Implementation of Hopcroft's Algorithm

Hang Zhou

ENS

7 January 2009

Introduction

Definition

Let $\mathcal{P}$ be a partition of Q. The number of classes in $\mathcal{P}$ is bounded by N. Hence every class of $\mathcal{P}$ can be entitled with a unique name between 1 and N.

Definition

Let $\mathcal{A}$ be a deterministic finite automaton (Q, A, E, I, F) and let $a \in A$ and $B, C \subseteq Q$. Then B is split by (C, a), if $B \cdot a \nsubseteq C$ and $B \cdot a \cap C \neq \emptyset$ with $B \cdot a = \{x \cdot a | x \in B\}$.

Hopcroft's Algorithm - First Version

ALGORITHM1()

$\mathcal{P} \leftarrow \{F, F^c\}$
$S \leftarrow \emptyset$
for $a \in A$
do INSERT $(min(F, F^c), a)$ to S
while $S \neq \emptyset$
do DELETE (C, a) from S
    $\mathcal{T} \leftarrow \{B \mid B \text{ is split by } (C, a)\}$
    for each $B \in \mathcal{T}$
    do $B', B'' \leftarrow$ SPLIT(B, C, a)
        REPLACE B by B' and B'' in $\mathcal{P}$
        for $b \in A$
        do if $(B, b) \in S$
            then REPLACE (B, b) by (B', b) and (B'', b) in S
            else INSERT $(min(B', B''), b)$ to S

Extreme Condition

Partition in this moment : 1234 - 5

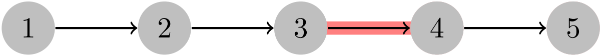
Extreme Condition

Partition in this moment : 1234 - 5

Extreme Condition

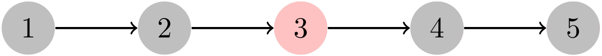
Partition in this moment : 123- 4 - 5

Extreme Condition



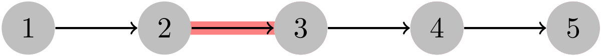
Partition in this moment : 123- 4 - 5

Extreme Condition



Partition in this moment : 12 - 3 - 4 - 5

Extreme Condition



Partition in this moment : 12 - 3 - 4 - 5

Extreme Condition

Partition in this moment : 1 - 2 - 3 - 4 - 5

Extreme Condition

Partition in this moment : 1 - 2 - 3 - 4 - 5

Extreme Condition

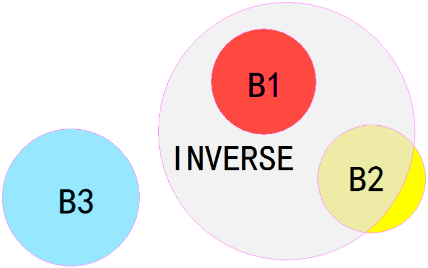
Partition in this moment : 1 - 2 - 3 - 4 - 5

The Question Throughout the Writing

What's the execution time of the *while* loop if it is carried out efficiently?

Idea:

- define INVERSE $= a^{-1}C$
- wish to prove $O(|\text{INVERSE}|)$ complexity of *while* loop

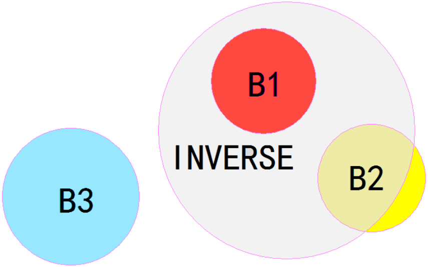


Difficulty

We cannot directly obtain $\mathcal{T}$ in $O(|\textsc{Inverse}|)$, what to do?

Solution:

- Instead, we calculate its approximate substitute $\mathcal{T}'$:
$$\mathcal{T}' = \{B \mid B \bigcap \textsc{Inverse} \neq \emptyset\}$$
- $O(1)$ time to check whether an element in $\mathcal{T}'$ is also in $\mathcal{T}$



Hopcroft's Algorithm - Second Version

ALGORITHM2()

$\mathcal{P} \leftarrow \{F, F^c\}$
$S \leftarrow \emptyset$
for $a \in A$
do INSERT $(min(F, F^c), a)$ to S
while $S \neq \emptyset$
do DELETE (C, a) from S
    INVERSE $\leftarrow a^{-1}C$
    $\mathcal{T}' \leftarrow \{B \mid B \bigcap \text{INVERSE} \neq \emptyset\}$
    for each $B \in \mathcal{T}'$
    do if $B \cdot a \nsubseteq C$
        then $B', B'' \leftarrow$ SPLIT(B, C, a)
            REPLACE B by B' and B'' in $\mathcal{P}$
            for $b \in A$
            do if $(B, b) \in S$
                then REPLACE (B, b) by (B', b) and (B'', b) in S
                else INSERT $(min(B', B''), b)$ to S

Calculate Inverse efficiently

By definition, $\textsc{Inverse} = a^{-1}C$.

We use a two-dimensional array Inv :

- Inv[x,a] :

  a pointer to the linked list of all states y, such that $y \cdot a{=}x$

- So we have :

$$\text{for all } x \in \textsc{Inverse}$$
$$\Updownarrow$$
$$\text{for all } y \in C \text{ and all } x \in \texttt{Inv}[y, a]$$

Support list operations efficiently

List operations includes:

- **search** an element
- **insert** an element
- **delete** an *arbitrary* element
- test whether the list is **empty**

We use a linked list `S` and an array `InList` together, s.t.

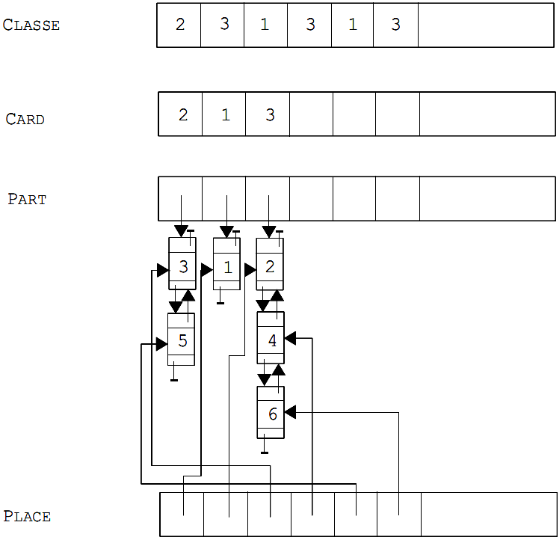
$$\texttt{InList[C,a]} = \begin{cases} \text{a pointer to the position of (C,a) in S} & \text{if } (C,a) \in S \\ \text{Null} & \text{if } (C,a) \notin S \end{cases}$$

$O(1)$ time of all operations above

Characterize a Partition

- `Class[x]` : the name of the class which contains state x;
- `Part [i]` : a pointer to the doubly linked list with all states in class i;
- `Card [i]` : the size of class i;
- `Place[x]` : a pointer to the position of state x in the linked list `Part[p]`, where `p=Class[x]`.

Example: Q={1, .., 6} and $\mathcal{P}$: 35-1-246



Split the Classes

- `Involved` : a linked list of the names of classes in $\mathcal{T}'$;
- `Size[i]` : the size of $B \bigcap a^{-1}C$, where B is the class i;
- `Twin[i]` : the name of the newly created class while splitting the class i.

Noting: They should be cleared every time the *while* loop is executed.

Time Complexity - Part 1

ALGORITHM2()

```
P ← {F, F^c}
S ← ∅
for a ∈ A
do INSERT (min(F, F^c), a) to S                    O(M)
```

Time Complexity - Part 2

ALGORITHM2()

$\mathcal{P} \leftarrow \{F, F^c\}$
$S \leftarrow \emptyset$
for $a \in A$
do INSERT $(min(F, F^c), a)$ to S $O(M)$

while $S \neq \emptyset$
do DELETE (C, a) from S $O(MN)$?

Time Complexity - Part 3

ALGORITHM2()

$\mathcal{P} \leftarrow \{F, F^c\}$
$S \leftarrow \emptyset$
for $a \in A$
do INSERT $(min(F, F^c), a)$ to S $\quad O(M)$

while $S \neq \emptyset$
do DELETE (C, a) from S $\quad O(MN)?$

  INVERSE $\leftarrow a^{-1}C$
  $\mathcal{T}' \leftarrow \{B \mid B \bigcap \text{INVERSE} \neq \emptyset\}$
  for each $B \in \mathcal{T}'$
  do if $B \cdot a \nsubseteq C$
    then $B', B'' \leftarrow \text{SPLIT}(B, C, a)$
      REPLACE B by B' and B'' in $\mathcal{P}$ $\quad O(MNlogN)?$

Time Complexity - Part 4

ALGORITHM2()

$\mathcal{P} \leftarrow \{F, F^c\}$

$S \leftarrow \emptyset$

for $a \in A$

do INSERT $(min(F, F^c), a)$ to S $\quad O(M)$

while $S \neq \emptyset$

do DELETE (C, a) from S $\quad O(MN)?$

    INVERSE $\leftarrow a^{-1}C$

    $\mathcal{T}' \leftarrow \{B \mid B \bigcap \text{INVERSE} \neq \emptyset\}$

    for each $B \in \mathcal{T}'$

    do if $B \cdot a \nsubseteq C$

        then $B', B'' \leftarrow$ SPLIT(B, C, a)

            REPLACE B by B' and B'' in $\mathcal{P}$ $\quad O(MNlogN)?$

            for $b \in A$

            do if $(B, b) \in S$

                then REPLACE (B, b) by (B', b) and (B'', b) in S

                else INSERT $(min(B', B''), b)$ to S $\quad O(MN)?$

Analysis of Time Complexity

Proposition

The number of pairs (C, a) inserted into S is at most $2MN$.

Lemma

The number of classes created during the execution is at most $2N - 1$.

Corollary

The number of iterations of the while loop is at most $2MN$.

Analysis of Time Complexity

Why does **Part 3**(i.e. bottleneck) have global execution time $O(MNlog_2N)$?

We take the following two steps:

- Evaluate $\sum |\textsc{Inverse}|$
  (i.e. the sum of $|\textsc{Inverse}|$ in all executions of the *while* loop)
- Ensure the global execution time to be $O(\sum |\textsc{Inverse}|)$

Bottleneck Analysis - Evaluation of $\sum |\textsc{Inverse}|$

Proposition

When $a \in A$ and $p \in Q$ are fixed, the number of (C, a) being ***deleted*** *from list S, such that $p \in C$, is bounded by $log_2 N$.*

Corollary

$\sum |\textsc{Inverse}|$ *is bounded by $MNlog_2 N$.*

Bottleneck Analysis - Implementation - Part 3.1

ALGORITHM3()

```
create an empty list Involved
for all y ∈ C and all x ∈ Inv[y, a]
do i ← Class[x]
    if Size[i] = 0
      then Size[i] ← 1
             insert i to Involved
      else Size[i] ← Size[i] + 1
```

Bottleneck Analysis - Implementation - Part 3.2

```
ALGORITHM3()
 create an empty list Involved
for all y ∈ C and all x ∈ Inv[y, a]
do i ← Class[x]
    if Size[i] = 0
      then Size[i] ← 1
             insert i to Involved
      else Size[i] ← Size[i] + 1

for all y ∈ C and all x ∈ Inv[y, a]
do i ← Class[x]
    if Size[i] < Card[i]
      then if Twin[i] = 0
              then Counter ← Counter + 1
                   Twin[i] ← Counter
                    delete x from class i and insert x into class Twin[i]
```

Bottleneck Analysis - Implementation - Part 3.3

ALGORITHM3()

```
create an empty list Involved
for all y ∈ C and all x ∈ Inv[y, a]
do i ← Class[x]
   if Size[i] = 0
      then Size[i] ← 1
           insert i to Involved
      else Size[i] ← Size[i] + 1

for all y ∈ C and all x ∈ Inv[y, a]
do i ← Class[x]
   if Size[i] < Card[i]
      then if Twin[i] = 0
              then Counter ← Counter + 1
                   Twin[i] ← Counter
                   delete x from class i and insert x into class Twin[i]

for all j ∈ Involved
do Size[j] ← 0
   Twin[j] ← 0
```

Conclusion

The global execution time of **Part 3** is $O(MNlog_2N)$.

So we have:

Theorem

Let A be an alphabet of M letters and let $\mathcal{A}$ be a DFA on this alphabet with N states, then Hopcroft's algorithm using previous data structures is in time $O(MNlog_2N)$ in the worst case.

Remark: This bound can also be proved to be tight.