第十次作业

崔浩 2018214160

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

算法描述:
在计算转移函数时，如果$P[q+1]=a$，则$\delta(q,a)=q+1$，否则$\delta(q,a)=\delta(\pi[q],a)$

算法正确性证明:
如果$q\neq m, P[q+1]\neq a$, 根据后缀函数递归引理，因为$\pi[q]=\sigma(P_q)$，可以推出$\sigma(P_q a)=\sigma(P_{\pi[q]}a)$，因此$\delta(q,a)=\delta(\pi[q],a)$
如果$q\neq m, P[q+1]=a$，说明当前字符匹配成功，$\delta(q,a)=q+1$
如果$q=m$，字符串匹配完成，需要等待继续匹配，$\delta(q,a)=\delta(\pi[q],a)$

算法伪码:

```
COMPUTE-TRANSITION-FUNCTION-QUICK(P, Σ):
    m=P. length
    π=COMPUTE-PREFIX-FUNCTION(P)
    for each a in Σ
        δ(0,a) = 0
    for q=1 to m
        for each a in Σ
            if q = m or P[q + 1] ≠ a
                δ(q,a) = δ(π[q],a)
            else
                δ(q,a) = q + 1
    return δ
```

复杂度分析:
COMPUTE-PREFIX-FUNCTION(P)复杂度为$\Theta(m)$，计算$\delta(0,a)$复杂度为$\Theta(|\Sigma|)$，两重循环复杂度为$\Theta(m|\Sigma|)$，总的时间复杂度为$O(m|\Sigma|)$

2. 基于重复因子的字符串匹配

A． 写出一个有效算法计算出$\boldsymbol{\rho(P_i)(i=1,2,\ldots,m)}$，算法的输入为模式$\mathbf{P[1..m]}$。算法的运行时间是多少？

算法描述:
计算$l=i-\pi[i]$，判断出循环的长度，如果$i \bmod l=0$，并且$p=i-j\times l>0, p-\pi[p]=l$，那么$\rho(P_i)=i/l$，否则$\rho(P_i)=1$

算法伪码:

```
CALCULATE-REPEAT(P):
```

```
m=P. length
for i=1 to m:
      π=COMPUTE-PREFIX-FUNCTION(Pi)
      l=i -π[i]
      if i mod l != 0
            return 1
      for j=1 to i
            p=i-j*l
            if p <= 0
                  break
            if  p − π[p] ≠ l
                  return 1
      ρ(P_i) = i/l
      return  ρ
```

复杂度分析：计算$\rho(P_i)$的复杂度为$O(m)$

B．对任何模式$P[1..m]$，设$\rho^*(P)$定义为$\max\rho(P_i)$。证明：如果从长度为 m 的所有二进制字符串所组成的集合中随机选择模式 P，则$\rho^*(P)$的期望值为$O(1)$

当 m=1 时，$E(\rho^*(P)) = 1$

假设当 m=k 时， $E(\rho^*(P)) = \frac{1}{2^k}\cdot\Sigma\rho^*(P) = O(1)$

当 m=k+1 时，$E(\rho^*(P')) = \frac{1}{2^{k+1}}\cdot\Sigma\rho^*(P') = \frac{1}{2^{k+1}}\cdot\Sigma 2\cdot(\rho^*(P)+1) = \frac{1}{2^k}\cdot\Sigma(\rho^*(P)+1) = \frac{1}{2^k}\cdot\Sigma\rho^*(P)+1 = O(1)+1 = O(1)$

因此$\rho^*(P)$的期望值为$O(1)$

C． 论证下列字符串匹配算法可以在$O(\rho^*(P)n+m)$的时间内正确找出模式 P 在文本 T 中出现的所有位置。

证明:

时间复杂度:

在检测到不匹配时，需要移位到已知匹配的前缀位置，总的匹配次数不超过 k*(n-m)

因此匹配阶段的复杂度为$O(k(n-m)) = O(\rho^*(P)(n-m)) = O(\rho^*(P)n)$

计算$\rho^*(P)$的复杂度为$O(m)$，因此总的算法复杂度为$O(\rho^*(P)n+m)$

算法正确性:

在当前字符匹配，也就是 T[s+q+1]=P[q+1]时，q=q+1，可以继续匹配。

在当前字符不匹配时, 向右移动的次数不能超过最小的重复因子长度 L, 因为$k = 1+\rho^*(P)$, q/k<=L，所以向右移动的字符数设置为 max（1,q/k）是安全的，因此算法是正确的。

3. 最长简单回路问题是在一个图中，找出一个具有最大长度的简单回路（无重复顶点），证明：这个问题是 NP 完全的。

（1）最长简单回路问题可以被规约为最长简单路径问题

复制图 G 为一个新图 G’，在 G’中添加|V|个顶点 w_1，w_2，⋯，$w_{|V|}$，其中 w_i 到 w_{i+1} 有一条有向边，v 到 w_1 有一条有向边，$w_{|V|}$到 u 有一条有向边，在 G’中的最长简单回路包含点 u， v，u->v 就是原图 G 中 u 到 v 的最长简单路径。

建立新图的时间复杂度为$O(|V|+|E|)$

只要 u 到 v 存在一条路径，那么图 G’中的环必然是最大长度的简单回路，因为新添加的|V|个点只在这条回路中，这条回路的点数至少为|V|+2,而其他回路的点数不超过|V|，所以该回路为 G’中的简单回路。如果 u->v 存在一条更长的路径，那么图 G’中的回路就不是最长的，这是一个矛盾，所以 u->v 的路径就是图 G 中的最长简单路径问题。

（2）最长简单路径问题是一个 NP 完全问题。

最长简单路径问题可以被规约为哈密顿回路问题。图中存在一条哈密顿回路当且仅当最长简单路径的长度为|V|-1。如果存在一条哈密顿回路，那么删去任意一条边，形成的路径就是最长简单路径，点数为|V|，必然是最长的。如果最长简单路径小于|V|-1，那么经过的点数小于|V|，则不可能存在哈密顿回路。因为哈密顿回路的存在问题是一个 NP 完全问题，所以最长简单路径问题是一个 NP 完全问题。

因此最长简单回路是一个 NP 完全问题。

4. 说明如何实现 GREEDY-SET-COVER，使其运行时间为$O(\sum_{S\in\mathcal{F}}|S|)$

重点在于如何选择$s\in\mathcal{F}$使得$|S\cap U|$最大，以及如何删除 U 中 S 的元素。

可以创建两个数据结构来存储这些数据，创建字典 D 存储每个元素出现在哪些集合中，创建字典 L 存储每个长度有哪些集合。

算法伪代码为：

```
GREEDY-SET-COVER(X, 𝓕):
U=X
E=∅
create dict D which is an array of list
create dict L which is an array of list
for each S of index i in F:
        for each a in S:
                D[a].append(i)
        L(len(S)).add(i)

for length in [max(len(S)), ……, 1, 0]:
        if length in L
                P=L[length]
                while len(P)!=0:
                        x=P.pop()
                        E.append(x)
                        for a in F[x]:
                                for y in D[a]:
                                        S’=F[y]
                                        remove y from L[len(S’)]
                                        remove a from S’
                                        L[len(S’)].add(y)
```