暑期学习总结及学习规划

MicDZ

September 15, 2019

目录

1 暑期之前所学的知识

在来到长郡之前，我曾参加过由广益组织的暑期信息学夏令营，对信息学竞赛有了一些基本的了解，对于基础的算法如二分、深度优先搜索、广度优先搜索、图的最短路径算法、图的最小生成树算法、简单的动态规划等有了一些基础的了解。并在洛谷上进行非系统的刷题。

我曾参加过 NOIp2017 年的普及和提高组与 NOIp2018 年的普及组和提高组，两年的竞赛大多依靠于自己的学习，缺少教练系统的指导，这使得我面对思维难度较大的题、灵活转变算法逻辑的题目时难以应对，在数学方面的理解上仍停留在一个普通的中学生的水平，不能适应信息学竞赛的需要。这直接导致了 NOIp2017 年提高组 D1T1、NOIp2018 年提高组 D1T2 的丢分。两年的竞赛均未能取得良好成绩。糟糕的成绩使我不得不参加中考，甚至产生了放弃竞赛的想法。

2 暑期集训所学的知识

暑期短暂的 25 天的集训，通过谭教练的培训，我重新对深度优先搜索、广度优先搜索、二分答案、图的最短路径算法、图的最小生成树算法进行理解，并开始了新的学习。接下来的内容是我对暑期学习的总结，均为我自己对算法的理解。

2.1 搜索的进阶

在集训之前，我的搜索的技能可以应对一些基础的不需要记忆化的题目。例如 NOIp2017 年奶酪、NOIp2017 棋盘等。但是对于更有难度的记忆化搜索以及迭代加深、启发式搜索几乎没有了解。

集训期间虽然仍没有更加深入地了解迭代加深等内容，但是我学习到了对搜索状态的压缩，例如二进制压缩、康托展开压缩的方法，这些是在此之前的训练中从未遇到的。

2.1.1 进制压缩法

我的黑白棋游戏二进制压缩的代码如下：

```
int binary_hash(char str[][]) {
    int ans=0;
    REP(i,1,4) REP(j,1,4)
        ans=(ans<<1)+str[i][j]-'0';
    return ans;
}
```

二进制压缩适用于对于状态的每一个位置有且仅有两种可能值的情况，否则需要多进制的压缩方式。其支持 $\Theta(1)$ 的单位置查询、$\Theta(n)$ 的压缩。

2.1.2 康托展开压缩法

康托展开的公式如下：

$$X = \sum_{i=1}^{n} a_i \prod_{j=1}^{i-1} j \tag{1}$$

$$= a_n(n-1)! + a_{n-1}(n-2)! + ... + a_1 * 0! \tag{2}$$

其中，a_i 表示的是表示原数的第 i 位在当前未出现的元素中是排在第几个。

康拓展开求出的是一个序列在对应长度序列的全排列中由小到大的顺序。求解某一序列的康托序与通过康托序逆推排列都是可以完成的。但是康托展开的朴素复杂度为 $\Theta(n^2)$，用 log 的

数据结构优化可以实现 $\Theta(n \log n)$ 但是实现过于复杂，逆康托展开的复杂度则更高，优化更为复杂。

2.1.3 哈希压缩法

还有另一种高效率的状压方法是哈希，它能够适用于各种类型的序列，例如字符串、矩阵，不受限于状态的位置及可能情况。但是无法做到通过散列值逆推出原序列，只能保证在大概率下每个序列有且仅有一个对应散列值。所以哈希更广泛应用于去重与互联网加密等环节中。哈希仍可能存在冲突即多个序列对应一个散列值的情况，但是发生错误的概率极低。无错哈希对内存的占用较高也不常用。

哈希的代码如下：

```
int hash(char s[]) {
    int len=strlen(s);
    int ans=0;
    REP(i,0,len-1)
        ans=(ans*base+s[i])%mod+prime;
    return ans;
}
```

朴素哈希的复杂度为 $\Theta(len)$，STLmap 映射的复杂度要再乘上一个 log 的级别。

通过压缩状态可以更好地判断该状态是否被搜索过、存储该状态下的最优解，为后期的记忆化搜索或是状压 dp 做好准备。这点是我在此前自己的学习中没能找到的。

2.2 图论的进阶

在集训之前我的图论知识仅限于最短路、最小生成树等基础的算法，没有 LCA、tarjan 等算法的学习。对 SPFA、Dijkstra 等算法的理解也不够深入。

2.2.1 单源最短路径算法

单源最短路径又分为单源单汇与单源多汇问题，它们的复杂度几乎相同那么只考虑单源多汇问题。

Dijkstra 算法是一种基于贪心的最短路径算法，通过堆优化后可以实现 $\Theta(m \log n)$ 的复杂度。其算法实现流程如下：首先从起点出发，更新与起点相连所有点的 dist 值为起点到该点的距离，再选定全局中 dist 值最小的点重复上述过程，等到所有点都被作为起点拓展过一遍时算法结束。特别的该算法无法解决存在负边权的图的最短路问题。

该算法的证明也是显然的。

证明：对于上面的这一张图, 若设定 1 为起点，首先会将 3、5、6 节点更新，此时全局最小则为 6 号节点，可以证明在不存在负边权的情况下，起点到 6 的最短路径一定是从 1 到 6，这是因为任何其他到达 6 的路径都需要经过非 1 到 6 这条边，那么到达 6 时的距离一定会大于直接从 1 到 6。同理从 6 再更新时也是如此。更严谨的来说是，全局最小的 dist 值的点的最短路径是确定的，再利用该已确定最短路径的点进行拓展一定可确定另一个或在此之前的已确定的比通过该最小点拓展更小的点为下一次的出发点，该出发点的最短路径也是确定了的。

SPFA 是对 Bellman-Ford 算法的队列优化版本，其复杂度为 $\Theta(km)$，在段凡丁论文中提出 k 是一个极小的常数是错误的，该算法的最坏复杂度为 $\Theta(nm)$，不如 Dijkstra 算法稳定，但对于一般的图而言（例如 USACO 热浪），SPFA 效率是 Dijkstra 的许多倍。其算法思路类似于 Bellman-Ford 算法，记录通过几次中转到达某地的最优情况，思路类似于 BFS。队列优化后的代码非常好写。

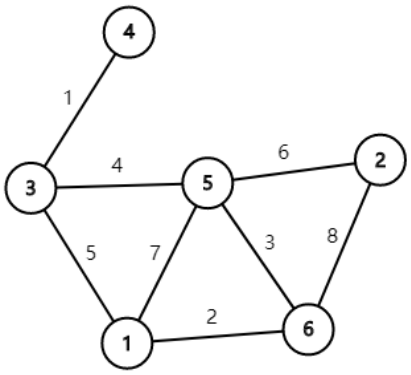

Figure 1: Dijkstra graph

2.2.2 最小生成树算法

kruskal 算法是基于贪心的最小生成树算法。其算法实现流程如下：将边按照边权从小到大排序，从最小的边开始加入生成树，如果当前边的两端点未联通则可以加入生成树不会形成环，直至加到第 $n-1$ 条边为止。算法的复杂度决定于排序与并查集大致为 $m \log m$。

证明 kruskal 的正确性在课堂上没有严格说明（?），参考《算法竞赛进阶指南》证明方法如下：

定理：任意一棵最小生成树一定包含无向图中权值最小的边。

证明：假设有一棵在无向图 G 中的最小生成树不包含权值最小的边。设边 $e(u,v,w)$ 为该无向图中权值最小的边。将 e 加入树种则会和树上从 u 到 v 的路径一起构成一个环，且环上其他边的权都比 w 大，那么 e 可以替代任何其他边形成一棵比当前最小更小的生成树，产生矛盾，则假设不成立，原命题成立。

有了上述的定理推导即可得出 kruskal 的正确性。

Prim 算法也基于上面的定理，不同的是其思路由向生成树中加边变为加点，实现流程类似于 Dijkstra 算法。每次选择距离已确定生成树最短的点加入生成树，用堆优化可以实现 $\Theta(m \log n)$ 的复杂度。

考虑 kruskal 算法与 Prim 算法的区别，Prim 在处理稠密图时效率高于 kruskal 算法。

2.2.3 倍增法求最近公共祖先

高效求解最近公共祖先的价值是巨大的。其中最简单的应用就是求解树任意两点之间的距离。

对于要求解节点 u 与节点 v 的最近公共祖先，朴素的思想是将 u 与 v 同时向上跳，并标记跳到过的点如果当任意一个点第一次跳到某一个被标记过的点时即为这两点的最近公共祖先。当树退化成链时求解的复杂度变为了 $\Theta(n)$。利用倍增思想优化是十分简单的，“即将 u、v 成倍地向上跳”。但是为了方便从上向下跳更为简单。首先将两点调至同一高度，由于知道高度差，这一步只需要对高度差进行二进制拆分即可。然后，将两点向上跳 $2^{\log n}$（即为一个极大的值，超过根就记为根）步，此时两点所到之处一定是相同，那么则说明跳多了，则将两点向上跳 $2^{\log n-1}$ 步，此时如果两节点不在同一位置这说明它们的公共祖先在它们原来位置的 2^{logn-1} 与 2^{logn} 之间。重复上述过程则可以在 $\Theta(\log n)$ 的级别下求出最近公共祖先。考虑上述算法的过程，只有在所到之点不同时两点才会向上跳即更新起点的位置，那么算法最后一定会停留在两点起点不相同但向上跳 2^0 所到之点就相同的位置，即为最近公共祖先的儿子，最后返回答案时返回其父亲即可。

还有一种效率极高但空间复杂度为 $O(n^2)$ 的最近公共祖先的算法。大概的思路是从叶子节点开始灌水，但空间复杂度极高几乎没用。

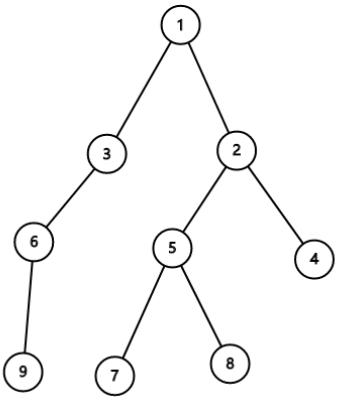

Figure 2: LCA graph

2.2.4 tarjan 求割点、割边、强连通分量

这是暑期所学图论知识最难的一部分了。tarjan 巧妙地利用深度优先搜索的时间戳解决了与联通性有关的大量问题。

对于上面 LCA graph 中的树，其各节点时间戳如下图：

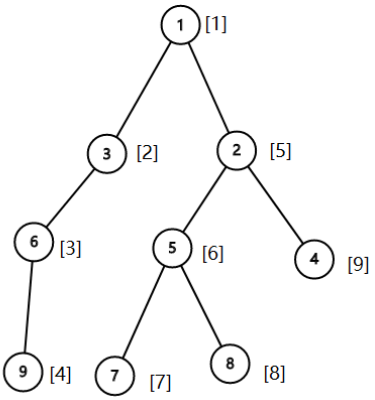

Figure 3: Tarjan graph

研究树的联通性问题是没有意义的，除了时间戳（dfn）外，另外引入一个追溯值（low）记录对于一个节点 u 其本身及其**深度优先搜索子树**所有点可以回到的最小的时间戳，对于一个普通的有向图其深度优先搜索时间戳 [] 及追溯值 () 为：

割点判定法则：在无向图中，当某一点 u，其任意一个出边所连的点 v 当且仅当满足 $dfn[u] <= low[v]$（该点有一个子树中所有的点都没有办法追溯回 u 的祖先，则当将 u 删除时该子树会断开）时 u 为该图的割点，特别的若 u 为根节点，只要有超过两个子节点即为割点。

割边判定法则：在无向图中，与求解割点类似的在深度优先搜索生成树中某一条边 (u, v)，当且仅当 $dfn[u] < low[v]$ 时，该边为割边。

强连通分量的求解方法：在有向图的深度优先搜索生成树中若存在某一点 u 当满足 $dfn[u] = low[u]$ 时，u 与其子树中所有的点都在同一个强连通分量中。利用栈记录下搜索顺序，当找到一个节点 u 时从栈顶到栈中的 u 之间的所有元素即为其子树，它们在同一个强连通分量中。

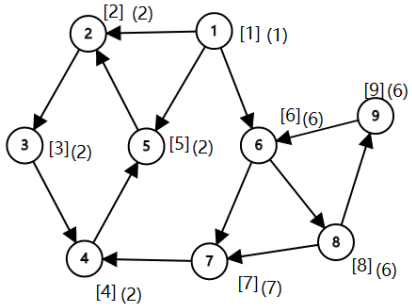

Figure 4: Tarjan graph

2.2.5 Kosaraju 求强连通分量

谭老师在课堂上详细地证明了 Kosaraju 的正确性，证明思路较为复杂。Kosaraju 与 Tarjan 思路相类似，首先在图中任选一个节点的作为起点开始 dfs，按照退出的顺序给节点编号就是搜索森林的后序序列。再每次选择未访问的标号最大的点进行搜索，其搜索树的节点都在同一个强连通分量中。

证明是比较复杂的。首先对于第一次搜索，编号小的到编号大的一定存在一条直接路径，除非它们不在同一棵树中。将图求逆后，编号小的一定不能再通过原来的路径到达编号大的，那么从编号大的节点开始 dfs 一遍，能够到达的节点在原图一定至少存在两条路径是让它们互相相连的。

2.3 数据结构的进阶

数据结构是高效的存储数据修改数据查询数据的方法，在集训之前我有了解基础的栈、队列、堆、树状数组等。通过集训我更深刻的了解了它们的实现原理并学习了更多的数据结构。

2.3.1 RMQ 区间最值

RMQ 是高效的静态区间最值查询算法，其预处理的复杂度为 $\Theta(n \log n)$，查询的复杂度为 $\Theta(1)$。应用范围较窄，仅限于最值的查询，但是代码实现容易。重要的是其倍增的思维。代码如下：

```
void init() {
        int t=log(n)/log(2)+1;

        REP(i,1,t-1) REP(j,1,n-(1<<i)+1)
                st[j][i]=max(st[j][i-1],st[j+(1<<(i-1))][i-1]);
}

int ask(int l,int r) {
        int t=log(r-l+1)/log(2);
        return max(st[l][t],st[r-(1<<t)+1][t]);
}
```

2.3.2 树状数组

树状数组巧妙地运用二进制的方法存储数列中的某一段，实现严格小于 $\log n$ 的修改与查询操作。

其 C 数组的定义规律如下：

$$C_k = \sum_{i=2^{lowbit(k)}}^{k} a_i$$

这样可以保证对于任何一段数列中的子序列 $\sum_{i=l}^{r} a_i$，都可以被拆分成至多 $\log(l-r+1)$ 段。

另外还有二维的树状数组，课堂上并没有讲解（？），但是课后的习题中有几道相关题目（POJ1195，POJ2155）。二维树状数组其实与一维是一致的，代码也十分相似。

2.3.3 二叉堆

堆是一个 $\Theta(\log n)$ 插入，$\Theta(1)$ 查询最值的数据结构。堆最重要的性质是父亲的权值不小于儿子的权值（大根堆）。堆的插入就依照其性质进行模拟即可。删除操作我们可以将其转换为将根替换为最后的一个节点再执行上述的插入操作即可。

一般的操作可以使用 C++STL 的优先队列实现。

2.4 递推与动态规划的初步

暑期简单讲解了递推算法与动态规划中一些简单的模型。

2.4.1 递推

递推算法基于的是数学思维，通常应用于存在一定规律的题目当中。能够推出答案满足的规律就能够写出对应的程序。

2.4.2 动态规划

动态规划与递推最大的不同是，其存在各种状态或者是“步骤”。对于每一个状态或“步骤”内的答案都是由递推得来。动态规划需要满足无后效性原则、最优子结构。也就是说，当前状态一定是通过其子状态推导得来，而前面的任何一步操作都是不会受到当前这一步所影响。

动态规划的简单模型如背包、最长上升子序列问题等上课时均有详细讲解，在考试中也频繁出现。这说明动态规划是极其重要的一部分。

2.5 数学的进阶

2.5.1 拓展欧几里得

问题如下：

求不定方程的整数解

$$ax + by = \gcd(a, b)$$

的解。

通过欧几里得原理：

$$\gcd(a, b) = \gcd(b, a\%b)$$

那么原式可化为：

$$bx' + a\%by' = gcd(b, a\%b)$$

那么只需求出 x 与 x'，y 与 y' 的关系即可：

$$bx' + a\%by' = gcd(b, a\%b) \tag{3}$$
$$= \gcd(a, b) = ax + by \tag{4}$$
$$\tag{5}$$

将含 a 与含 b 的合并

$$bx' + a\%by' = ax + by \tag{6}$$
$$bx' + ay' - \lfloor \frac{a}{b} \rfloor b \times y' = ax + by \tag{7}$$
$$b(x' - y - \lfloor \frac{a}{b} \rfloor \times y') + a(y' - x) = 0 \tag{8}$$

已知该式恒成立，则：

$$x = y' \tag{9}$$
$$y = x' - \lfloor \frac{a}{b} \rfloor \times y' \tag{10}$$

再利用辗转相除的函数递归计算 x，y 即可。
代码如下：
这里需要注意的是，递归的边界为 $x = 1, y = 0$ 时的一组特解。
那么将此式拓展为一般结论，即求解：

$$ax + by = c$$

首先讨论有无解，当 $\gcd(a, b) \nmid c$ 一定无解。另 $k = \frac{c}{\gcd(a,b)}$：

$$ax'k + by'k = \gcd(a, b) \times k = c$$

就得到了 $x = x'k$，$y = y'k$。
如果要求 x 非负且最小，另 $t = \frac{b}{\gcd(a,b)}$，则 $(x\%t + t)\%t$ 就是 x 的最小非负解。加 t 主要是为了处理负数的问题。

2.5.2 Chess Queen 的推导

李瑞晨在课堂上的推导方法过于复杂，这里有一个我总结后的简易版本。已经得到了

$$ans_1 = C_m^1 A_n^2 = nm(n-1)$$

$$ans_2 = C_n^1 A_m^2 = nm(m-1)$$

ans1 与 ans2 为在同一行与同一列互相攻击的皇后数
ans3 为在同一斜列的互相攻击皇后数显然

$$ans_3 = 2 \times [A_2^2 + A_3^2 + ... + A_n^2 \times (m - n + 1) + ... + A_3^2 + A_2^2]$$

将此式展开

$$ans_3 = [1 \times 2 + 2 \times 3 + ... + (n-1) \times n] \times 4 + 2n(n-1)(m-n)$$

前半部分就像小学奥数中的通项求和问题了

$$ans_3 = \sum_{i=1}^{n-1} i(i+1) \times 4 + 2n(n-1)(m-n) \tag{11}$$

$$= \sum_{i=1}^{n-1} i^2 \times 4 + \sum_{i=1}^{n-1} \times 4 + 2n(n-1)(m-n) \tag{12}$$

因为

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(n+2)}{3} - \frac{n(n+1)}{2} \tag{13}$$

$$= \frac{n(n+1)(2n+1)}{6} \tag{14}$$

由 (14)，得

$$ans_3 = [\frac{n(n-1)(2n-1)}{6} + \frac{n(n-1)}{2}] \times 4 + 2n(n-1)(m-n) \tag{15}$$

$$= \frac{2n(n-1)(2n-1)}{3} + 2n(n-1)(m-n) \tag{16}$$

最后再把 ans_1，ans_2，ans_3 加起来即可

3 学习规划

暑假的学习已经帮助我初步地认识到了算法及其一些基础的应用。在未来还需要学习省选的知识，并不断提升自己的数学思维能力。

3.1 近期规划

- 在 Vjudge 上将“挑战程序设计竞赛（第二版）题集”尽量完成
- 看《组合数学引论》
- 尽量地做一些数据结构的难题目，夯实数据结构的基础。学习数据结构的灵活用法
- 尽量地做一些有一定难度的动态规划并学习状态压缩动态规划、树形动态规划、动态规划的各种优化等
- 跟着学长的题单做一些有难度的图论题目
- 在实战中学习，尽量参加每周的 Atcoder 比赛与 Codeforces 比赛
- 改正好考试中错的每一道题目并总结出错的原因
- 提高代码能力，提高细心程度。应对例如“时间复杂度”之类的题目

3.2 远期规划

- 将《算法竞赛进阶指南》中的全部题目独立完成
- 将《算法竞赛入门经典》及《算法竞赛入门经典：训练指南》中的题目大部分完成

4 后记

通过这一次对暑期学习的总结，我发现自己仍存在许多对知识的误解与盲区。遇到这类的问题，通常可以通过搜索相关资料而很快地解决。而如果没能够及时地进行复习总结，这些被我遗漏的知识可能会在很久以后才能被发现。到那个时候很可能会导致考试失分产生不好的结果。这让我意识到了，在学习知识的时候一定要扎实，弄清楚每一个环节，这样才能保证没有知识的遗漏，才可以保证考试的成功。

后续还有更多知识的学习，例如数据结构中的线段树、各种树形结构，动态规划中的树形 dp、状压 dp、动态规划的各种优化。希望自己能够从容应对！