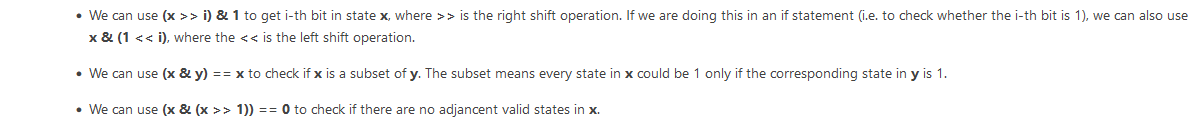
初始m，子集s，补集c = s ^ m 或 (m - s)

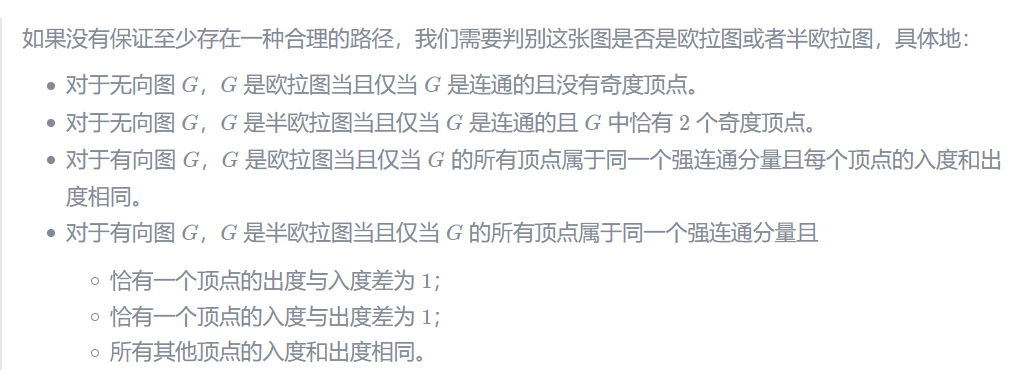
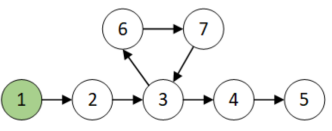
low\_bit : a & -a 比如:a = 14 (1100) a& -a = 100

low\_bit\_pos : ffs(n) 比如ffs(8) = 4 ; ffs(10) = 2

\_\_builtin\_ctz(n) 二进制后面有几个0 ，\_\_builtin\_ctz (10) = 1

<https://cp-algorithms.com/algebra/all-submasks.html>



* 求1-n中每个数字的二进制数中1的个数
  + cnt[0] = 0; for(int i = 0; i < n; ++i) cnt[i] = cnt[i>>1] + (i&1); (批量求)
  + bitset<n>(i).count() (求单个或少量几个)
* 最短路径问题有最优子结构性质(问题的最优解是由子问题的最优解组成)，这是floyed动态规划求最短路的理论基础
* 最长路径问题没有最优子结构性质，是NP完全问题
* Dijkstra可以处理有环的图，不能处理边有负数的图，Bellman-ford和floyed可以，三个算法都不能用于有负权环的图(有负权环的图不存在最小路径)
* Bellman-ford的队列优化算法称为SPFA，Bellman-ford(第n次松弛有更新)和SPFA(一个顶点入队次数大于等于n)都可以判断负权环
  + 详细教程[https://zhuanlan.zhihu.com/p/96621396]
  + 只让当前点能到达的点入队
  + 如果一个点已经在队列里，便不重复入队
  + 如果一条边未被更新，那么它的终点不入队
* **有向无环图**求**最短路**和**最长路**可以用拓扑排序，按照拓扑排序的顺序使用DP，复杂度O(n)，有向无环图的最短路问题没必要用Dijkstra。
  + [https://leetcode-cn.com/problems/parallel-courses-iii/ 并行课程3]
  + [https://leetcode-cn.com/problems/largest-color-value-in-a-directed-graph 有向图的最大颜色]
* 哈密顿路径，只经过图中所有节点一次的路径。
* 欧拉路径，只经过图中所有边一次的路径。 欧拉图，有欧拉回路的图(所有点出度入度一样)，半欧拉图，有欧拉路径但没有回路的图(除起点和终点，所有点出入度一样)。
  + 判断是否存在欧拉图或半欧拉图
    - 
  + **Hierholzer算法 保证有欧拉图或半欧拉图的情况下** DFS遇到一条边后把它删掉，继续往下遍历，遇到**死胡同**时，逆序把这条边加入答案，对于欧拉图，自然成立，对于半欧拉图，**只有在终点会遇到死胡同**。
    - void dfs(int start) {
    - auto& ends = graph[start];
    - while(!ends.empty()) {
    - int end = ends.back();
    - ends.pop\_back();
    - dfs(end);
    - ans.push\_back({start, end});
    - }
    - }
    -  遍历过程中，**3和5都是死胡同**
  + 如果已知存在欧拉路径，则不需要通过回溯来找路径，回溯可行但是效率比较低，容易超时。
  + [https://leetcode-cn.com/problems/valid-arrangement-of-pairs/ 合法重新排列数对]
    - 详细图解: https://leetcode-cn.com/problems/valid-arrangement-of-pairs/solution/zhuan-hua-wei-ou-la-lu-jing-wen-ti-tu-ji-aqmu/
  + [https://leetcode-cn.com/problems/reconstruct-itinerary/ 重新安排行程]
  + [https://leetcode-cn.com/problems/cracking-the-safe/ 破解保险箱]
* 两种存图方式，稠密图适合用**二维矩阵**，稀疏图适合用**邻接表**
* Kruscal对边排序+并查集，适用于稀疏图，prim算法对点集操作，适用于稠密图
  + [https://leetcode-cn.com/problems/min-cost-to-connect-all-points/]
  + [https://leetcode-cn.com/problems/find-critical-and-pseudo-critical-edges-in-minimum-spanning-tree/]
* 最短路径问题
  + 743. 网络延迟时间 [Dijkstra 模板题 如果是稀疏图，用堆的写法较快，如果是稠密图，用枚举的写法比较快]
  + 6032. 得到要求路径的最小带权子图 [求**三个点**组成的最小子图]
  + 有约束条件的最短路 -> Bellman ford算法 -> **在k步之内**，求起点到所有顶点的最短距离 初始化dp[0][start] = 0 start在第0步，到起点start距离是0
    - 787. K站中转内最便宜的航班 [同1928 1.dfs 2.Bellman-ford算法 在k步之内，求起点到所有顶点的最短距离]
    - 1928. 规定时间内到达终点的最小花费 [一条路径上有代价m n，在m <= limit下，求最短的n。1.dfs 2.把m<limit看成在limit步内，求最小n，转为bellman-ford]
* 使用数组{n1, n2, n3....}中的元素不限次，和为sum的方案？爬楼梯问题，求解的是该问题的**排列数**，零钱兑换II问题，求的是该问题的**组合数。**
  + **爬楼梯问题**
    - for 遍历容量
      * for 遍历物品
        + **求排列数**
  + 零钱兑换II
    - for 遍历物品
      * for 遍历容量
        + **求组合数**
* 背包容量正序和逆序，决定一个物品是否可以使用多次，正序可多次使用(完全背包)，逆序只能使用一次(01背包)。
  + [https://leetcode-cn.com/problems/coin-change-2/solution/ling-qian-dui-huan-iihe-pa-lou-ti-wen-ti-dao-di-yo/]
* **背包问题总结**[https://leetcode-cn.com/problems/last-stone-weight-ii/solution/yi-pian-wen-zhang-chi-tou-bei-bao-wen-ti-5lfv/]
  + **01背包** (一个物品最多选一次)
    - 转移方程1：f[i][j]=max(f[i−1][j],f[i−1][j−c[i]]+w[i])
    - ​for (int i = 1; i <= n; i++)
    - for (int j = V; j >= 0; j--)
    - f[j] = max(f[j], f[j - c[i]] + w[i]);
  + **完全背包** (一个物品选任意次)
    - 转移方程1：f[i][j]=max(f[i−1][j−k∗c[i]]+k∗w[i])∣0≤k∗c[i]≤j
    - 转移方程2：f[i][j]=max(f[i−1][j],f[i][j−c[i]]+w[i])
    - ​for (int i = 1; i <= n; i++)
    - for (int j = V; j >= c[i]; j--)
    - f[j] = max(f[j], f[j - c[i]] + w[i]);
  + **多重背包**(一个物品最多选p[i]次)
    - 转移方程：f[i][j]=max(f[i−1][j−k∗c[i]]+k∗w[i])∣0≤k≤p[i]
    - for (int i = 1; i <= n; i++)
    - for (int j = V; j >= c[i]; j--)
    - for (int k = 1; k <= p[i] and k \* c[i] <= j; k++)
    - f[j] = max(f[j], f[j - c[i] \* k] + w[i] \* k);
  + **分组背包**(物品分组，每组的物品最多选一个)
    - 转移方程：f[g][j]=max(f[g−1][j],f[g−1][j−c[i]]+w[i] (物品i⊆组g)
    - for (int g = 1; g <= group\_num; g++)
    - for (int j = V; j >= 0; j--)
    - for (int k = 1; k <= group[i].size(); k++) //物品k属于组i
    - if (j >= c[k]) f[j] = max(f[j], f[j - c[k]] + w[k]);
* 01背包问题，恰好装满的初始化:dp[0] = 0，dp[1.....V] = 无效值(正无穷或负无穷); 不要求完全装满dp[0......V] = 0; 可以这样理解：如果要求背包恰好装满，那么此时只有容量为 0的背包可能被价值为 0的nothing“恰好装满”，其它容量的背包均没有合法的解，属于未定义的状态，它们的值就都应该是 +−∞(无效值，如果题目求最大值，则-∞，反之同理)。如果背包并非必须被装满，那么任何容量的背包都有一个合法解“什么都不装”，这个解的价值为 0，所以初始时状态的值也就全部为 0了。
* 利用dp对数字进行分类。(含有某一数字的分一类)
  + https://leetcode-cn.com/problems/rotated-digits/
* 二分法求最长上升子序列 [俄罗斯套娃问题]
* 面试题 08.13. 堆箱子 [上升子序列的最大数据总和]
* 在动态规划中应用状态机 [学生出勤记录II](https://leetcode-cn.com/problems/student-attendance-record-ii/solution/yi-zhang-tu-rang-ni-kan-dong-ben-ti-de-z-3jn3/)
* 数组中选取子集，达到某一目标 的题目 (我的题解https://leetcode-cn.com/problems/target-sum/solution/by-mountain-ocean-v4g1/)
  + 目标值明确，可以把目标值看出背包容量，数组值看做物品，转成背包问题
    - 416. 分割等和子集 (目标值sum / 2 -> 01背包)
    - 494. 目标和 (题目给出target -> 选取一部分整数构成 (sum + targe) / 2 -> 01背包)
  + 目标值不明确，容量不知道，不能用背包，只能枚举子集
    - 1755. 最接近目标值(target)的子序列和(sum) [题目求abs(sum - target)=min最小值，min是多少不知道，目标值不明确，二分拆数组降规模，枚举子数组的子集的和，双指针]
    - 2035. 将数组分成两个数组并最小化数组和的差 [同1755 ]
    - 805. 数组的均值分割[同1755]
* 分割数组类型的题目
  + [416. 分割等和子集] 是否能把数组分为两个等和子集 **回溯 或 01背包的动态规划**
  + [473. 火柴拼正方形]是否能把数组分为4个等和子集 **回溯**
  + [698. 划分为k个相等的子集] **回溯**
  + [410. 分割数组的最大值] 把数组分割成m个连续子集(数组顺序不变)，使得所有和的最大值最小 **动态规划**
  + [689. 三个无重叠子数组的最大和] **动态规划 (思路和410类似)**
  + [813. 最大平均值和的分组] **动态规划(思路和410类似) + 记忆化递归**
  + [1335. 工作计划的最低难度] 动态规划
  + [1723. 完成所有工作的最短时间] 把数组分割成m个子集(顺序可以变)，使得所有和的最大值最小 **回溯(分组的资源分配问题怎么剪枝？)** 或 **状态压缩** 或 **模拟退火**
  + [1655. 分配重复整数] **回溯** 或 **状态压缩(状压的思路和1723一样)**
    - **资源分配问题 (一个位置，可以分配多个资源)**
    - **用状态压缩解题的思路 (1723) 状态为state的资源，分配给前i个位置(人)，枚举第i个人可能得到的所有资源的情况(枚举子集)**
    - **for (int i = 0; i < n; ++i)**
      * **for (int state = 0; state < totalState; ++state)**
        + **for (subMask = state; subMask; subMask = (subMask - 1) & state)**

**int left = state - subMask**

**dp[i][state] <- dp[i-1][left] + val(subMask)**

* + [6007. 数组的最大与和]
  + [1879. 两个数组最小的异或值之和] 同6007
    - 资源分配问题**(一个位置，最多分配一个资源)**
    - **n个位置，m个资源，状态数1<<n，state中c个1，dp[state]表示前c个资源分配到状态为state的位置中**
    - **思路1**
    - **for (int i = 0; i < totalState; ++i)**
      * **int c = bitset<32>(i).count();**
      * **for(int j = 0; j < n; ++j)**
        + **if ((i & (1 << j)) == 0) // 枚举i中的空位置**

**Int newState = i | (1 <<j);**

**f[newState] <- f[i] + val(assign c+1 to j); // i中放前c个资源，第c + 1个资源放到空位置中**

* + - **思路2 (更好理解)**
    - **for (int i = 0; i < totalState; ++i)**
      * **int c = bitset<32>(i).count();**
      * **for(int j = 0; j < n; ++j)**
        + **if (i & (1 << j)) // 枚举i中已经放置的位置**

**Int left = i - (1 << j) // 挖空位置j**

**f[i] <- f[left] + val(assign c to j); // left中放前c-1个资源，第c个资源放到被挖空的位置中**

* + [1681. 最小不兼容性]
    - 资源分配问题 一个数组分割为多个组，每个组满足某个约束，单个组可以看成是当前状态的一个子集，dp[state] <- dp[state - group] + val(group)
  + [1986. 完成任务的最少工作时间段] **状态压缩 (一个状态的最值，可以用两个不相交的子集得来。用枚举子集，计算两个不相交的集合的值)**
  + [1621. 大小为 K 的不重叠线段的数目] **动态规划**
  + [472. 连接词] **动态规划**
  + 1278. 分割回文串 III **分割字符串常见的动态规划思路**
  + 131. 分割回文串 回溯法 求一个字符串的所有分割方案
* 路径问题
* 62.不同路径（中等）：路径问题第一讲
* 63.不同路径 II（中等）：路径问题第二讲
* 64.最小路径和（中等）：路径问题第三讲
* 120.三角形最小路径和（中等）：路径问题第四讲
* 931.下降路径最小和（中等）：路径问题第五讲
* 1289.下降路径最小和 II（困难）：路径问题第六讲
* 1575.统计所有可行路径（困难）：路径问题第七讲（记忆化搜索）
* 1575.统计所有可行路径（困难）：路径问题第八讲（动态规划）
* 576.出界的路径数（中等）：路径问题第九讲
* 1301.最大得分的路径数目（困难）：路径问题第十讲
* 区间内的线段问题
  + 覆盖区间的最少线段数
    - 1024. 视频拼接 (按终点排序+贪心)
    - 1326. 灌溉花园的最少水龙头数目 (和1024一样)
  + 不重叠区间
    - 1235. 规划兼职工作
    - 1621. 大小为 K 的不重叠线段的数目
    - 689. 三个无重叠子数组的最大和
    - 435. 无重叠区间 (最大不重叠区间 按终点排序+贪心 或动态规划 前i区间内的最大值)
    - 1477. 找两个和为目标值且不重叠的子数组 (滑动窗口求所有和为targe的子数组)
* 最短路径问题
  + 1976. 到达目的地的方案数 (从源点，走最短路径，到达节点i的方案数)
  + 2092. 找出知晓秘密的所有专家
* 常见求最大值 最小值问题，dp，**二分，**bfs/dfs等
* 博弈问题
  + 877. 石子游戏 I - V
  + 913. 猫和老鼠
* 单调栈问题适用场景：**在一维数组中找第一个满足某种条件的数**(在一维数组中对每一个数找到第一个比自己小(大)的元素)
  + 84. 柱状图中最大的矩形
  + 85. 最大矩形
* 字符串哈希(O(1)时间判断子字符串是否相等，**理解字符串转哈希值的数学表达式**)
  + 1316. 不同的循环子字符串 (https://www.bilibili.com/video/av83044577?p=2) (在长度不同的子串中找重复子串)
  + 187. 重复的DNA序列(在长度相同的子串中找重复子串)
  + 1044. 最长重复子串 (在长度相同的子串中找重复子串)
* 计算一个区间[left, right]内，包含某个字符pos的子串的个数 = (pos - left + 1) \* (right - pos + 1)
  + [828. 统计子串中的唯一字符]
* 匈牙利算法 求 二分图最大匹配
  + 基本介绍 https://www.cnblogs.com/cruelty\_angel/p/10808729.html
  + LCP 04. 覆盖
* 差分技巧
  + 给定数组a[0] a[1] a[2] a[3] a[4] a[5]
  + 差分数组d[0] = a[0] d[1] = a[1] - a[0] d[2] = a[2] - a[1] d[3] = a[3] - a[2] d[4] = a[4] - a[3] d[5] = a[5] - a[4]
  + 差分数组前缀和 a[0] a[1] a[2] a[3] a[4] a[5]
  + 性质区间[L, R] + 1 等价于d[L] + 1 和 d[R + 1] - 1
  + 1526. 形成目标数组的子数组最少增加次数 (https://www.bilibili.com/video/BV1ri4y137Js?p=5)
* 线段树 O(n)建树 O(log n)对数组进行更新和查询
  + 6030. 由单个字符重复的最长子字符串