寒假回顾

廖嘉琦

北京理工大学 ACM 俱乐部

2022 年 7 月 11 日

寒假集训都讲了些啥??

- C 语言复习与拓展
- STL 应用
- 排序，枚举，模拟
- 二分，三分，快速幂
- 贪心
- 图与搜索 (BFS，DFS)
- 拓扑排序，最短路
- 双指针，前缀和
- 矩阵快速幂，简单数论
- 动态规划
- 并查集，最小生成树
- 线段树，树状数组

C 语言

- ~~C 语言程序设计考的如何?~~
- C++ 学的咋样?
- C with STL!
- 相信你们对语法应该很熟练!

STL 容器

Sequence	Associative	Unordered associative	adaptors
array	set	unordered_set	stack
vector	map	unordered_map	queue
deque	multiset	unordered_multiset	priority_queue
forward_list	multimap	unordered_multimap	
list			

更多细节，详见Containers library - cppreference

算法库

- Non-modifying sequence operations
  - std::count, std::count_if
- Modifying sequence operations
  - std::reverse, std::swap
- Partitioning operations
  - std::partition, std::stable_partition
- Sorting operations
  - std::sort, std::stable_sort, std::nth_element
- Binary search operations (on sorted ranges)
  - std::lower_bound, std::upper_bound

算法库-续 1

- Other operations on sorted ranges
  - std::merge, std::inplace_merge
- Set operations (on sorted ranges)
  - std::set_intersection, std::set_union
- Heap operations
  - std::make_heap, std::sort_heap
- Minimum/maximum operations
  - std::max, std::min, std::max_element
- Comparison operations
  - std::equal, std::lexicographical_compare

算法库-续 2

- Permutation operations
  - std::is_permutation, std::next_permutation
- Numeric operations
  - std::partial_sum, std::adjacent_difference
- Operations on uninitialized memory
  - std::partial_sum, std::adjacent_difference

算法库-续 2

- Permutation operations
  - std::is_permutation, std::next_permutation
- Numeric operations
  - std::partial_sum, std::adjacent_difference
- Operations on uninitialized memory
  - std::partial_sum, std::adjacent_difference

更多细节，详见Algorithms library - cppreference，或许会收获小惊喜哦

排序

- std::sort / cmp
- 比较型排序 VS. 非比较型排序
- 与贪心结合
- 高维时排序一维从而降维

枚举

- 确定解空间
- 减少枚举的空间（剪枝）
- 选择合适的顺序枚举
- DFS

模拟

- 码量大，代码长
- 建议先做好写的题，彻底想清楚所有细节再写
- 何必折磨自己呢?

二分与三分

- 二分寻找：std::lower_bound, std::upper_bound
- 三分寻找：查找单峰函数的最值
- 倍增：每次翻倍，二进制拆分组合
- 二分答案：答案具有某种单调性

快速幂

基本思想是通过倍增减少重复计算，将指数用二进制分解，例如

$$a^{11} = a^8 a^2 a^1$$

而计算 a^{2^i} 时可以令 $i = 0$ 从小往大计算，每次平方自己。

快速幂

基本思想是通过倍增减少重复计算，将指数用二进制分解，例如

$$a^{11} = a^8 a^2 a^1$$

而计算 a^{2^i} 时可以令 $i = 0$ 从小往大计算，每次平方自己。

应用：

- $O(\log p)$ 求 a^p：求逆元 $a^{p-2} \bmod p$
- $O(k^3 \log n)$ 求 $A^n_{k\times k}$：求解线性递推方程 $h_i = \sum_{j=1}^{k} a_j \times h_{i-j}$

贪心

适用范围

贪心算法在有最优子结构的问题中尤为有效。最优子结构的意思是问题能够分解成子问题来解决，子问题的最优解能递推到最终问题的最优解。

贪心

适用范围

贪心算法在有最优子结构的问题中尤为有效。最优子结构的意思是问题能够分解成子问题来解决，子问题的最优解能递推到最终问题的最优解。

常见证法：

- 反证法
- 归纳法

贪心

适用范围

贪心算法在有最优子结构的问题中尤为有效。最优子结构的意思是问题能够分解成子问题来解决，子问题的最优解能递推到最终问题的最优解。

常见证法：

- 反证法
- 归纳法

常见套路：

- 按照某规则排序后选择
- 每次取出最值用于更新

图的存储 - 稠密图

使用邻接矩阵存图时，**无法存储重边**，具体性能如下：

- 空间复杂度：$O(n^2)$
- 遍历点 u 的所有出边：$O(n)$
- 遍历整张图：$O(n^2)$
- 判断 (u, v) 间是否存在边：$O(1)$

若 $m \ll n$，则二维数组中存在大量空位，浪费了很多空间。

图的存储 - 一般图

其一是**邻接表**，基于 C++ 提供的标准库 `std::vector<int>`，将点 i 所有出边的端点插入下标为 i 的 `vector` 中。

其二是**链式前向心**，本质上是用**链表**手动实现的**邻接表**。

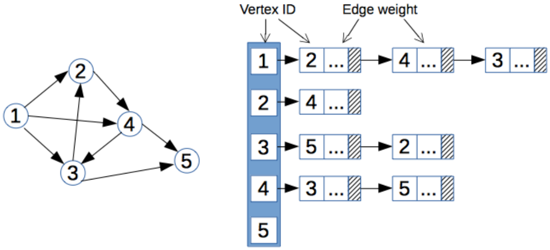

图: 链式前向心

图的存储 - 一般图

邻接表[1]的具体性能如下：

- 空间复杂度： $O(m)$
- 遍历点 u 的所有出边： $O(d^+(u))$
- 遍历整张图： $O(m)$
- 判断 (u, v) 间是否存在边： $O(d^+(u))$

[1] **链式前向心**在跑网络流时，反向边易于访问，相比于**邻接表**更加方便。

图的遍历

- 深度优先搜索（DFS）：每次都尝试向更深的节点走
  - 递归
  - 树上应用
- 广度优先搜索（BFS）：每次都尝试访问同一层的节点
  - 队列
  - 无权最短路

拓扑排序

有向边 $i \to j$ 意味着节点 j 依赖于节点 i。

拓扑排序的目标是将 DAG 上的所有节点排序，使得排在前面的节点不能依赖于排在后面的节点。

BFS，每次将入度为 0 的点入队，复杂度 $O(|V| + |E|)$。

拓扑排序

有向边 $i \to j$ 意味着节点 j 依赖于节点 i。

拓扑排序的目标是将 DAG 上的所有节点排序，使得排在前面的节点不能依赖于排在后面的节点。

BFS，每次将入度为 0 的点入队，复杂度 $O(|V|+|E|)$。

字典序最值？将队列替换成堆，复杂度 $O(|E|+|V|\log|V|)$。

最短路

- 全源最短路
  - Floyd： $O(|V|^3)$
    - 传递闭包： $O(\frac{|V|^3}{w})$
  - Johnson： $O(|V||E|\log|V| + |V||E|)$
- 单源最短路
  - 非负权图： Dijkstra： $O((|V| + |E|)\log|V|)$
  - 负权图： Bellman-Ford(SPFA)： $O(|V||E|)$

双指针算法

- 对于所有合法区间 $[l, r]$，随着 r 递增，l 单调不减
- 区间限制，记作 $g(l, r)$
- 区间状态，记作 $f(l, r)$
- $f \Rightarrow g$
- 求合法区间长度的最值

观察转移

1. $[l, r] \Rightarrow [l, r+1]$：插入 a_{r+1}，复杂度记为 $O(\alpha)$
2. $[l, r] \Rightarrow [l+1, r]$：删除 a_l，复杂度记为 $O(\beta)$

合并时易删除难，通常 $O(\beta) > O(\alpha)$

举个栗子

- $f = \sum / \prod /\mathsf{xor}$， $O(\alpha) = O(\beta) = O(1)$，存在逆运算，插入可撤销
- $f = \min / \gcd /\mathsf{and}$， $O(\alpha) = O(1)$， $O(\beta) = O(n)$，合并时丢失信息，插入无法撤销
- $f =$ 背包， $O(\alpha) = O(V), O(\beta) = ?$

举个栗子

- $f = \sum / \prod /\text{xor}$，$O(\alpha) = O(\beta) = O(1)$，存在逆运算，插入可撤销
- $f = \min / \gcd /\text{and}$，$O(\alpha) = O(1)$，$O(\beta) = O(n)$，合并时丢失信息，插入无法撤销
- $f =$ 背包，$O(\alpha) = O(V), O(\beta) = ?$

用空间换时间的方法可以解决删除问题，但此处地方不够，略过不讲。

前缀和

- 不支持修改
- 提前计算好前 i 项，通过逆运算 $O(1)$ 查询区间和
- 高维前缀和基于容斥原理
- 树上前缀和维护路径和
- 与差分互为逆运算

矩阵快速幂优化线性递推方程

对于一个 k 次的线性递推：

$$h_n = a_1 h_{n-1} + a_2 h_{n-2} + \ldots + a_k h_{n-k}, \forall\ n \in \mathbb{N}\ ,\ n > k$$

矩阵快速幂优化线性递推方程

对于一个 k 次的线性递推：

$$h_n = a_1 h_{n-1} + a_2 h_{n-2} + \ldots + a_k h_{n-k}, \forall\ n \in \mathbb{N}\ ,\ \ n > k$$

构造转移矩阵 A 和初始向量 x

$$A = \begin{bmatrix} a_1 & a_2 & \cdots & a_{k-2} & a_{k-1} & a_k \\ 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 1 & 0 \end{bmatrix}_{k \times k} \qquad x = \begin{bmatrix} h_{k-1} \\ h_{k-2} \\ \vdots \\ h_2 \\ h_1 \\ h_0 \end{bmatrix}_{k \times 1}$$

矩阵快速幂优化线性递推方程[2]

则

$$Ax = \begin{bmatrix} a_1 & a_2 & \cdots & a_{k-2} & a_{k-1} & a_k \\ 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} h_{k-1} \\ h_{k-2} \\ \vdots \\ h_2 \\ h_1 \\ h_0 \end{bmatrix} = \begin{bmatrix} h_k \\ h_{k-1} \\ \vdots \\ h_3 \\ h_2 \\ h_1 \end{bmatrix}$$

[2]特征多项式 +FFT 可以优化到 $O(k \log k \log n)$

矩阵快速幂优化线性递推方程[2]

则

$$Ax = \begin{bmatrix} a_1 & a_2 & \cdots & a_{k-2} & a_{k-1} & a_k \\ 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} h_{k-1} \\ h_{k-2} \\ \vdots \\ h_2 \\ h_1 \\ h_0 \end{bmatrix} = \begin{bmatrix} h_k \\ h_{k-1} \\ \vdots \\ h_3 \\ h_2 \\ h_1 \end{bmatrix}$$

因此 $A^{n-k+1}x = [h_n, h_{n-1}, \cdots, h_{n-k+1}]^\top$。

[2]特征多项式 +FFT 可以优化到 $O(k \log k \log n)$

欧几里得算法

$$gcd(m, n) = gcd(n, m \bmod n)$$

欧几里得算法就是不断地递归应用这个等式，直到 $n = 0$ 时结束。

复杂度 $O(\log m)$。

裴蜀定理

若 a 和 b 是整数，方程 $ax + by = d$ 有整数解当且仅当 $\gcd(a, b) | d$。

裴蜀定理仅能知道解是否存在，具体求解的方法为扩展欧几里得算法。

拓展欧几里得算法

求方程 $ax + by = \gcd(a, b)$ 的整数解（根据裴蜀定理判定有解）。

显然 $bx' + (a \bmod b)y' = \gcd(a, b)$ 同样存在整数解。

同时，$a \bmod b = a - b\lfloor \frac{a}{b} \rfloor$，再联立方程可以得到

$$ax + by = bx' + (a - b\lfloor \frac{a}{b} \rfloor)y'$$

整理后，

$$ax + by = ay' + b(x' - \lfloor \frac{a}{b} \rfloor)y'$$

扩展欧几里得算法

如果已经知道了 (x', y')，那么在满足下式的情况下方程成立。

$$\begin{cases} x = y' \\ y = x' - \lfloor \frac{a}{b} \rfloor \end{cases}$$

这样就可以利用方程 $bx' + (a \bmod b)y' = \gcd(a, b)$ 的整数解 (x', y') 来计算出方程 $ax + by = \gcd(a, b)$ 的整数解了。

最后只要按照前面计算最大用因数的方法，在递归的过程中加入 x 和 y 的计算就可以了。当递归到末尾的时候 b 变为 0，这是方程有整数解 $x = 1, y = 0$。

同余

同余的概念提供了一种描述整除性质的简便方法。

如果 m 整除 $a-b$，我们就说 a 与 b 模 m 同余并记为

$$a \equiv b \pmod{m}$$

同余

如果

$$a_1 \equiv b_1 \pmod{m}$$

$$a_2 \equiv b_2 \pmod{m}$$

那么

$$a_1 \pm a_2 \equiv b_1 \pm b_2 \pmod{m}$$

$$a_1 a_2 \equiv b_1 b_2 \pmod{m}$$

乘法逆元

在模 p 意义下，x 的乘法逆元 x^{-1} 满足如下同余方程：

$$xx^{-1} \equiv 1 \pmod p$$

在同余意义下，逆元就是除法，但逆元可能不存在。

素数与素数筛

素数

大于 1 的正整数 a，除了可以被 1 和 a 整除，没有其他的约数

如何判定?

- 暴力判定：$O(\sqrt{x})$
- Miller-Rabin：$O(k\log x)$
- Eratosthenes 筛法：$O(n\log\log n)$
- 欧拉筛法：$O(n)$

Eratosthenes 筛法

基本思路：删去所有素数的倍数，未被删除的就是素数了。

从 2 开始自增，先删去 2 的所有倍数。下一个未被删去的数一定是素数，并删去它的所有倍数，然后不断进行这个操作。

被删的数总共有 $n(\frac{1}{2}+\frac{1}{3}+\cdots)=O(n\log\log n)$。

本筛法常用于预处理 $1\sim n$ 中每个数的所有因数。

欧拉筛法

对于任意一个合数，通过埃氏筛法，它的所有质因数都会将它删除一次，即它被删除了多次，效率不高。

欧拉筛的核心思想是让每个合数都只被它的最小质因子删除。

代码实现时只比埃氏筛法多一行，实际表现上看提升并不大。

筛法的应用

- 求 x 的所有约数和
- 求 x 的约数个数
- 求某个积性函数 $f(x)$
- $\cdots$

GCD 与 LCM

- 最大公约数：$\gcd(a,b)=\gcd(b, a \bmod b)$
- 最小公倍数：$lcm(a,b)=\frac{a\times b}{\gcd(a,b)}$
- 欧拉函数：$\varphi(n)$ 表示小于等于 n 且和 n 互质的数的个数
- 费马小定理：对于任意整数 a，$a^{p-1}\equiv 1 \pmod p$
- 欧拉定理：若 $\gcd(a,m)=1$，则 $a^{\varphi(m)}\equiv 1 \pmod m$

组合数学

- 排列：$A_m^n = \frac{n!}{(n-m)!}$ （有序）
- 组合：$C_m^n = \frac{n!}{(n-m)!m!}$ （无序）
- 加法原理：分类完成
- 乘法原理：分步完成
- 容斥原理：$|\bigcup_{i=1}^{n} S_i| = \sum_{m=1}^{n} (-1)^{m-1} \sum_{a_i < a_{i+1}} |\bigcap_{i=1}^{m} S_{a_i}|$
- 抽屉原理：n 个物体划分为 k 组，至少一组含有 $\geq \left\lceil \frac{n}{k} \right\rceil$ 个物品
- 卡特兰数：$H_n = \frac{H_{n-1}(4n-2)}{n+1}$

组合数

递推式：

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$$

对称性：

$$\binom{m}{n} = \binom{n-m}{m}$$

卢卡斯定理（p 为质数）：

$$\binom{n}{m} \bmod p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$$

动态规划

动态规划解决的最优化问题（否则是递推）需要满足两点性质：

- 重叠子问题
- 最优子结构

动态规划

动态规划解决的最优化问题（否则是递推）需要满足两点性质：

- 重叠子问题
- 最优子结构

动态规划有两种实现方式：

- 制表法：自底向上
- 记忆化搜索：自顶向下

重叠子问题

大前提：原问题可被拆分成求解形式相同的子问题

- 状态数有限 $\Rightarrow$ 可计算、存在边界
- 求解存在依赖关系 $\Rightarrow$ DAG 结构
- 边界答案已知 $\Rightarrow$ 与状态直接相关

最优子结构

大前提：原问题的最优解由相关子问题的最优解组合而成

- 子问题的其他解对原问题无贡献 $\Rightarrow$ 无需计算
- 贪心也是可能的解法

常见 DP

- 线性 DP（序列 DP、区间 DP）
- 图上 DP（树形 DP、DAG 上 DP）
- 状压 DP
- 期望 DP
- 插头 DP
- 动态 DP
- $\cdots$

常见的 DP 的优化方法

- 基于单调性的优化 $\Rightarrow$ 优化转移方程的计算
  1. 斜率优化
  2. 单调队列/单调栈优化
  3. 四边形不等式优化
- 基于状态设计的优化 $\Rightarrow$ 改变转移方程的形式
- 基于数据结构的优化
  1. 线段树
  2. 平衡树（set、map 等）

DP 的一般解题思路

1. 数学符号化待求问题
2. 设计**状态表示**和**最优化值**
   - 状态表示：对当前子问题的解的局面集合的一种充分的描述
   - 最优化值：对应的状态集合下的最优化信息，最终得到答案
3. 推出状态转移方程、分析复杂度
4. 考虑是否需要优化（或者回到第 2 步修改状态表示）
5. 写代码、跑样例、调试、提交、AC!

背包 DP

给定一些物品，每个物品有价值、花费、个数等参数，要求在满足花费限制下求得最大价值，物品间可能存在一些额外限制。

背包 DP

给定一些物品，每个物品有价值、花费、个数等参数，要求在满足花费限制下求得最大价值，物品间可能存在一些额外限制。

- 01 背包
- 完全背包
- 多重背包
- 混合背包
- 分组背包
- $\cdots$

如何优化 DP——以多重背包为例

问题

有 N 种物品和一个容量为 V 的背包。第 i 种物品的体积是 C_i，价值是 W_i，有 M_i 个。

求解如何使背包中物品总价值最大。

多重背包——朴素做法

- 把「每种物品选 k 次」等价转换为「有 k 个相同的物品，每个物品选一次」，转换成 01 背包模型
- 设计状态 $f[i,j]$ 为前 i 种物品，占用 j 的空间的最大价值
- 转移方程：$f[i,j]=\max\limits_{k=0}^{M_i} f[i-1,j-k\times C_i]+k\times W_i$
- 复杂度 $O(V\sum M_i)$

多重背包·优化 1——二进制分组

- 原状态中哪里存在冗余?
- 一次性选择 x 件和选择 x 次 1 件是等价的
- 将 M_i 进行二进制分组，将它分成 $\lceil \log_2 M_i \rceil$ 件物品
- 例如 $30 = 2^1 + 2^2 + 2^3 + 2^4$，将 $M_i = 30$ 拆成 4 件物品，它们的体积和价值为对应倍数的单个物品
- 复杂度降为 $O(V \sum \log_2 M_i)$

多重背包·优化 2——单调队列

- 观察转移方程：$f[i,j]=\max_{k=0}^{M_i} f[i-1,j-k\times C_i]+W_i\times k$
- $f[i,j],f[i,j+C_i],f[i+j+2\times C_i],\cdots$ 属于同一组
- 它们与 C_i 前的系数 k 有关，且 k 是连续的
- 因此我们要维护的是 k 在某段可行区间内的 $\max\{f[i,j+k\times C_i]\}$
- 因此可以用单调队列维护 $\max\{f[i,j+k\times C_i]\}$
- 复杂度 $O(VN)$

并查集

并查集是一种树形的数据结构，它用于处理一些不交集的问题。

它支持两种操作：

- 查找：确定某个元素处于哪个子集 $\Rightarrow$ 路径压缩
- 合并：将两个子集合并成一个集合 $\Rightarrow$ 启发式合并

最小生成树

最小生成树是在连通加权无向图中一棵权值最小的生成树。

- Kruskal：$O(|E| \log |E|)$，基于并查集
  - Kruskal 重构树
- Prim：$O((|V| + |E|) \log |V|)$，基于最短路
- 次小生成树
- 最小树形图

线段树与树状数组

掏出老古董课件！

详见 线段树与树状数组 by Skqliao