杂谈

FizzyDavid

杂谈内容

- 一些经验与易错点~~血的教训~~
- 一些trick与知识点
  - 常数长度多项式的任意次幂的线性求法与p-recursive
  - 通用的欧几里得
  - 动态图上数据结构题的一类离线做法

- __builtin_ctz和__builtin_ctzll
  count trailing zeros
  注意0的值**未定义**

- __builtin_clz和__builtin_clzll
  count leading zeros
  注意0的值**未定义**

- __builtin_popcount和__builtin_popcountll
  count ones

- Left/Right Shift Operator (<</>>)
  参数范围必须在[0, 31(63)]，否则**未定义**
  有符号数右移会**补符号位**（~~JSOI惨失队长的傻逼选手的忠告~~）
  手写bitset时请不要偷懒，还是乖乖写unsigned吧
  e.g. (-1)>>1 = -1

- 三项相加会爆int 相减也会
  e.g. x-y+z ❌ (x-y)%mod+z ✅
- cipolla需要特判0
- new的东西不会调用构造函数，需要手动清空
  topcoder的struct/class也是（

- 多点求值的求值集合如果为空，可能会死循环
- `unordered_map`频繁清空可能会慢
- 等比序列求和需要特判公比为1
  （模意义下即使原来不是1也有可能变为1）

- 给定c次多项式P(x)，求P(x)^d。

- Q = P^d
Q’ = d*P^(d-1) P’
Q’P = d*P^d P’
Q’P = d*P’Q

- 比较两边x^n系数，可得递推式

- e.g. sqrt(a+bx)
a(n+1)Q[n+1]+bnQ[n] = b/2Q[n]
Q[n+1] = ab(1/2-n)/(1+n)Q[n]

- 类似的推导方法:
- `e.g. e^P(x) ln(P(x))`
- `e.g. P(x)e^P(x)`
- `e.g. cos(P(x)) sin(P(x))` （仅供娱乐）

- P-recursive sequence：
  第n项的递推式的系数是关于n的多项式
- https://en.wikipedia.org/wiki/Holonomic_function
- 列未知数，高斯消元

- 给定矩阵N行N列矩阵A、B，求：$\sum_{x=1}^{L} A^x B^{\left\lfloor \frac{Px+R}{Q} \right\rfloor}$
- N<=20, L, P, Q, R, PL/Q<= 10^18
- O(N^3 log 10^18)
- https://loj.ac/submission/315887

- 一条斜线段（从左下到右上）对应一个操作序列，序列中有两种操作U和R（Up和Right）。以单位长度为间隔画出所有横线和竖线，（即画出所有x=n的竖线、y=n的横线，n是任意整数）。当斜线碰到横线时进行操作U，碰到竖线时进行操作R，若同时碰到（即碰到整点时）规定先进行U后进行R，即进行操作UR。

- 操作序列需要满足**信息可合并**。只需要两个操作序列的信息，即可计算出将这两个操作序列拼接后的信息。（e.g. 线段树节点信息）

- 操作可以被看成代码段：
  初始化： `int x = 0, sum = 0;`
  U： `x++;`
  R： `sum += x;`

  可以实现斜线下整点计数。

- O(1)快速维护：
  最终x，sum累加次数（R操作的次数），最终sum

- 操作可以被看成代码段：
  初始化： `matrix curA = I, curB = I, sum = 0;`
  U： `curB = curB * B;`
  R： `curA = curA * A, sum += curA * curB;`
  可以实现本题：万能欧几里得。

- O(1)次矩阵乘法快速维护：见代码

- 还可以实现：loj类欧几里得模板题
  https://loj.ac/problem/138

$$\sum_{x=0}^{n} x^{k_1} \left\lfloor \frac{ax+b}{c} \right\rfloor^{k_2}$$

- solve(P, Q, R, L, A, B) A、B为操作序列段
  生成序列共有L个B，将其编号为1~L。
  第i-1个B 与 第i个B 之间共有
  floor((Pi+R)/Q)-floor((P(i-1)+R)/Q)个A。

- e.g. solve(2, 3, 1, 3, A, B) -> ABBAB
  solve(5, 3, 1, 3, A, B) -> AABABAAB

- R = R mod Q ： 无影响
  P = P mod Q ： B = A^floor(P/Q) B

- solve(P, Q, R, L, A, B) 0 <= P,R < Q
  第i个B之前共有floor((Pi+R)/Q)个A

- 对于第x个A和第y个B，满足x <= floor((Py+R)/Q)
  x <= (Py+R)/Q
  Qx-R <= Py
  (Qx-R)/P <= y
  ceil((Qx-R)/P) <= y
  floor((Qx-R+P-1)/P) <= y
  第x个A前共有floor((Qx-R-1)/P)个B

- 当x=0时，floor((Qx-R-1)/P)<0，第一个A要特殊考虑。
  最后一个A后还会有多余的B。

- solve(P, Q, R, L, A, B)
- R’ = R mod Q
  P’ = P mod Q
  B’ = A^floor(P/Q) B
- solve(Q, P’, Q-R’-1, L-1, B’, A)
- 总复杂度O(log max(P, Q))
  \sigma log(P/Q) = \sigma log(P)-log(Q)

- 优点1：通用性好
- 优点2：代码简洁
- 优点3：常数小
- 优点4：不涉及逆操作，可以任意模数
- 优点5：欧几里得模板部分无须改动，只需专注信息部分
- 缺点：对于整点计数之类的简单问题，可能显得略繁琐

- 一个无向图，每次操作加边或删边，求每次操作后图中桥的个数
- 离线 O(n log n)

- 按时间分治，考虑某段时间区间[l,r]内图的变化
- 只考虑在[l,r]中可能出现的边，先考虑[l,r]中都出现的边，这些边形成的环可以缩成一个点，最终剩下一棵树。
- 考虑[l,r]中操作更改的边，共r-l+1条。将图缩为这些边的端点形成的虚树。
- 分治下去时，保证了图的规模与时间区间的长度同级。进行缩环、求虚树时，只要做到图的规模的线性，总复杂度就是O(n log n)。

- e.g. 一棵树，每次翻转一条路径上边的存在性，求独立集个数
- e.g. 无向图，一个加边删边操作序列，每次询问x与y是否在操作序列区间[l, r]中一直保持联通

- Thank you!
- Questions are welcomed.