kririae 2018 - 7 - 7日训练总结

> 那么就对第一天的训练进行一个简单的总结吧

多重背包$logn$优化

说到多重背包的$logn$优化，其实需要配合鬼谷子的钱袋食用 我们考虑任意一个数n，我们可以证明，可以通过把$\sum 2^i \le n$中的每一个2^i和$n - \sum 2^i$构成任意一个小于等于n的数字。不写证明，简单来说，我们先这么看，我们求出第一个$\sum 2^i > n$，这样最后一位2^i一定是大于之前的$n - \sum 2^i$。既然$\sum 2^i > n$都可以全部构成，那么少了一点也不差多少是吧.jpg

最长上升子序列和最长公共子序列的$O(nlogn)$算法

附上核心代码

```
scanf("%d", &n);
for (R int i = 1; i <= n; ++i) scanf("%d", &a[i]);
for (R int i = 1; i <= n; ++i) scanf("%d", &b[i]);
for (R int i = 1; i <= n; ++i) p[a[i]].push_back(i);
for (R int i = 1; i < maxn; ++i) reverse(p[i].begin(), p[i].end());
for (R int i = 1; i <= n; ++i)
  for (R int j = 0; j < p[b[i]].size(); ++j) rep.push_back(p[b[i]][j]);
memset(g, 0x3f, sizeof(g));
int ans = 0;
for (R int i = 0; i < rep.size(); ++i)
{
  int pos = lower_bound(g + 1, g + 1 + n, rep[i]) - g;
  g[pos] = rep[i], ans = max(ans, pos);
}
cout << ans << endl;
```

大意和其他的优化其实是类似的，都是利用重复性。

首先说简单以前的最长上升子序列吧，我们发现，长度为i的子序列会有很多个，在这很多个子序列中，我们考虑怎么样的子序列可以不用。

我们考虑子序列的结尾a_i。对于相同长度的子序列，假如说可以转移，我们转移后的长度是$len+1$，我们引入“潜力”，可以很显而易见知道的是，a_i对于后面转移的可能性会高得多，所以我们保留最小的一个a_i。稍微具体一点，我们定义g数组，$g[i]$表示长度为i的最长上升子序列的最后一个字符是$g[i]$。我们转移的时候在g数组中$lower_bound$到可以转移的位置，然后看这个位置是不是最后一个，如果是的话，我们使长度为pos的位置$= rep[i]$。这里有一个实现的小操作，我们如果所有的都小于$rep[i]$的话，那么二分到的是最后一个位置。

接下来我们来看相对比较复杂的最长公共子序列…

首先说实现吧，我们对于A, B序列求最长公共子序列，那么，对于A中的任意一个A_i，我们找出A_i在所有中出现的位置，假如说是$ababa$，那么a的出现的位置就是$1, 3, 5$，我们把这串数字倒过来，$5, 3, 1$，然后放进B中，假如说B是$babab$，我们把B替换成$[4, 2], [5, 3, 1], [4, 2], [5, 3, 1], [4, 2]$，然后对这个序列求最长上升子序列。至于为什么，我理解的还不是很透彻，不过可以大概概括一下，假如说是aaa, aaa。B串就会被替换成$3, 2, 1, 3, 2, 1, 3, 2, 1$，这样，倒序就保证了每一个字符自会被选一次。而不是三次。而对于A中的任意$A_j, A_i\ (j > i)$，都能够保证在替换后的序列中是连续上升的…好复杂啊。

快速幂 && 龟速乘

快速幂用来加速，龟速乘用来防止爆ll。

先说快速幂吧，对于a^b，可以拆成$a^{2^i} \cdot a^{2^j} \ldots$我们分解$b$为二进制，然后进行操作。

然后说龟速乘，防止爆ll。$a \cdot b$，我们可以拆成$(2^i + 2^j + \ldots) \cdot b$，然后拆开，然后加起来~

然后说矩阵快速幂，我们的矩阵是介个样子的。

先说比较简单的斐波那契吧，递推公式如下$f[n] = f[n-1] + f[n-2]$。我们可以这么构造矩阵:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} f[n-1] \\ f[n-2] \end{bmatrix} = \begin{bmatrix} f[n-1] + f[n-2] \\ f[n-1] \end{bmatrix} = \begin{bmatrix} f[n] \\ f[n-1] \end{bmatrix}$$

这样我们就从

$$\begin{bmatrix} f[n-1] \\ f[n-2] \end{bmatrix} \to \begin{bmatrix} f[n] \\ f[n-1] \end{bmatrix}$$

这就是一个递推步骤，然后快速幂同理，只是要注意乘法方向。

看一个类似的题?

[NOI2012] 随机数生成器

递推公式如下:

$f[n+1] = (af[n] + c) \bmod m$。

简单推敲一下，可以发现，矩阵这么构造

$$\begin{bmatrix} a & 1 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} f[n] \\ c \end{bmatrix} = \begin{bmatrix} a \cdot f[n] + c \\ c \end{bmatrix} = \begin{bmatrix} f[n+1] \\ c \end{bmatrix}$$

以上~注意这道题需要龟速乘

后面的各种dp练习，具体我就不描述了

LUOGU1108 低价购买

简单来说，给定一个序列A，求出一个最长下降自序列，然后统计所有不同的最长下降自序列的个数。 难度在于什么时候需要初始化统计和怎么统计不同的。

我们考虑“两个完全相同的序列”满足什么条件，那么怎么样让它不满足了，我们发现a_{i-1}和上一个序列的b_{i-1}不同时，就满足这俩序列一定不同，我们筛选一下相同的，然后不从相同的转移，就ok啦

然而，初始化什么地方，我们这么考虑，长度为1的序列不需要由别人构成，所以我们$g[i] = 1$，长度不为1的个数一定是由小于其长度的序列构成，所以我们都赋值为0.

ZJOI2007 时态同步

一道相对比较简单的树形dp，我们考虑一棵深度为2的树，我们的完成时间是其子节点中完成时间最晚的，我们就每次把其子节点中小于最晚的点补充到最晚去，成本就是其差值绝对值之和，对于这棵树进行递归处理

今天就到这里吧，其实还有一个LUOGU1537，有点水就不解释了。