数据结构与算法第五次作业

Shuxin Chen, 2015013229

2017 年 4 月 9 日

1. 给出一个有效算法，计算出相应于某给定模式 P 的字符串匹配自动机的转移函数 δ。所给出的算法的运行时间应该是 $O(m|\Sigma|)$。

   我们首先证明，如果 $q = m$ 或 $P[q+1] \neq a$，则 $\delta(q,a) = \delta(\pi[q],a)$。若此假设是正确的，由于 $\pi[q] \leq q$ 恒成立，我们只需要使用两重循环分别枚举字符串的每一位和字符集中的每一个字母即可，时间复杂度为 $O(m|\Sigma|)$。

   我们的证明基于 KMP 算法的正确性。在 KMP 算法中，如果模式串匹配到了位置 q，文本串当前的字母为 a，那么这一步会在 $\pi^*[q] = \{q, \pi[q], \pi[\pi[q]], \cdots\}$ 集合中找到一个最大的位置 $\pi_0[q]$，使得该位置的下一个字母是 a，用自动机中的 δ 转移表示即为 $\delta(q,a) = \pi_0[q] + 1$。若找不到，$\delta(q,a) = 0$。我们基于此正确性，通过数学归纳法来证明猜想。

   当 $q = 0$ 时，显然 $\pi[q] = 0$，那么 $\delta(q,a) = \delta(\pi[q],a)$ 恒成立。在后续的算法实现中，我们会定义 $\delta(q,a) = 0$，这也是符合常理的，因为 $q = 0$ 且 $P[1] \neq a$ 时，模式串的第一个字符就无法匹配，因此转移函数的值只能为 0。

   假设对于 $\forall q < q_0 < m$，均有 $\delta(q,a) = \delta(\pi[q],a)$。那么当 $q = q_0$ 时，设 $t = \pi[q]$，根据 KMP 算法，有 $\pi^*[q] = \pi^*[t] + \{q\}$。因为 $P[q+1] \neq a$，所以 $\pi_0[q] \neq q$，这样就有 $\pi_0[q] \in \pi^*[t]$。那么根据定义，$\pi_0[q]$ 和 $\pi_0[t]$ 都是在 $\pi^*[t]$ 集合中找到的最大的位置，使得该位置的下一个字母是 a，因此我们有

   $$\delta(q,a) - 1 = \pi_0[q] = \pi_0[t] = \delta(\pi[q],a) - 1$$

   于是有 $\delta(q,a) = \delta(\pi[q],a)$。

   当 $q_0 = m$ 时，因为第 $m+1$ 位不存在，所以 $\pi^*[q]$ 中不能含有 q，此时 $\pi^*[q] = \pi^*[t]$，仍然有 $\delta(q,a) = \delta(\pi[q],a)$。根据归纳假设，猜想得证。

现在我们可以开始设计算法了。首先使用 KMP 算法计算出 π 数组。接着 $\forall\sigma \in \Sigma$，初始化 $\delta(0,\sigma) = 0$。然后从小到大枚举位置 q，$\forall\sigma \in \Sigma$，若 $q = m$ 或 $P[q+1] \neq \sigma$，则表示匹配失败，$\delta(q,\sigma) = \delta(\pi[q],\sigma)$，否则匹配成功，$\delta(q,\sigma) = q+1$。以下是该算法的伪代码。

```
AUTOMATA-MATCH(P, m)
 1  π[1..m] = KNUTH–MORRIS–PRATT(P, m)
 2  foreach σ in Σ
 3      if P[1] == σ
 4          δ(0, σ) = 1
 5      else
 6          δ(0, σ) = 0
 7  for q = 1 to n
 8      foreach σ in Σ
 9          if q = m or P[q + 1] ≠ σ
10              δ(q, σ) = δ(π[q], σ)
11          else
12              δ(q, σ) = q + 1
13  return δ
```

2. 基于重复因子的字符串匹配

   (a) 我们来探究重复因子和 KMP 算法中 π 数组的关系。令 $u = \rho(x)$，将 x 表示成 $x = y^u = yy\cdots yy$，那么 $\pi[|x|] \geq |x| - |y|$，因为 x 的 $|x| - |y|$ 前缀等于 $|x| - |y|$ 后缀。据此我们可以看出，若 $\pi[|x|] < \frac{1}{2}|x|$，那么 $u = 1$。

   考虑 $u > 1$ 的情况，此时 $\pi[|x|] \geq \frac{1}{2}|x|$，且 $\pi[|x|] \geq |x| - |y|$。若 $w = \pi[|x|] > |x| - |y|$，令 $v = |x| - w$，根据 π 的定义，有 $\forall p \leq w$，有 $x[p] = x[p+v]$。由于 $x = y^u$，令 $w_0 = |x| - |y|, v_0 = |y|$，我们还有 $\forall p \leq w_0$，有 $x[p] = x[p+v_0]$。由于 $|x|$ 是 v_0 的倍数，且 $v < v_0$，令 $v_1 = gcd(v, v_0)$，表示 v 和 v_0 的最大公约数，那么 $\forall p \leq |x| - v_1$，有 $x[p] = x[p+v_1]$。因为 v_1 是 v 的约数，所以 v_1 也是 $|x|$ 的约数，那么 x 也能表示成 $x = z^{|x|/v_1}$，由于 $v_1 < v$，这与 ρ 函数的最大性质矛盾！因此 $\pi[|x|] > |x| - |y|$ 为假，只能有 $\pi[|x|] = |x| - |y|$，稍作变换可得 $|y| = |x| - \pi[|x|]$，这就是我们即将设计的算法的基础。

根据上述证明，我们可以设计求出 ρ 的算法。对于模式串 $P[1..m]$，求出它的 KMP 数组 π，对于每一个位置 q，令 $l = q - \pi[q]$，若 q 能整除 l，那么 $\rho[q] = q/l$，否则 $\rho[q] = 1$。该算法之比 KMP 算法多了一条语句，因此伪代码略，算法的时间复杂度为 $O(m)$。

(b) 我们用 $P(\rho^*(P) = k)$ 表示概率。

当 $k = 1$ 时，令 $P(\rho^*(P) = 1) = c \leq 1$。

当 $k > 1$ 时，这个概率可以通过枚举 k 值取到的位置坐标算出，显然只有 $k, 2k, \cdots$ 这些位置的 ρ 值可以取到 k。通过放缩，我们可以得到

$$\begin{aligned} P(\rho^*(P) = k) &\leq \sum_{i=1}^{\lfloor \frac{m}{k} \rfloor} \frac{2^i * 2^{m-ki}}{2^m} \\ &= \sum_{i=1}^{\lfloor \frac{m}{k} \rfloor} \frac{2^i}{2^{ki}} \\ &\leq \frac{\frac{1}{2^{k-1}}}{1 - \frac{1}{2^{k-1}}} \\ &= \frac{1}{2^{k-1} - 1} \\ &\leq \frac{1}{2^{k-2}} \end{aligned}$$

由期望的公式，可知

$$\begin{aligned} E(\rho^*(P)) &= c * 1 + \sum_{k=2}^{m} P(\rho^*(P) = k) * k \\ &= c + \sum_{k=2}^{m} \frac{k}{2^{k-2}} \\ &< c + 8 \end{aligned}$$

因此，$\rho^*(P)$ 的期望值为 $O(1)$。

(c) 我们首先证明该算法的正确性，可以使用反证法。该算法的关键在于当出现不匹配时，会令 s（起始匹配位置）后移 $\lceil q/k \rceil$ 步。假设这后移的 $\lceil q/k \rceil$ 步是不合理的，它导致有一个正确的位置 s 被跳过，以 s 作为起始位置的 m 个元素可以和 P 匹配，即 $\forall 0 \leq i < m$，有 $T[s + i] = P[i + 1]$。

我们再令 s_0 为后移之前的起始位置，因为 s 被跳过，所以它离 s_0 的距离不会超过后移的长度，即 $l = s - s_0 < \lceil q/k \rceil$。由于 l 是整数，因此有 $l \leq \lceil q/k \rceil - 1$。

与 (a) 问的方法相似，我们可以得出，模式串中的前 $q+1$ 个字符具有长度为 l 的循环节，即具有重复因子 $\lceil\frac{q+1}{l}\rceil$。根据不等式，我们有

$$\rho^*(p) \geq \lceil\frac{q+1}{l}\rceil \geq \frac{q+1}{l} \geq \frac{q+1}{\lceil\frac{q}{k}\rceil - 1} > \frac{q+1}{\frac{q}{k}} > k = 1 + \rho^*(p)$$

矛盾！由反证法可知该算法是正确的。

接下来我们来证明算法的时间复杂度。在整个 while 循环中，只有 q 和 s 的累加可以贡献时间复杂度，其中 s 至多累加 $n-m$ 次，即 $O(n)$，而 q 的累加基于 s 的累加，根据 s 每次累加的步数，我们可以得出，$\sum\lceil\frac{q}{k}\rceil = O(n-m) = O(n)$，即 q 的累加次数为 $O((1+\rho^*(P))n) = O(\rho^*(P)n)$。此外，这个算法还有一部分的时间复杂度在于求出 $\rho^*(P)$ 的过程，由 (a) 问可知其时间复杂度为 $O(m)$。三部分时间复杂度相加，即可得出整个算法的时间复杂度为 $O(\rho^*(P)n+m)$。

3. 独立集

(a) 我们将独立集问题变成一个判定性问题，即给出图 $G=(V,E)$，判定图中是否有一个规模为 k 的独立集。其形式定义为 INDSET $= \{\langle G,k\rangle : G$ 是一个包含规模为 k 的独立集的图 $\}$。

首先我们证明独立集问题是 NP 问题，即在多项式时间内可以判定。我们给出一个顶点集合 V'，且有 $|V'|=k$，判定该集合是否是一个独立集。我们只需要枚举 E 中的每一条边，判断它是否仅与 V' 中的一个顶点相关联。这种判定显然是多项式时间内可以完成的，因此独立集问题是 NP 问题。

接下来我们用最大团问题来归约独立集问题，以此证明独立集问题是 NPC 问题。我们证明 G 中存在规模为 k 的团当且仅当 $\overline{G}$ 中存在规模为 k 的独立集。在证明之前，我们观察独立集的定义，它的定义为图中的每一条边至多与独立集中的一个顶点相关联，这个定义等价于，独立集中的任意两个点之间都没有边相连。

若 G 中存在规模为 k 的团，设这个团的顶点集合为 V_0，那么 V_0 中的任意两个点之间都有边相连，因此在 $\overline{G}$ 中，V_0 中的任意两个点之间都没有边相连，即 V_0 为规模为 k 的独立集。由于证明中的所有过程都是可逆的，因此反之亦然，我们通过归约的方法证明了独立集问题是 NPC 问题。

(b) 我们设黑箱为 JUDGE(G,k)，它会判定图 G 中是否存在规模为 k 的独立集，若存在，它会返回 1，否则返回 0。

首先通过二分 k 的方式（k 的范围显然为 $1\sim|V|$ 之间的正整数），得到 G 中最大独立集的规模 k_0，这一步的时间复杂度为 $O(|V|)$。

然后我们依次枚举 $|V|$ 中的每一个节点 s，设与 s 关联的边的集合为 E'，构造新图 $G' = (V - \{s\}, E - E')$，构造的时间复杂度显然是多项式级别的。我们调用 $\textsc{Judge}(G', k_0)$，若它返回 1，说明删除 s 并不会影响这个图的最大独立集，因此我们永久删除这个点，即令 $G = G'$。若它返回 0，说明 s 一定出现在这个图的最大独立集中，并不能删除，因此不做任何操作。当枚举完所有节点之后，得到最终的图 G 的顶点集合 V 即为一组规模为 k_0 的最大独立集。

因为该算法中的每一步操作都是多项式级别的，它们的乘积也是多项式级别的，因此整个算法的运行时间是关于 $|V|$ 和 $|E|$ 的多项式。

(c) 首先，若 G 中每个顶点的度数均为 2，那么 G 是若干个互不相交的环的并。这样以来，我们可以把 G 的最大独立集问题分解成每个环的最大独立集问题。在求出每个环的最大独立集之后，取它们的并即为 G 的最大独立集。

对于一个环来说，根据独立集的定义（无论使用原始定义还是在 (a) 问中等价的定义），我们可以得到结论，两个相邻的节点不能同时出现在独立集中，因此当环的大小为 n 时，这个环的独立集大小为 $\lfloor n/2 \rfloor$，将任意一个编号为 1，再将与 1 相邻的某个节点编号为 2，按照这个顺序依次编号。取编号为奇数的所有节点，即为一种可行的独立集。

该算法只需要一遍 dfs 即可，因此时间复杂度为 $O(|V|)$。

(d) 我们首先来计算二分图独立集的上界。假设二分图的节点数共为 m，最大匹配数为 n，那么不在匹配中的 $m - 2n$ 个节点之间一定都没有边相连（如果某两个点之间有边相连，就可以加入匹配了）。而在匹配中的 $2n$ 个节点，最多只能选出其中的 n 个节点。在最好的情况下，选出的这 n 个节点与不再匹配中的 $m - 2n$ 个节点可以组成一个独立集，它的规模 $n + m - 2n = m - n$ 即为二分图独立集的上界。

随后我们来构造一种可行的规模为 $m - n$ 的独立集的方案，为此我们先引入算法导论中的**顶点覆盖问题**，即**最小点覆盖**。对于任意一个点覆盖，设它取出的顶点集合为 V'，可以发现，$V - V'$ 中任意两个节点都没有边相连（如果某两个点之间有边相连，那么这条边必须被覆盖，它的两个端点中至少有一个在 V'，这就出现了矛盾），那么 $V - V'$ 是一个独立集。如果我们能构造出规模为 n 的点覆盖，那么我们就能由此得出规模为 $m - n$ 的独立集。

构造规模为 n 的点覆盖的方法相对简单一些。设图 G 为二分图，且是连通的（若不连通，可以将 G 分成若干个连通分支，对每个连通分支分别构造点覆盖，最后合并），设 M 为二分图的某个最大匹配，P 是关于 M 的交互道路，即

$P = a_1b_1a_2b_2 \cdots a_nb_n$，$P$ 中的 $2n$ 个节点即为在匹配中的 $2n$ 个节点。我们接下来证明，所有不在匹配中的节点（若存在）要么只和 a 类节点有边相连，要么之和 b 类节点有边相连，即不在匹配中的节点只会和二分图中的一边的节点相连。

我们使用反证法，假设有两个不在匹配中的节点 c_1, c_2，c_1 和 a_i 有边相连，c_2 和 b_j 有边相连，不妨设 $i \leq j$，此时 $c_1a_ib_ia_{i+1}b_{i+1} \cdots a_jb_jc_2$ 组成一条增广路，说明 M 不是最大匹配，矛盾！因此不在匹配中的节点只会和二分图中的一边的节点相连。这样以来，我们只要选取与不在匹配中的节点有边相连的那一类节点（a 类或者 b 类），就构造出了一个规模为 n 的点覆盖。因为每一类节点都恰好有 n 个，这表明我们恰好选出了 n 个节点。并且，因为 G 中的边只有两类，一是两个端点都是匹配节点，二是一个端点是匹配节点另一个是非匹配节点。第一类边的两个端点属于不同的类，那么它们必然都被覆盖，第二类边根据上面的证明，也同样都被覆盖，因此我们构造出了一个规模为 n 的点覆盖。最后我们取出剩下的 $m - n$ 个节点，它们构成了一个规模为 $m - n$ 的独立集。

最后我们来分析算法的时间复杂度，我们用匈牙利算法求出二分图的最大匹配，时间复杂度为 $O(m^3)$。由最大匹配得出点覆盖这一步只需要对图进行一次遍历即可，时间复杂度为 $O(m^2)$，由点覆盖得出独立集这一步只需要对节点取反，时间复杂度为 $O(m)$，因此算法的总时间复杂度为 $O(m^3)$。