Incremental Construction of Minimal Finite State Automata

Jan Daciuk

Gdańsk University of Technology

e-mail: jandac@eti.pg.gda.pl

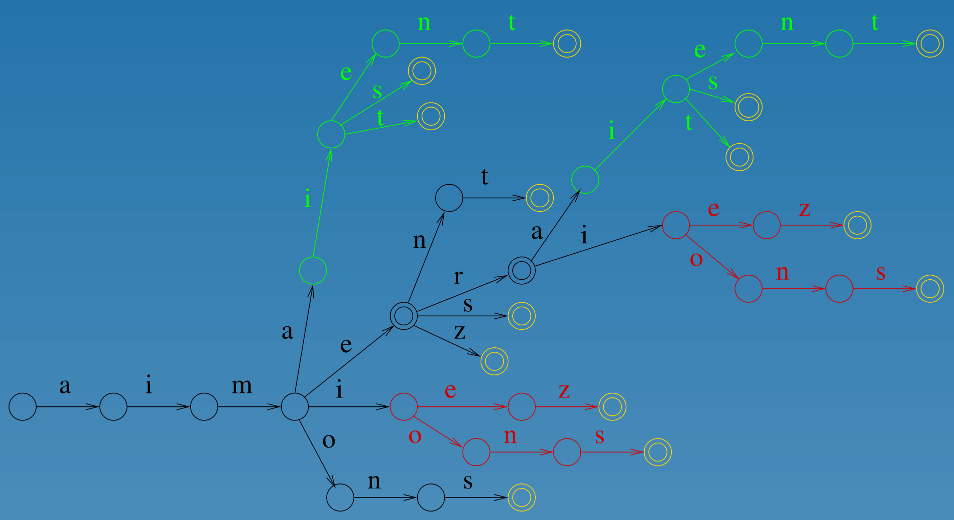
Overview

- Incrementality and semi-incrementality
- Trie construction, minimization, synchronization
- Incremental algorithm for sorted data
- Unsorted data and confluence states
- Incremental algorithm for unsorted data
- Performance

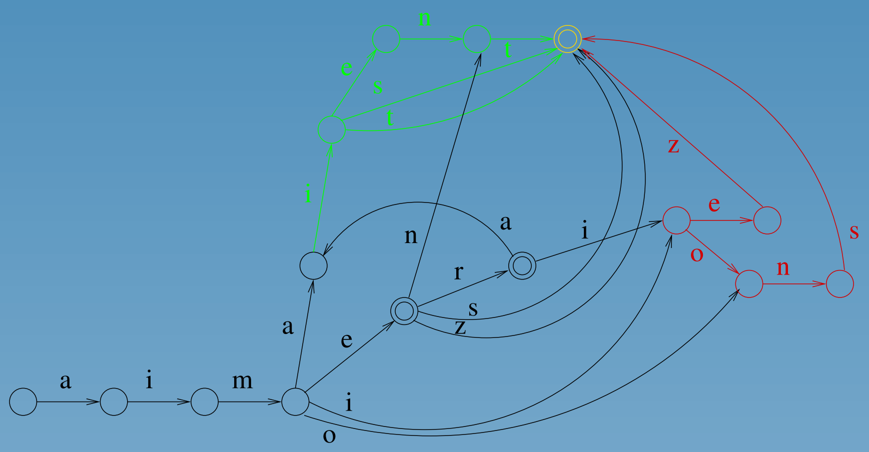
Incrementality and automata

- Final automata
  - ◇ ideal implementation of dictionaries
  - ◇ very efficient once constructed
  - ◇ traditional construction needs much memory
- Incremental and semi-incremental construction requires less memory
- Moore's law

Traditional construction

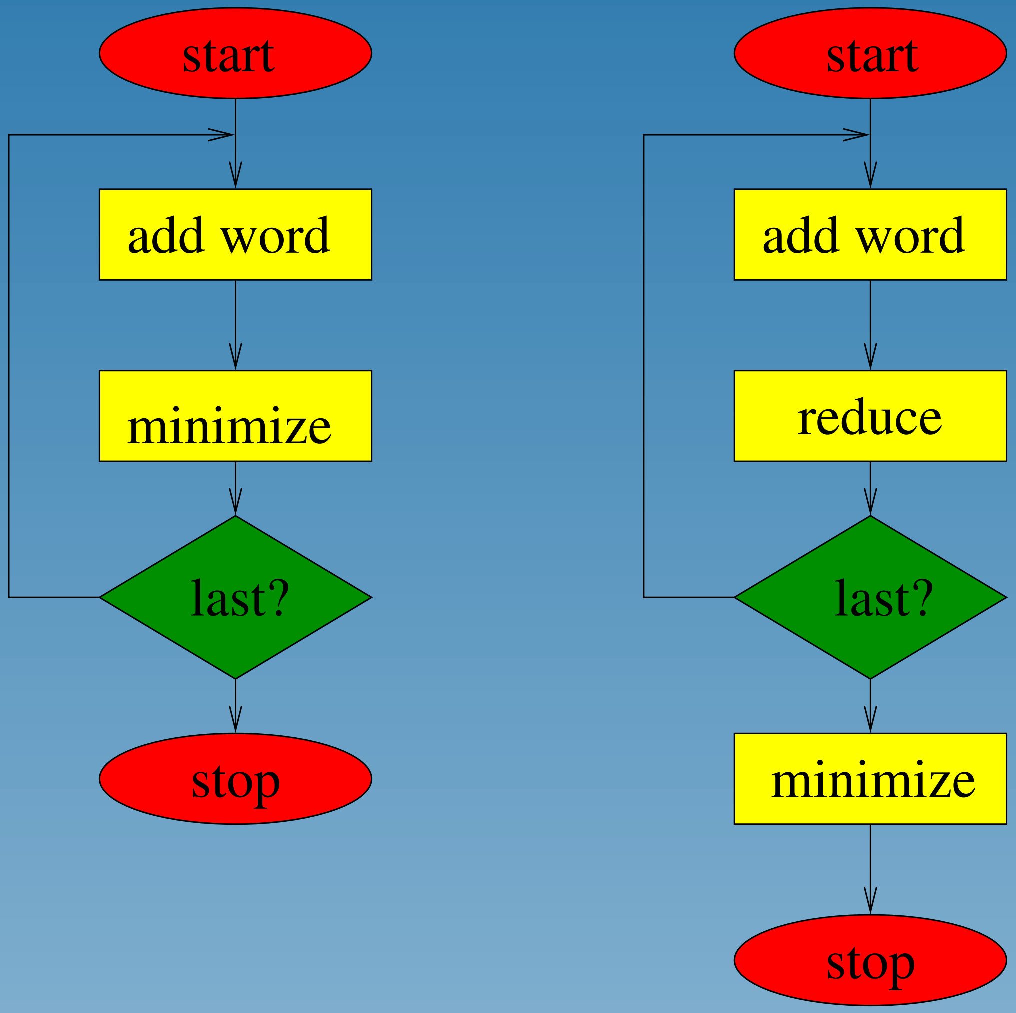


- Construct a trie



- Minimize it

Incremental vs. semi-incremental construction algorithms



Incremental and semi-incremental algorithms for acyclic automata

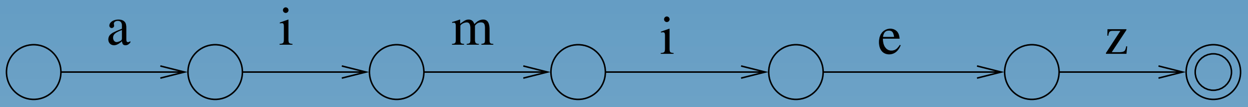
- Incremental algorithm for (lexicographically) sorted data (Daciuk, Mihov, Ciura, Deorowicz)

- Incremental algorithm for unsorted data (Aoe, Morimoto, Hase, Sgarbas, Fakotakis, Kokkinakis, Daciuk, Watson, Revuz. . .)

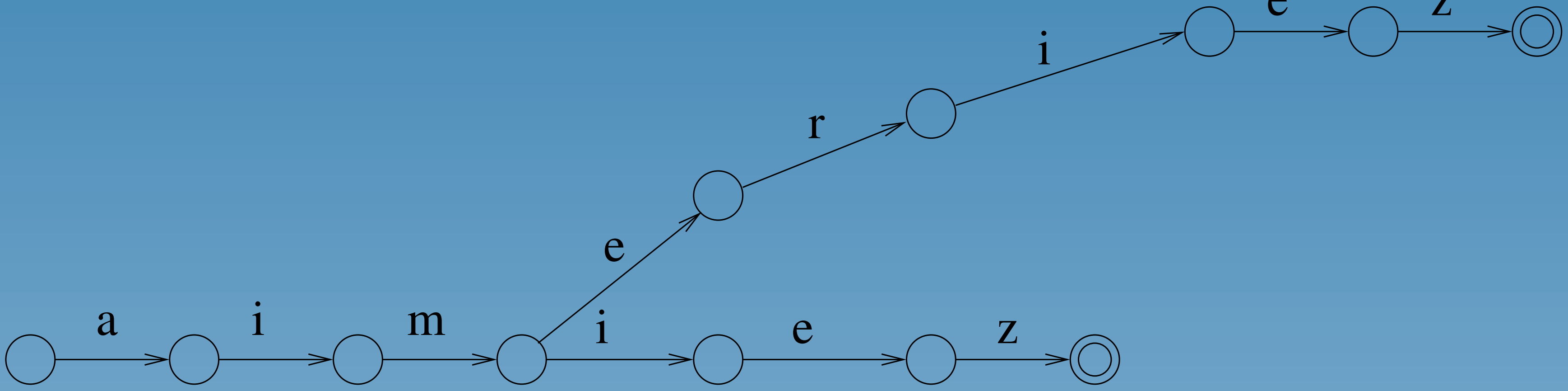
- Semi-incremental algorithm for data lexicographically sorted on reversed strings (Revuz)

- Semi-incremental algorithm for data sorted on decreasing length of strings (Watson)

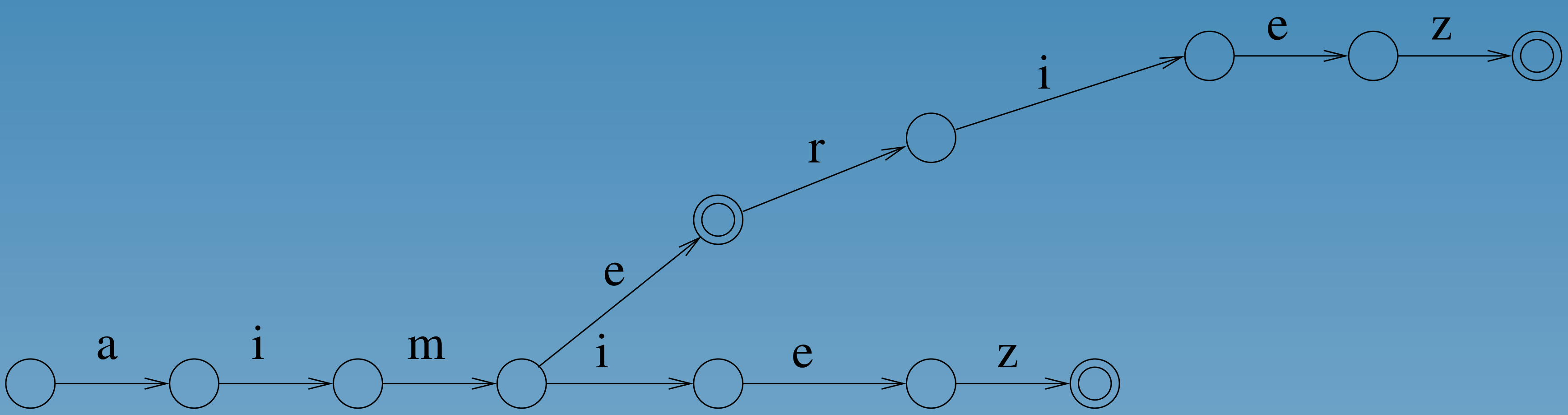
Construction of the trie



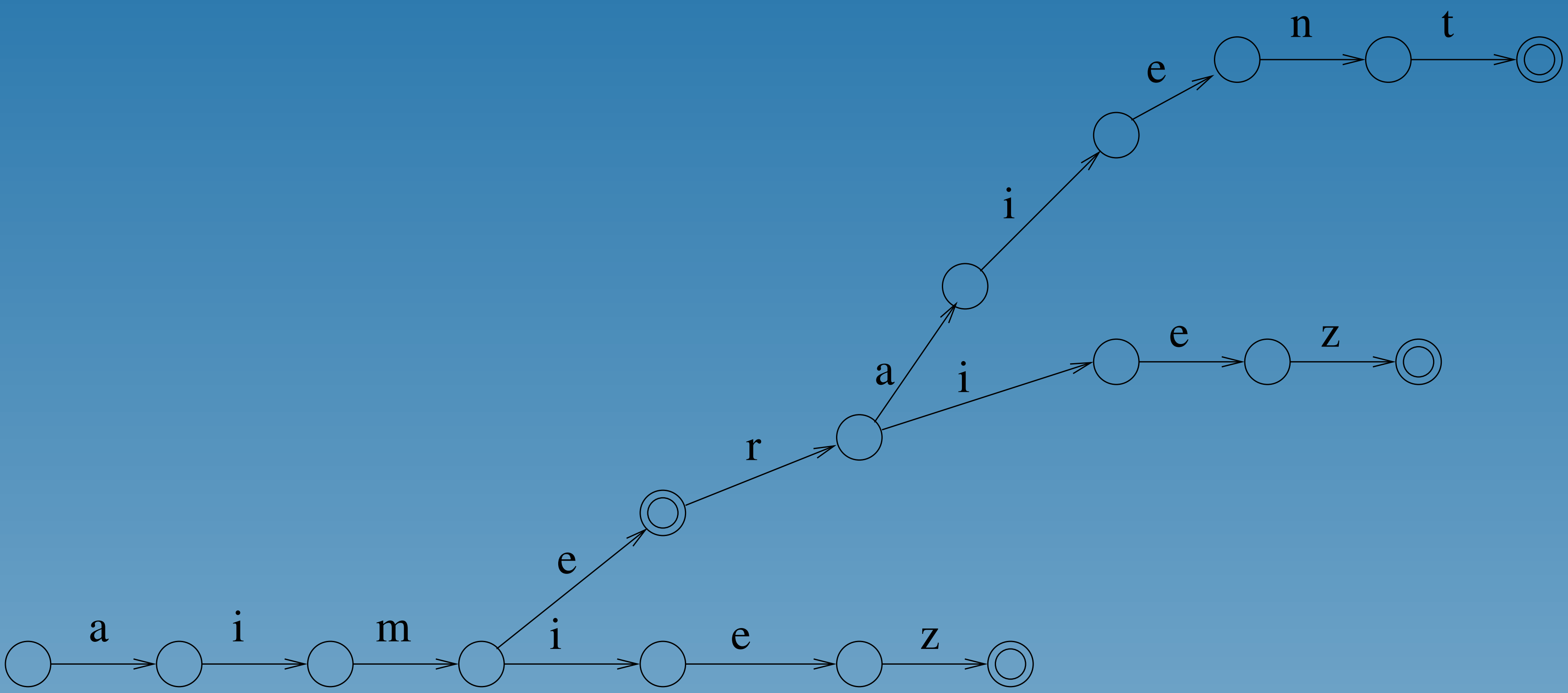
Construction of the trie



Construction of the trie



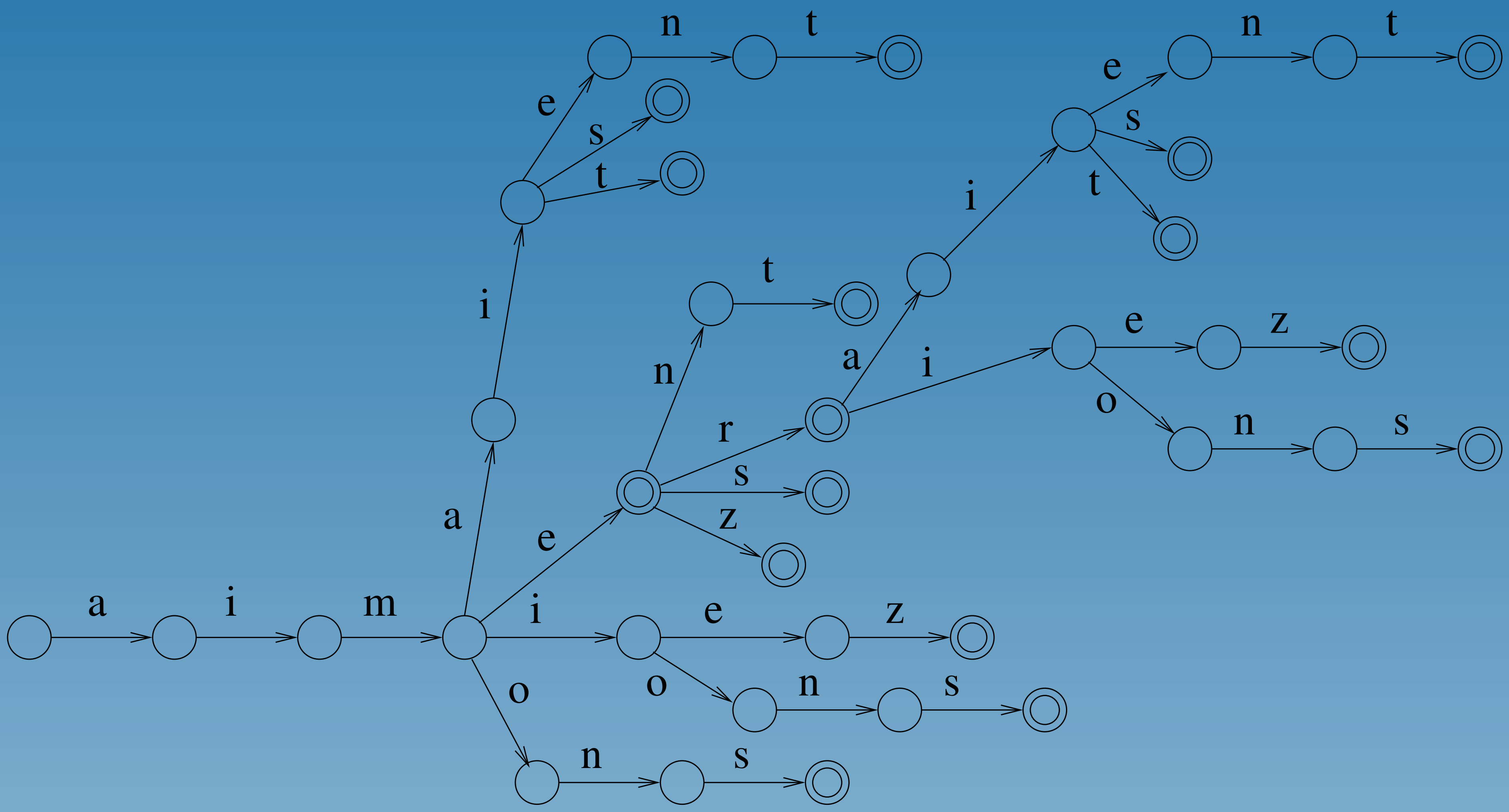
Construction of the trie



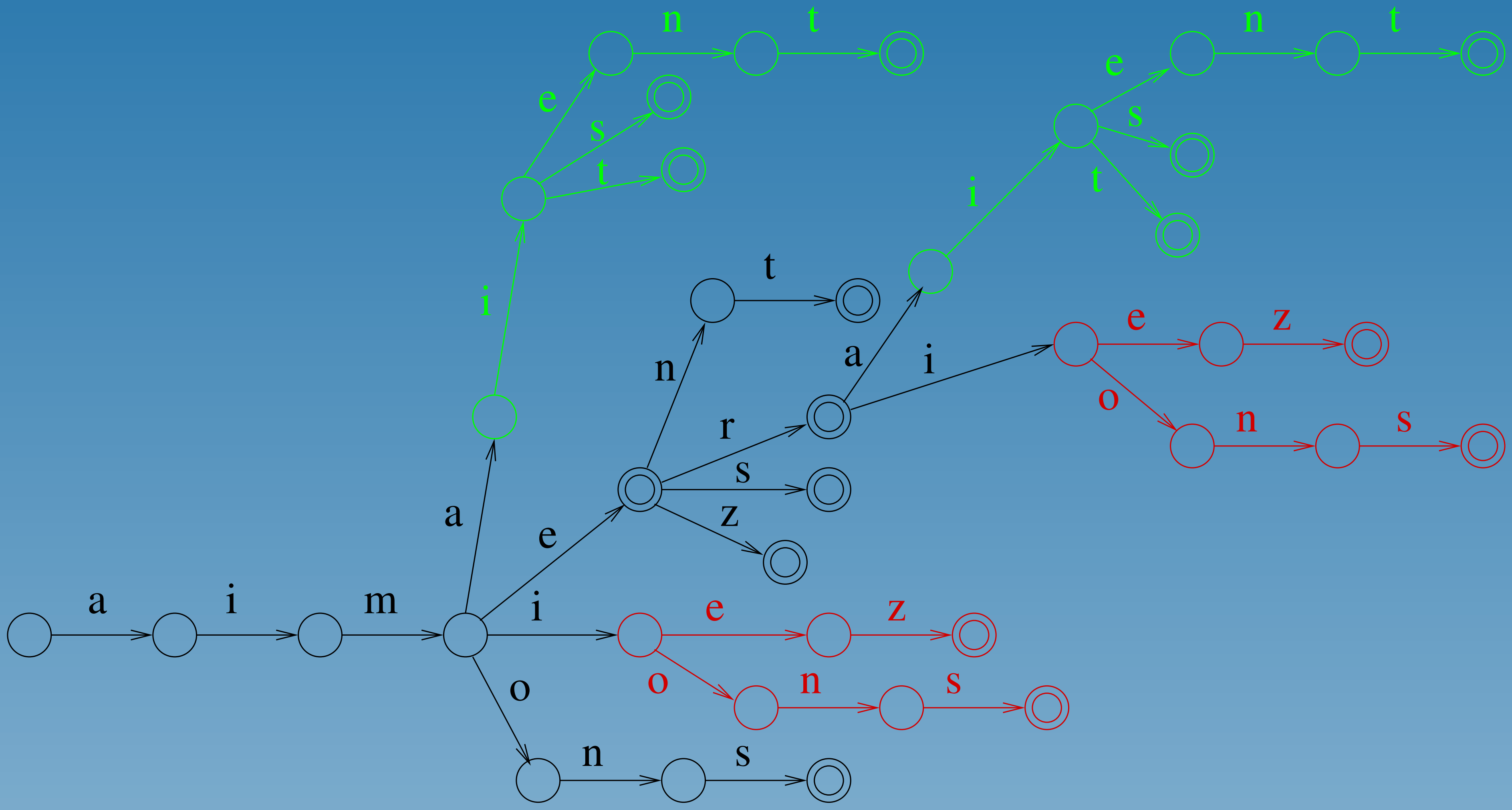
What is actually minimization?

- $M = (Q, \Sigma, \delta, q_0, F)$, $|M| = |Q|$
- M is minimal iff $\forall_{M' : \mathcal{L}(M') = \mathcal{L}(M)} |M| < |M'|$
- $\overrightarrow{\mathcal{L}}(q) = \{w : \delta^*(q, w) \in F\}$, $\mathcal{L}(M) = \overrightarrow{\mathcal{L}}(q_0)$
- $p \equiv q$ iff $\overrightarrow{\mathcal{L}}(p) = \overrightarrow{\mathcal{L}}(q)$.
- M is minimal iff $\forall_{p,q \in Q}\, p \equiv q \Leftrightarrow p = q$
- $\overrightarrow{\mathcal{L}}(q) = \cup_{a:\delta(q,a) \neq \perp}\; a\overrightarrow{\mathcal{L}}(\delta(q, a)) \cup \begin{cases} \emptyset & q \notin F \\ \epsilon & q \in F \end{cases}$

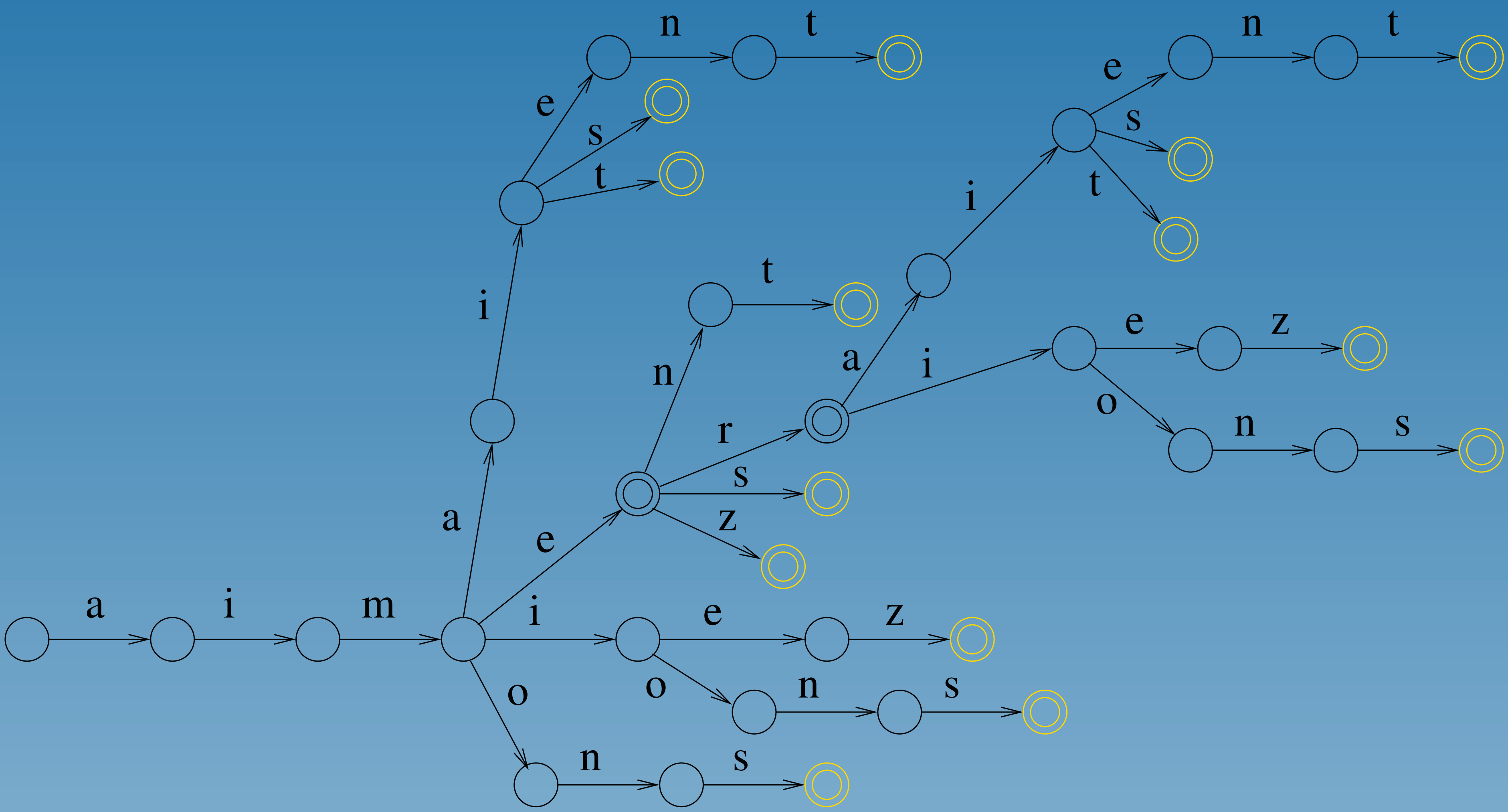
What is minimization of a trie?



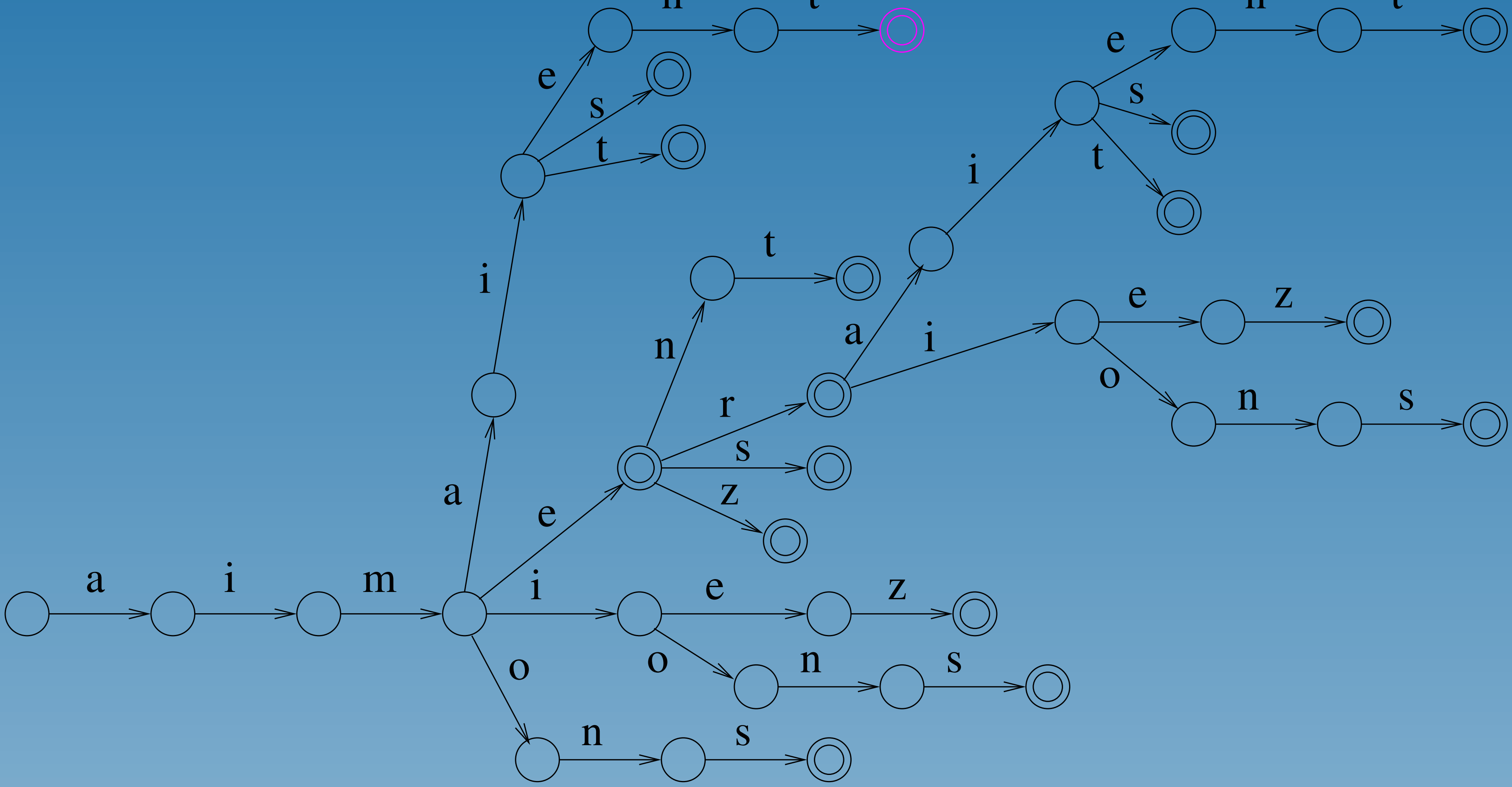
What is minimization of a trie?



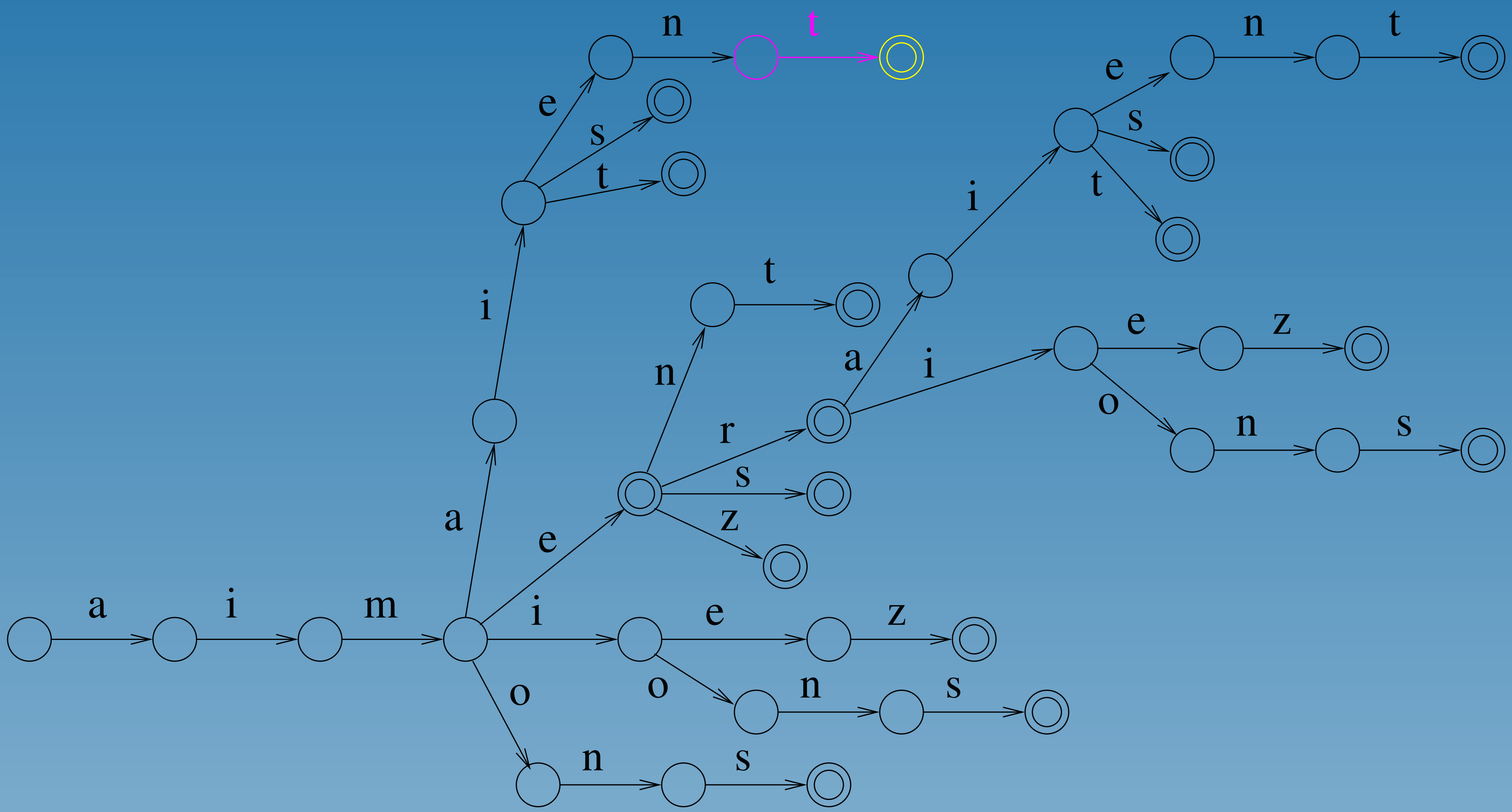
What is minimization of a trie?



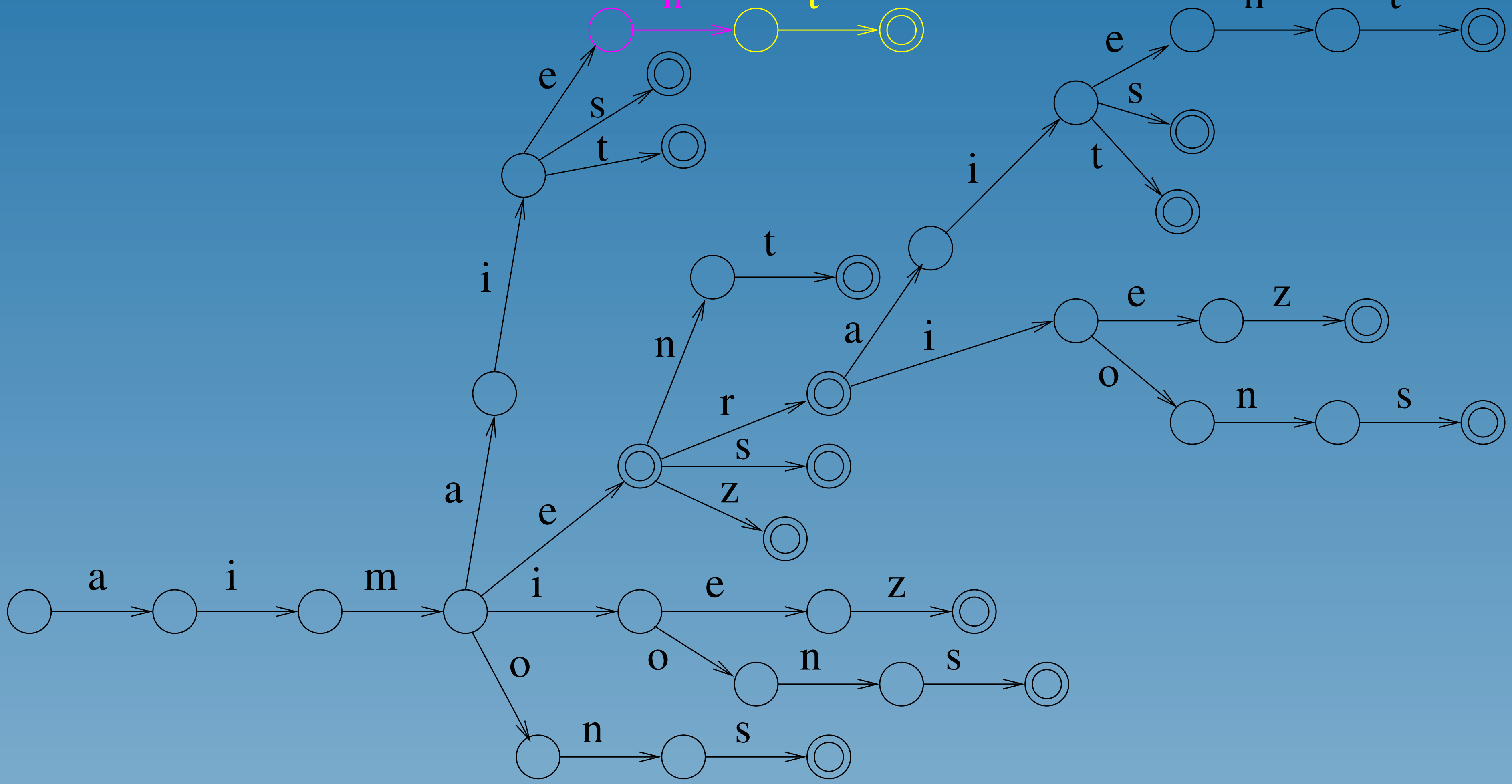
What is minimization of a trie?



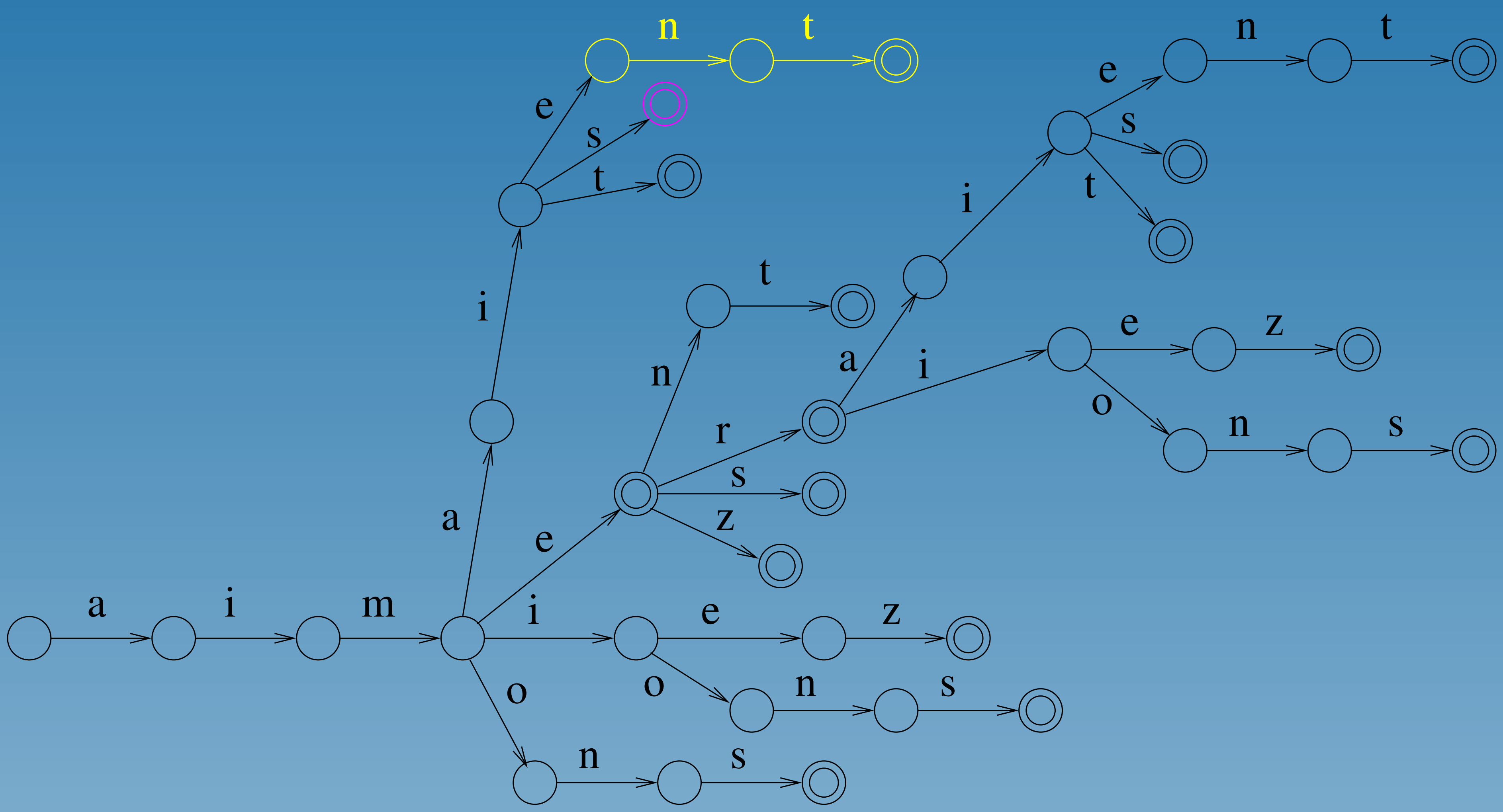
What is minimization of a trie?



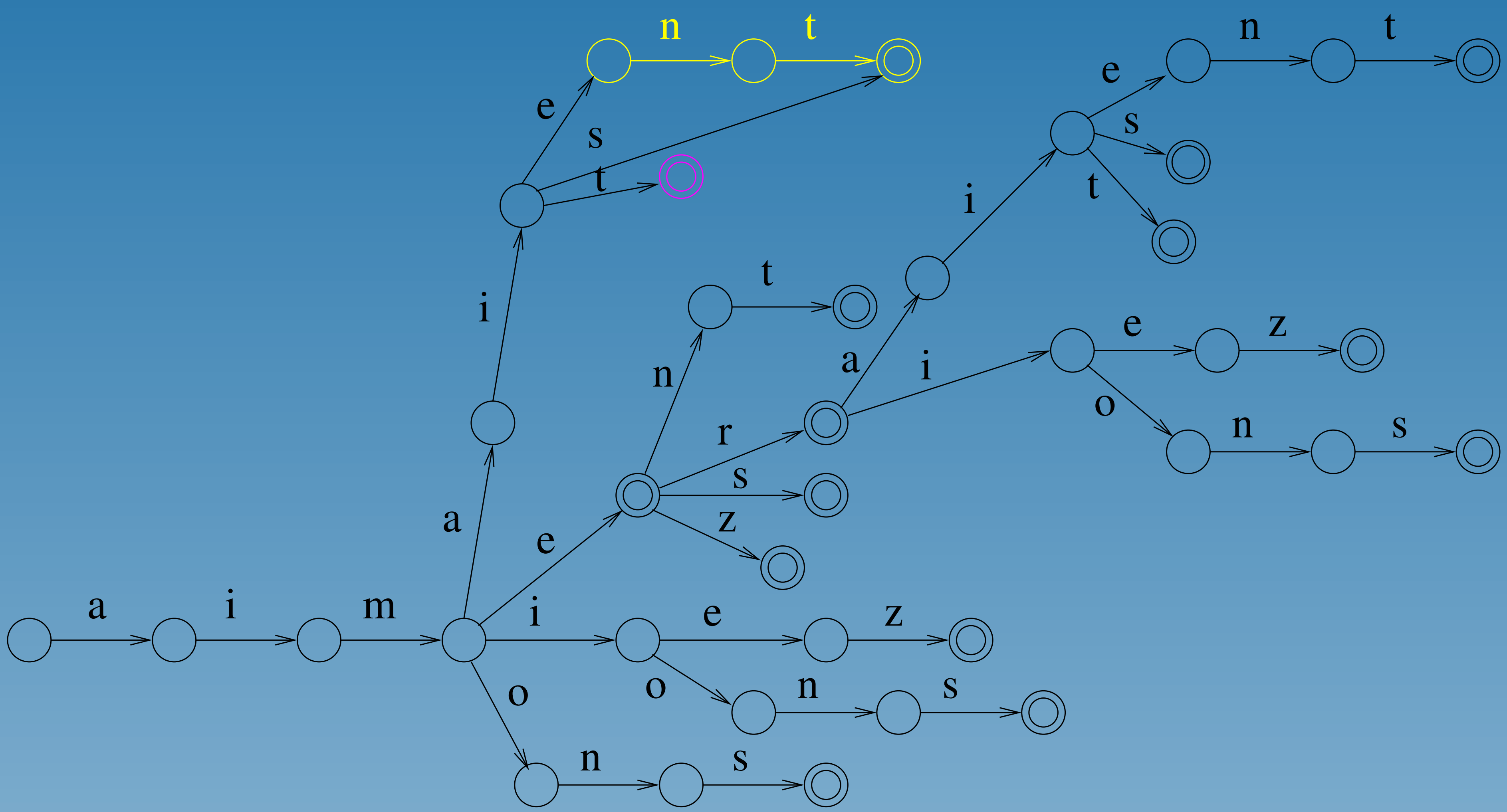
What is minimization of a trie?



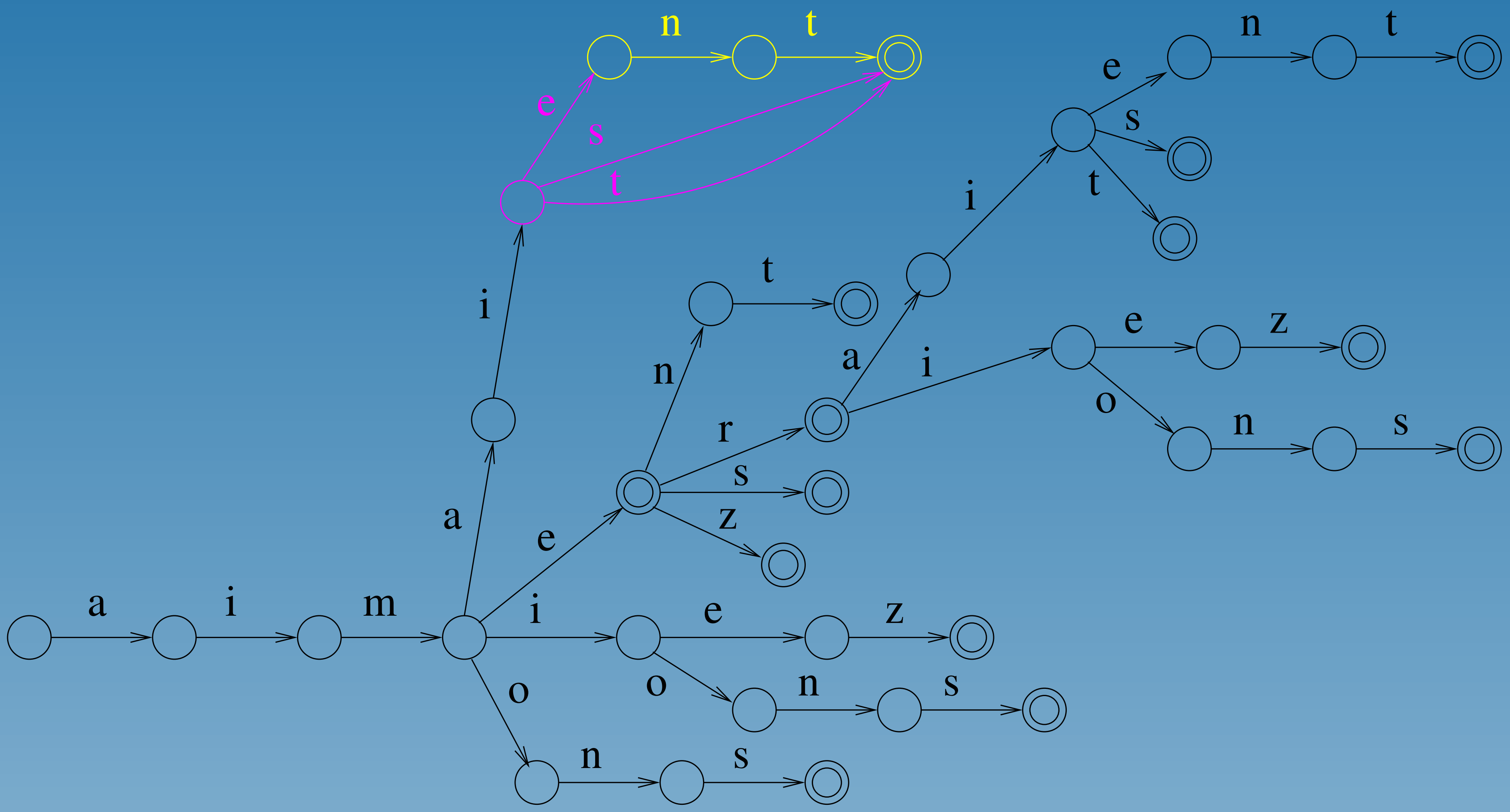
What is minimization of a trie?



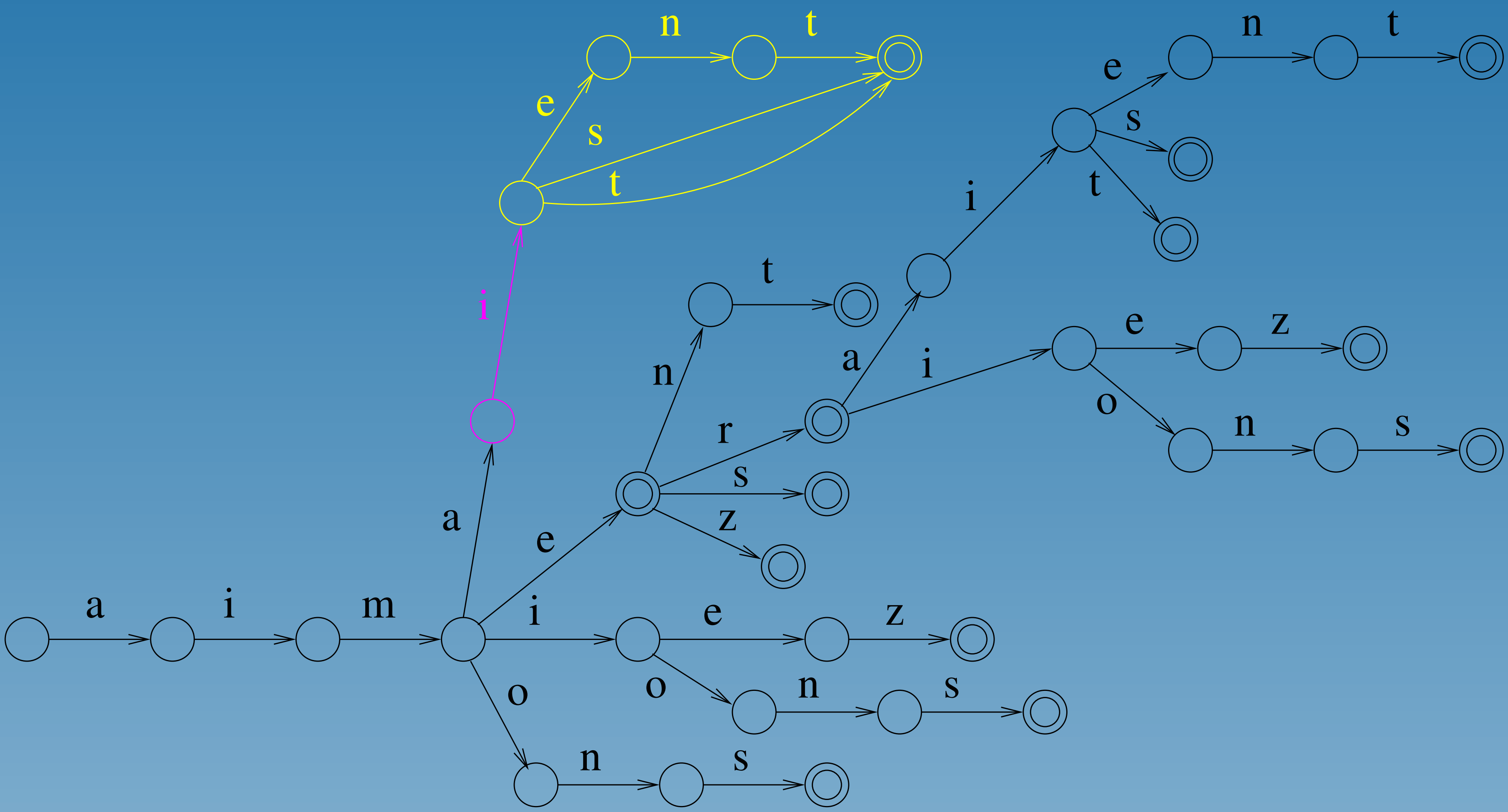
What is minimization of a trie?



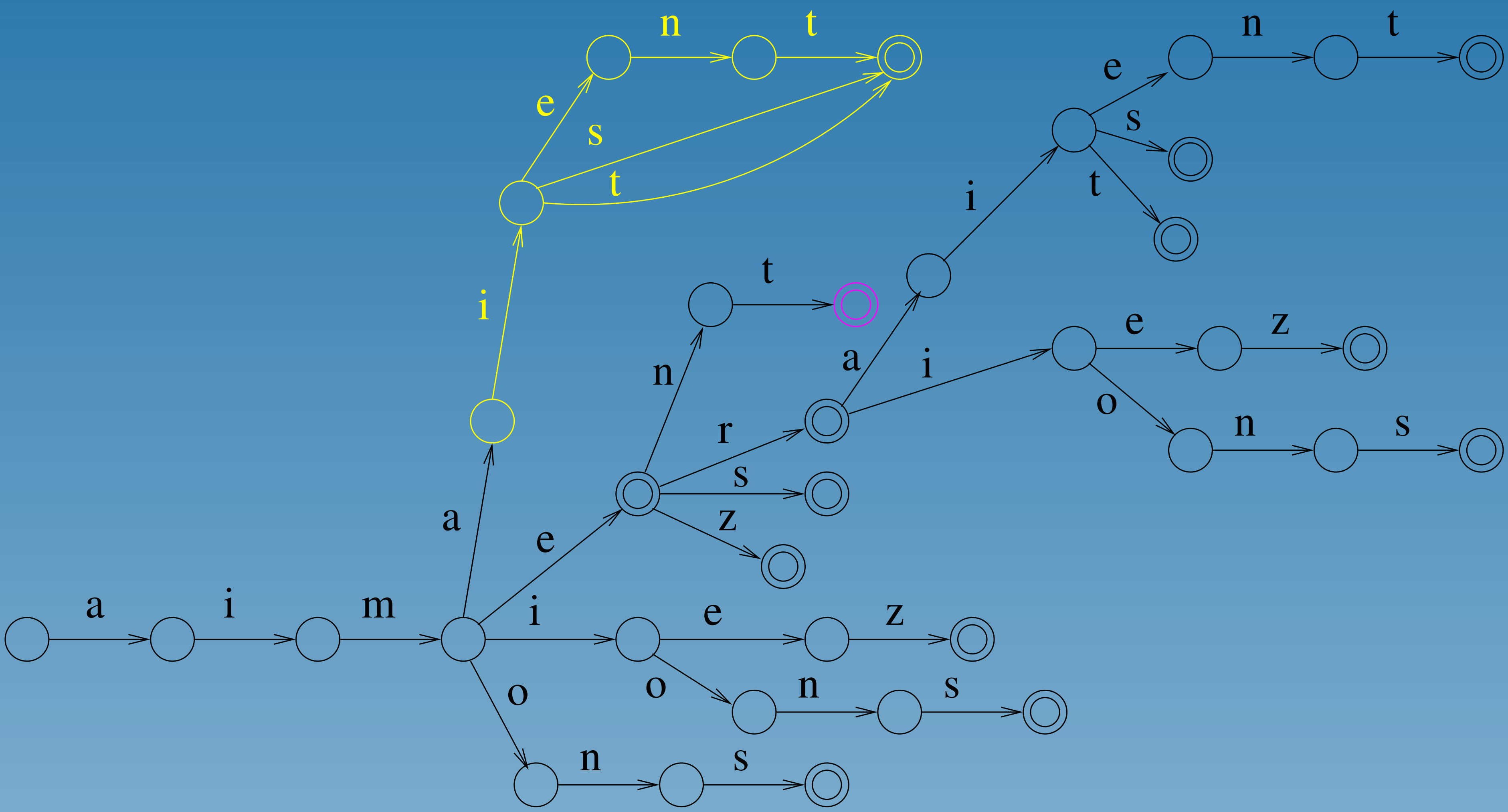
What is minimization of a trie?



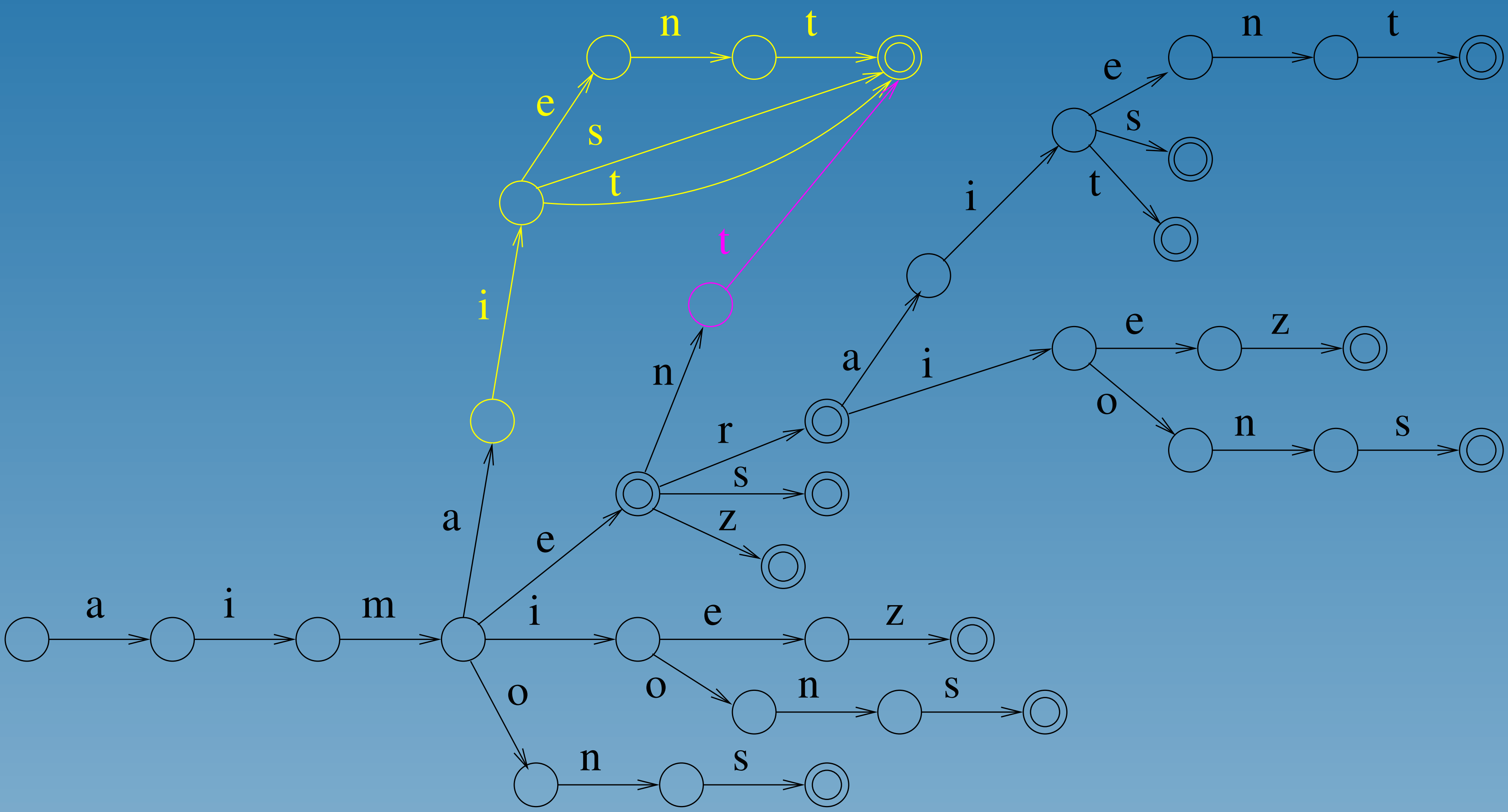
What is minimization of a trie?



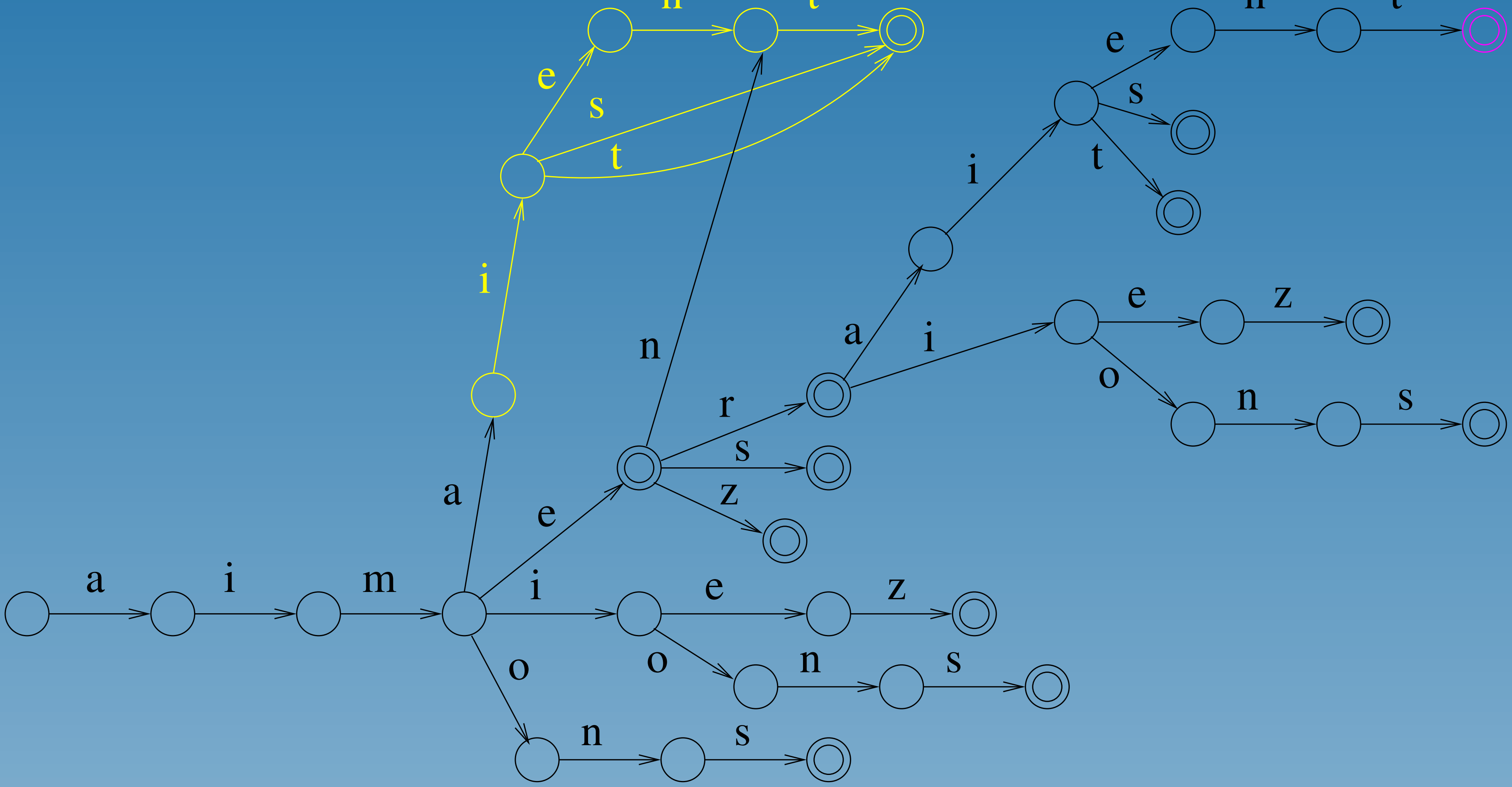
What is minimization of a trie?



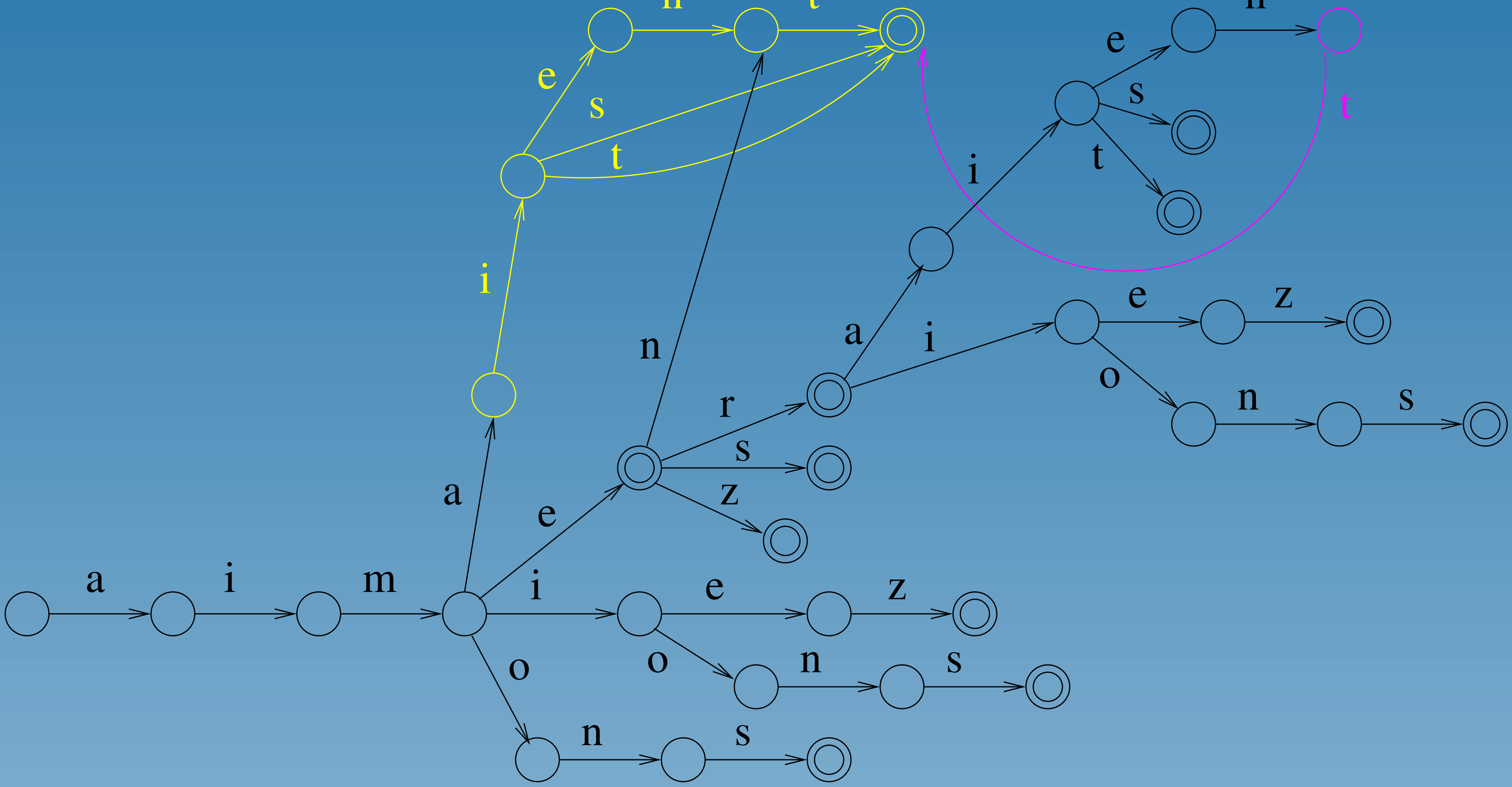
What is minimization of a trie?



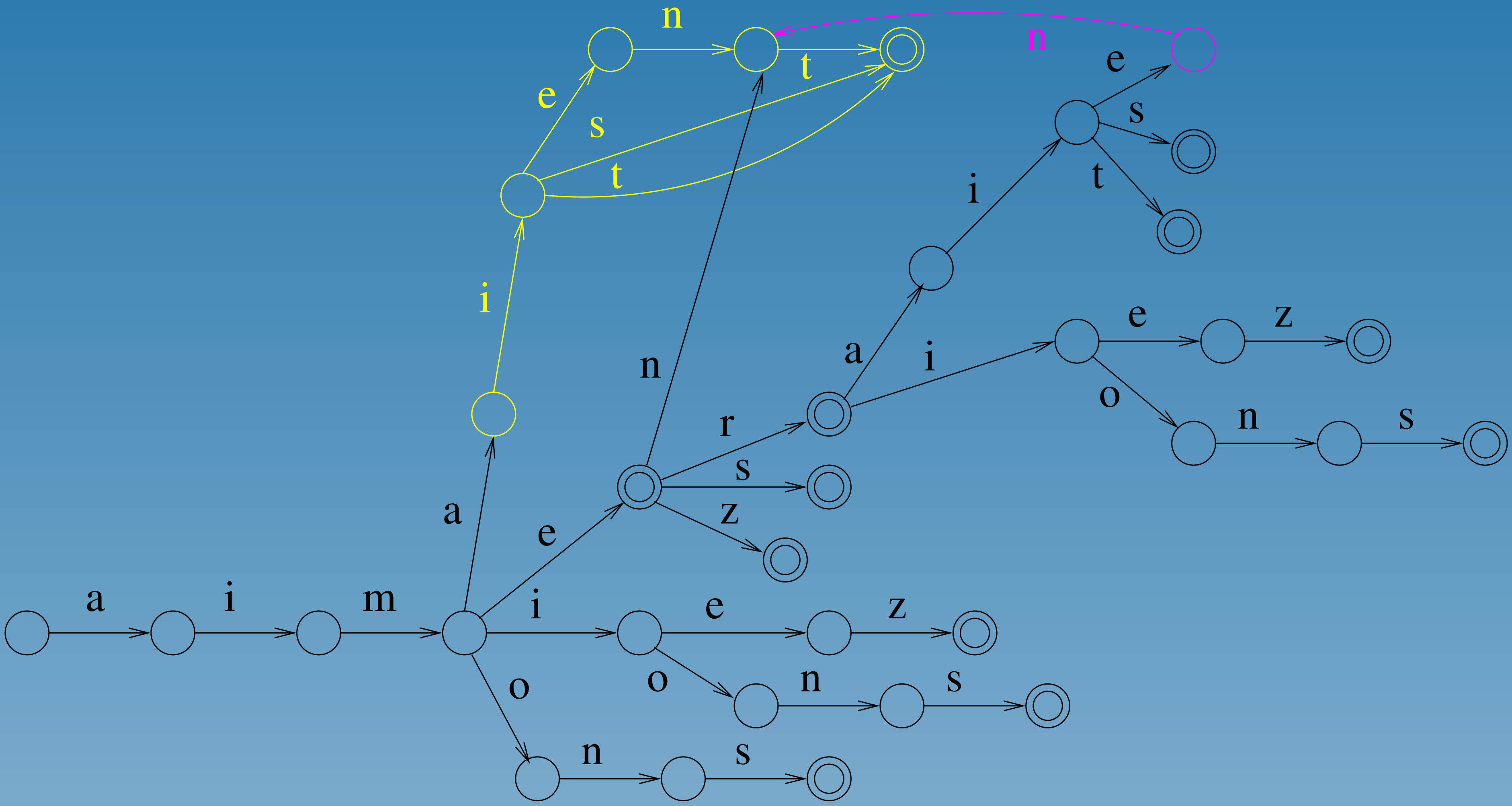
What is minimization of a trie?



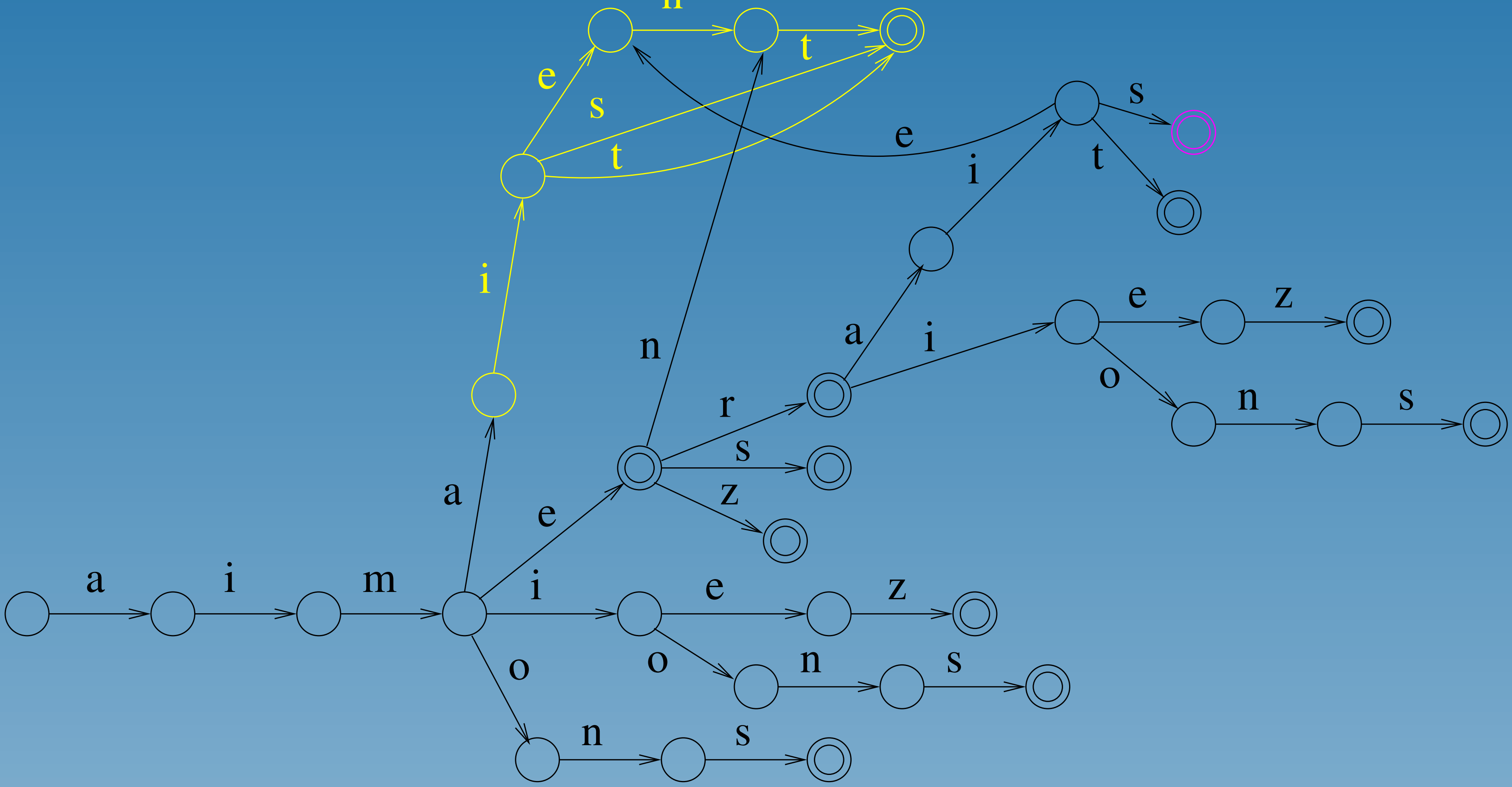
What is minimization of a trie?



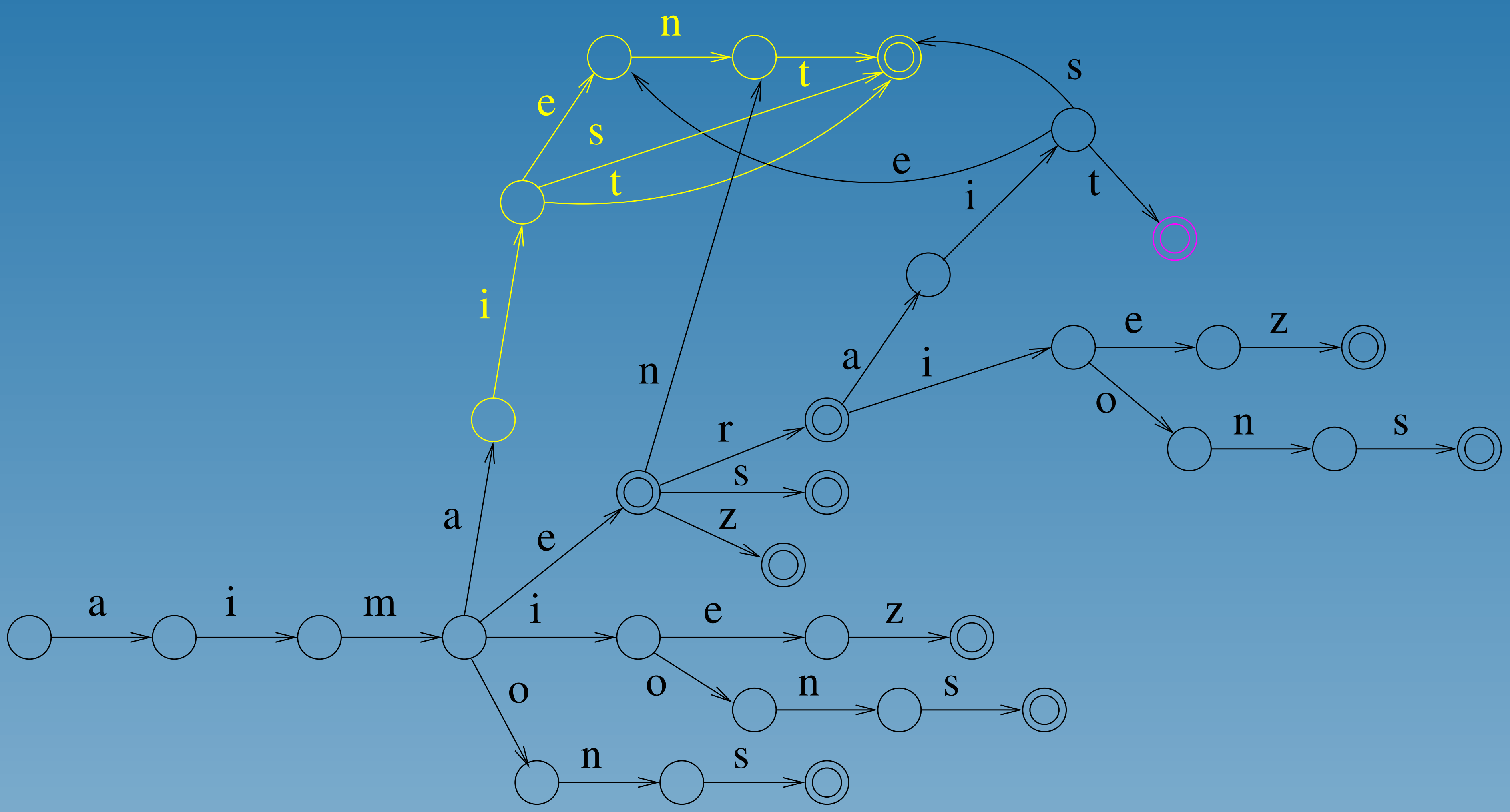
What is minimization of a trie?



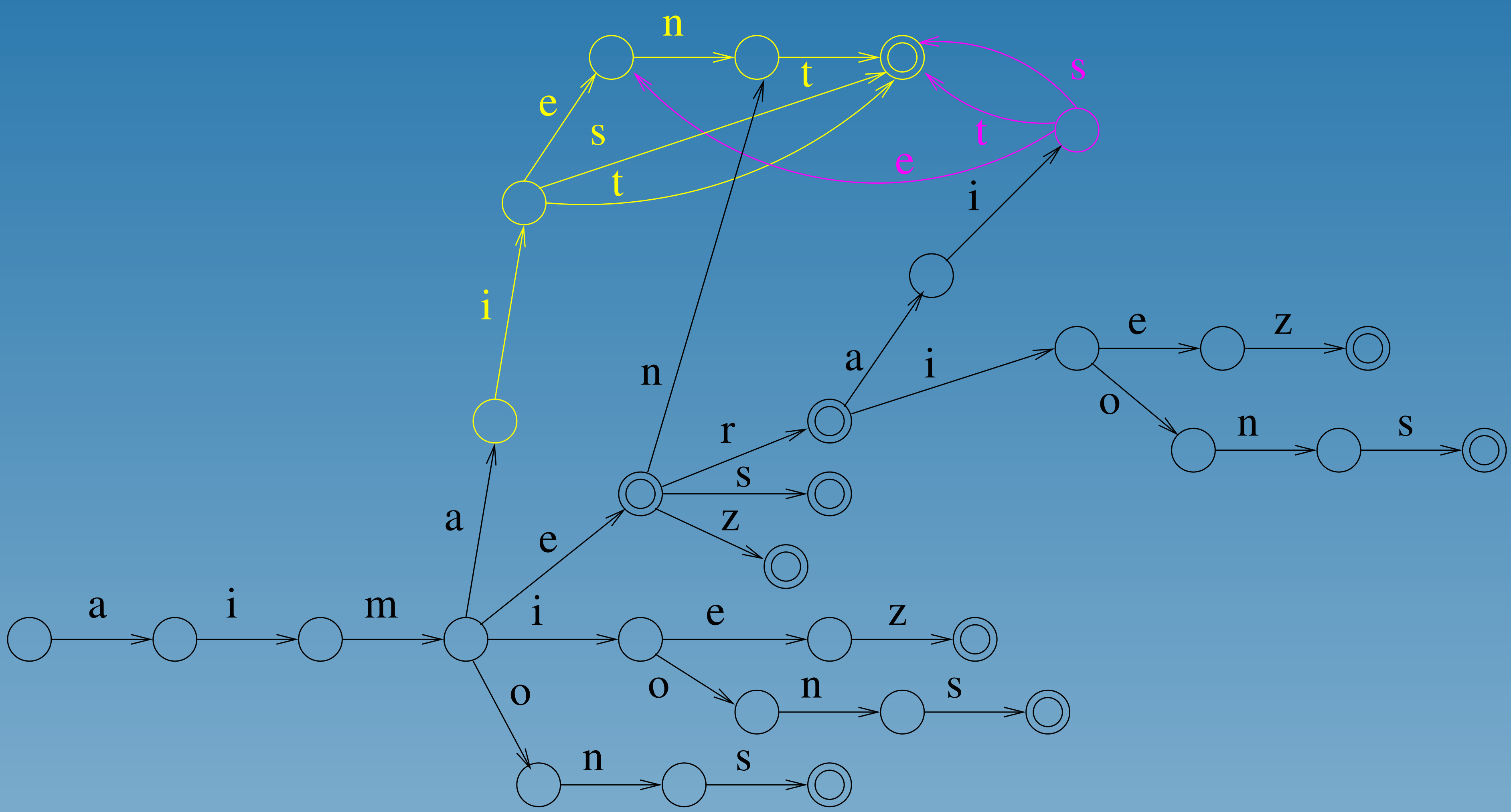
What is minimization of a trie?



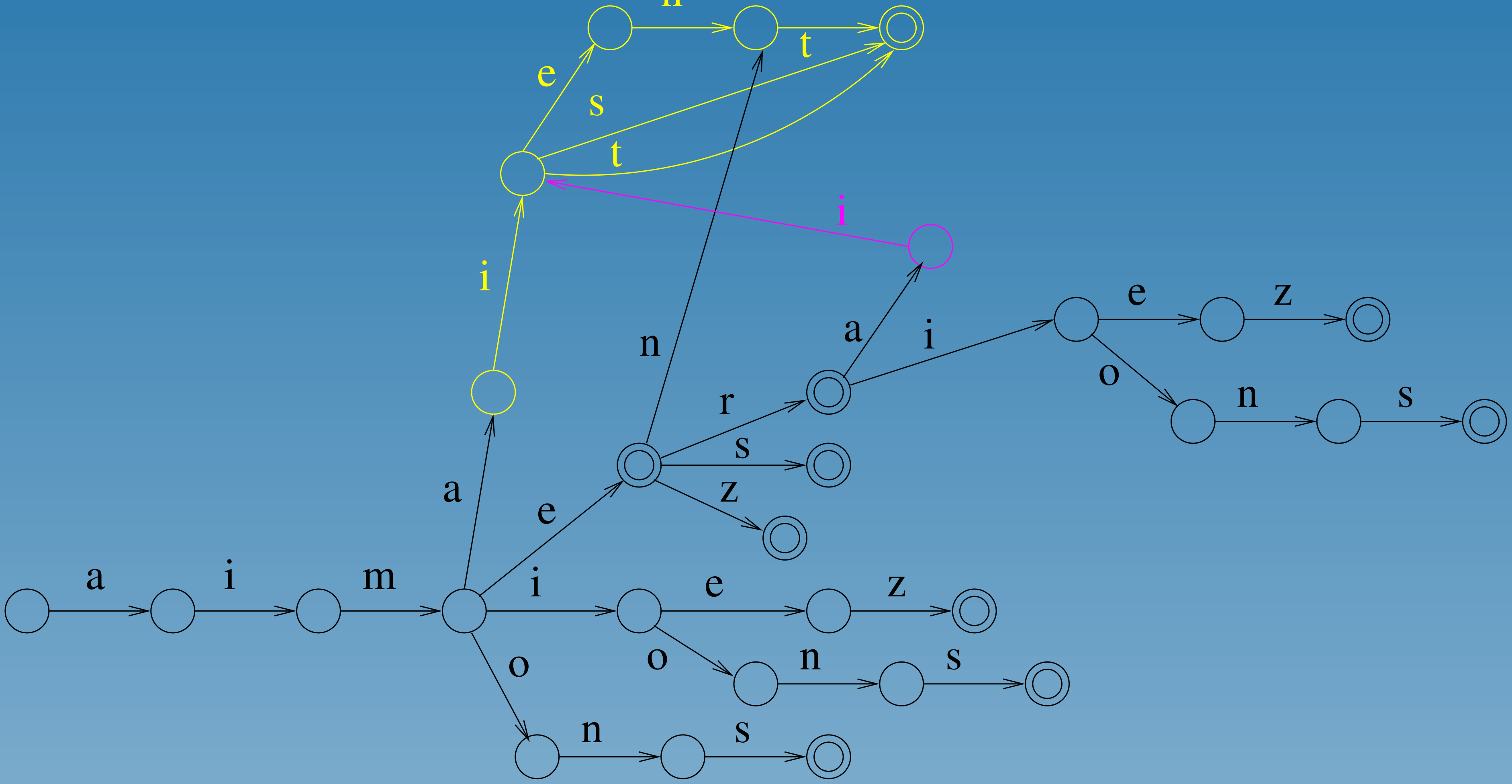
What is minimization of a trie?



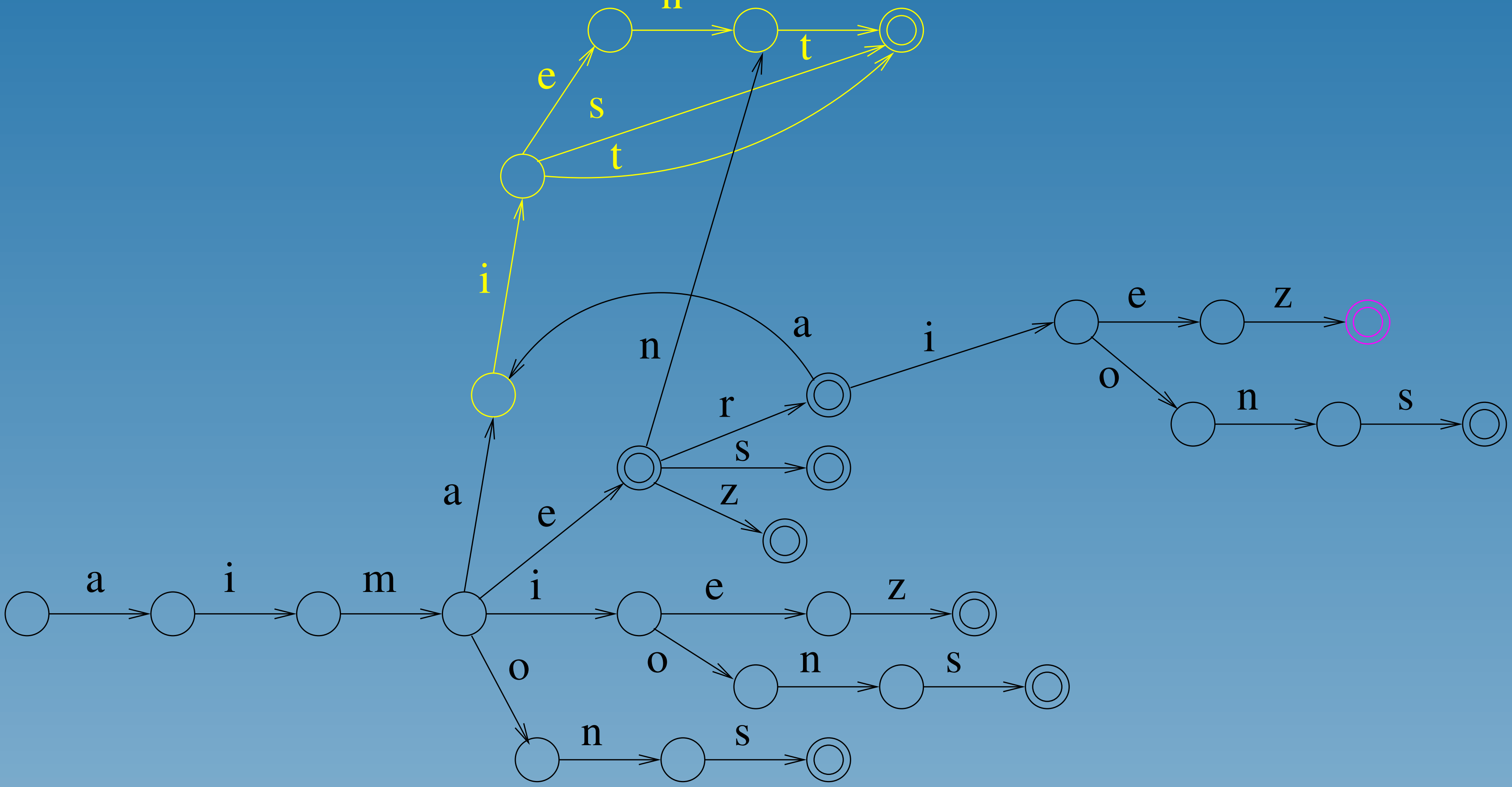
What is minimization of a trie?



What is minimization of a trie?



What is minimization of a trie?



The register

How does one check equivalence of two states?

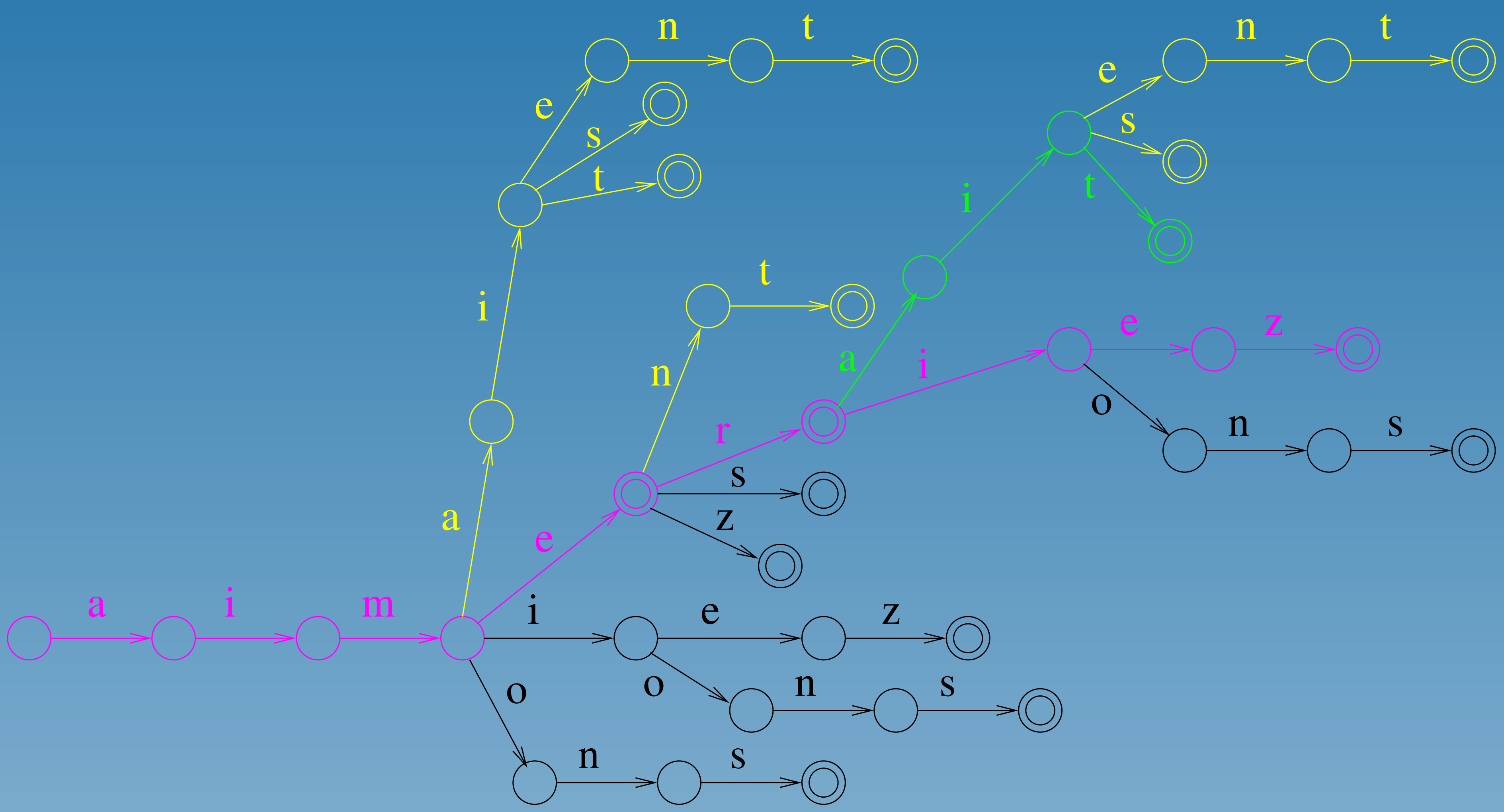
- Use the recursive definition of $\overrightarrow{\mathcal{L}}(q)$
- Visit states using postorder, so that children have unique right languages
- Keep pointers to states with unique $\overrightarrow{\mathcal{L}}(q)$ in a sparse table
- Use hash function on finality and transitions

Result: Operations on the register are $\mathcal{O}(1)$

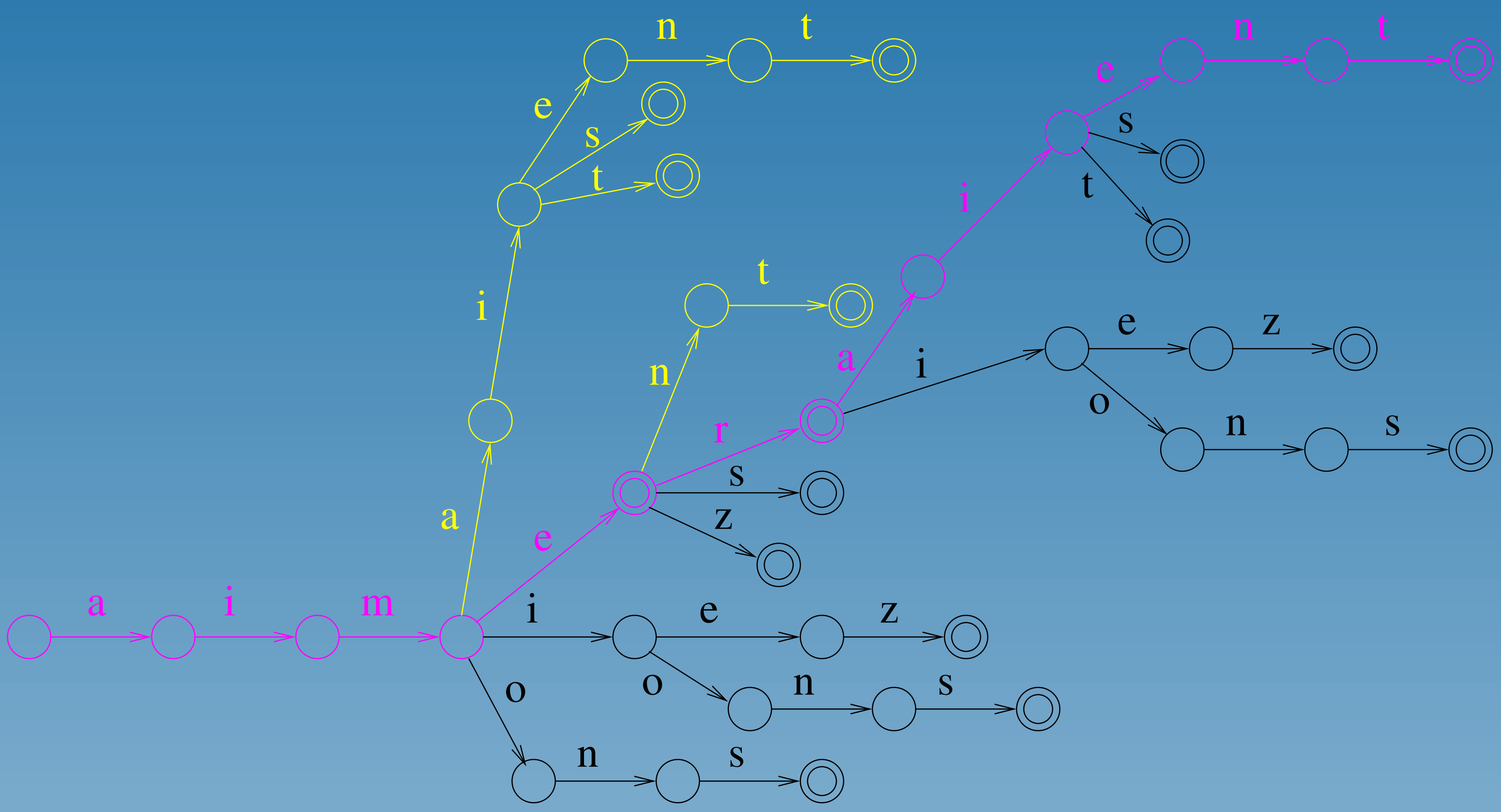
Synchronization

- Add a word to the language of the automaton and minimize the whole automaton again
  - ◇ does not pose any restrictions on input data
  - ◇ the same states have to processed over and over again – slow
- Add a word to the language of the automaton and minimize the part that will not change in the future
  - ◇ requires data to be sorted in some way
  - ◇ faster due to procesing of states only when once

Synchronization – sorted data



Synchronization – sorted data



Incremental construction from sorted data

```
function sorted_construction;
  $w' \leftarrow \epsilon$;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
    while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
      $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    if $i \leq |w'|$ then repl_or_reg$(\delta(s, w_i), w'_{i+1 \ldots |w'|})$; end if;
    while $i \leq |w|$ do
      $\delta(s, w_i) \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
  end while;
  repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
  if $v \neq \epsilon$ then
    $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2 \ldots |v|})$;
  end if;
  if $\exists_{r \in R}\, r \equiv q$ then
    delete $q$; return $r$;
  else
    $R \leftarrow R \cup \{q\}$; return $q$;
  end if;
end function;
```

Incremental construction from sorted data

```
function sorted_construction;
   $w' \leftarrow \epsilon$;
   while input not empty do
      $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
      while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
         $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
      end while;
      if $i \leq |w'|$ then repl_or_reg$(\delta(s, w_i), w'_{i+1\ldots|w'|})$; end if;
      while $i \leq |w|$ do
         $\delta(s, w_i) \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
      end while;
      $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
   end while;
   repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
   if $v \neq \epsilon$ then
      $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2\ldots|v|})$;
   end if;
   if $\exists_{r \in R}\, r \equiv q$ then
      delete $q$; return $r$;
   else
      $R \leftarrow R \cup \{q\}$; return $q$;
   end if;
end function;
```

Incremental construction from sorted data

```
function sorted_construction;
  $w' \leftarrow \epsilon$;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
    while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
      $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    if $i \leq |w'|$ then repl_or_reg$(\delta(s, w_i), w'_{i+1\ldots|w'|})$; end if;
    while $i \leq |w|$ do
      $\delta(s, w_i) \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
  end while;
  repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
  if $v \neq \epsilon$ then
    $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2\ldots|v|})$;
  end if;
  if $\exists_{r \in R}\, r \equiv q$ then
    delete $q$; return $r$;
  else
    $R \leftarrow R \cup \{q\}$; return $q$;
  end if;
end function;
```

Incremental construction from sorted data

```
function sorted_construction;
  $w' \leftarrow \epsilon$;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
    while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
      $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    if $i \leq |w'|$ then repl_or_reg$(\delta(s, w_i), w'_{i+1 \ldots |w'|})$; end if;
    while $i \leq |w|$ do
      $\delta(s, w_i) \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
  end while;
  repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
  if $v \neq \epsilon$ then
    $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2 \ldots |v|})$;
  end if;
  if $\exists_{r \in R}\, r \equiv q$ then
    delete $q$; return $r$;
  else
    $R \leftarrow R \cup \{q\}$; return $q$;
  end if;
end function;
```

Incremental construction from sorted data

```
function sorted_construction;
   $w' \leftarrow \epsilon$;
   while input not empty do
      $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
      while $i \le |w|$ and $\delta(s, w_i)! = \perp$ do
         $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
      end while;
      if $i \le |w'|$ then repl_or_reg($\delta(s, w_i), w'_{i+1\ldots|w'|}$); end if;
      while $i \le |w|$ do
         $\delta(s, w_i) \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
      end while;
      $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
   end while;
   repl_or_reg($q_0, w'$);
end function;
function repl_or_reg($q, v$);
   if $v \neq \epsilon$ then
      $\delta(q, v_1) \leftarrow$repl_or_reg($\delta(q, v_1), v_{2\ldots|v|}$);
   end if;
   if $\exists_{r \in R}\, r \equiv q$ then
      delete $q$; return $r$;
   else
      $R \leftarrow R \cup \{q\}$; return $q$;
   end if;
end function;
```

Incremental construction from sorted data

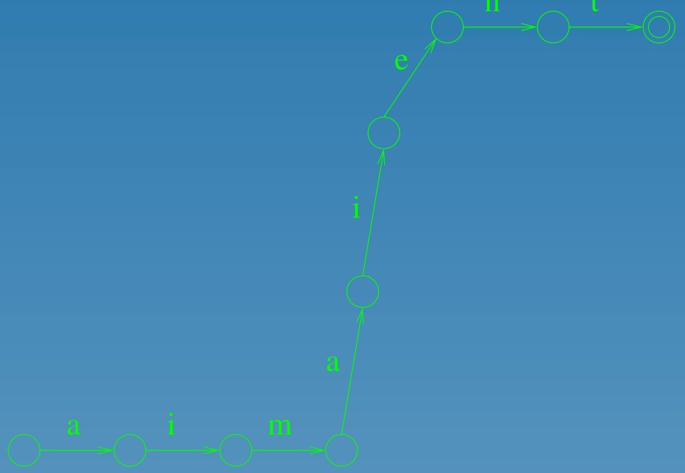
```
function sorted_construction;
  $w' \leftarrow \epsilon$;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
    while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
      $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    if $i \leq |w'|$ then repl_or_reg$(\delta(s, w_i), w'_{i+1 \ldots |w'|})$; end if;
    while $i \leq |w|$ do
      $\delta(s, w_i) \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
  end while;
  repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
  if $v \neq \epsilon$ then
    $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2 \ldots |v|})$;
  end if;
  if $\exists_{r \in R}\, r \equiv q$ then
    delete $q$; return $r$;
  else
    $R \leftarrow R \cup \{q\}$; return $q$;
  end if;
end function;
```

Incremental construction from sorted data

```
function sorted_construction;
  $w' \leftarrow \epsilon$;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
    while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
      $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    if $i \leq |w'|$ then repl_or_reg$(\delta(s, w_i), w'_{i+1 \ldots |w'|})$; end if;
    while $i \leq |w|$ do
      $\delta(s, w_i) \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
  end while;
  repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
  if $v \neq \epsilon$ then
    $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2 \ldots |v|})$;
  end if;
  if $\exists_{r \in R}\, r \equiv q$ then
    delete $q$; return $r$;
  else
    $R \leftarrow R \cup \{q\}$; return $q$;
  end if;
end function;
```

Incremental construction from sorted data – examples

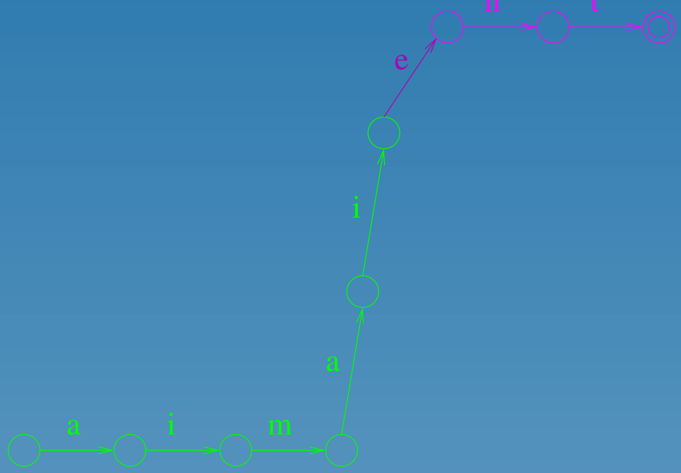
```
function sorted_construction;
    $w' \leftarrow \epsilon$;
    while input not empty do
        $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
        while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        if $i \leq |w'|$ then
            repl_or_reg$(\delta(s, w_i), w'_{i+1 \ldots |w'|})$;
        end if;
        while $i \leq |w|$ do
            $\delta(s, w_i) \leftarrow$ new state;
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
    end while;
    repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
    if $v \neq \epsilon$ then
        $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2 \ldots |v|})$;
    end if;
    if $\exists_{r \in R}\, r \equiv q$ then
        delete $q$; return $r$;
    else
        $R \leftarrow R \cup \{q\}$; return $q$;
    end if;
end function;
```



aimaient

Incremental construction from sorted data – examples

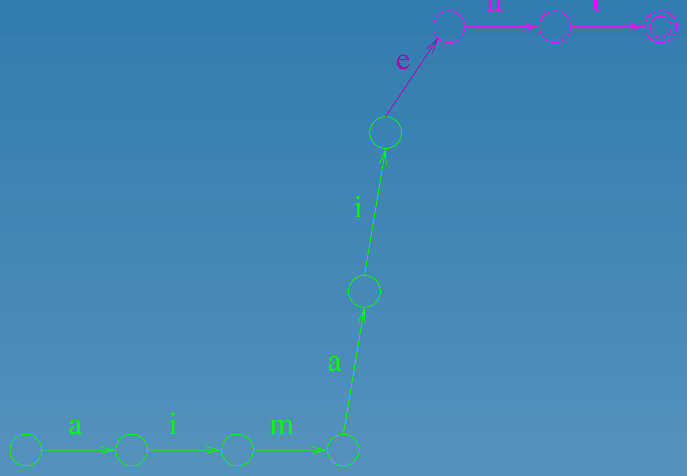
```
function sorted_construction;
  $w' \leftarrow \epsilon$;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
    while $i \leq |w|$ and $\delta(s, w_i)! = \bot$ do
      $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    if $i \leq |w'|$ then
      repl_or_reg$(\delta(s, w_i), w'_{i+1\ldots|w'|})$;
    end if;
    while $i \leq |w|$ do
      $\delta(s, w_i) \leftarrow$ new state;
      $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
  end while;
  repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
  if $v \neq \epsilon$ then
    $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2\ldots|v|})$;
  end if;
  if $\exists_{r \in R}\, r \equiv q$ then
    delete $q$; return $r$;
  else
    $R \leftarrow R \cup \{q\}$; return $q$;
  end if;
end function;
```



aimais

Incremental construction from sorted data – examples

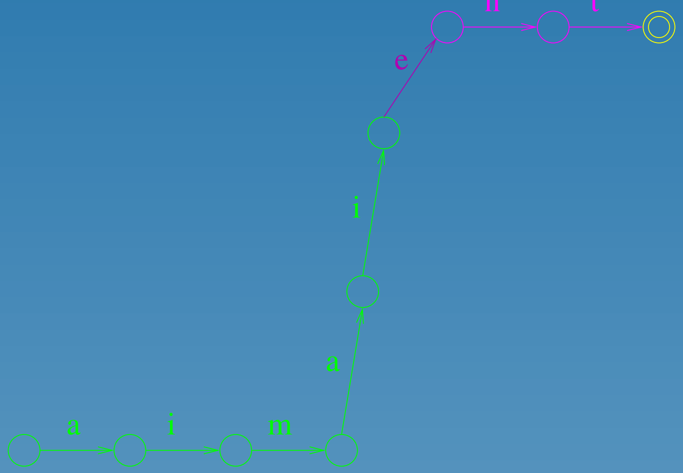
```
function sorted_construction;
    $w' \leftarrow \epsilon$;
    while input not empty do
        $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
        while $i \le |w|$ and $\delta(s, w_i)! = \perp$ do
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        if $i \le |w'|$ then
            repl_or_reg($\delta(s, w_i), w'_{i+1\ldots|w'|}$);
        end if;
        while $i \le |w|$ do
            $\delta(s, w_i) \leftarrow$ new state;
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
    end while;
    repl_or_reg($q_0, w'$);
end function;
function repl_or_reg($q, v$);
    if $v \neq \epsilon$ then
        $\delta(q, v_1) \leftarrow$ repl_or_reg($\delta(q, v_1), v_{2\ldots|v|}$);
    end if;
    if $\exists_{r \in R}\, r \equiv q$ then
        delete $q$; return $r$;
    else
        $R \leftarrow R \cup \{q\}$; return $q$;
    end if;
end function;
```



aimais

Incremental construction from sorted data – examples

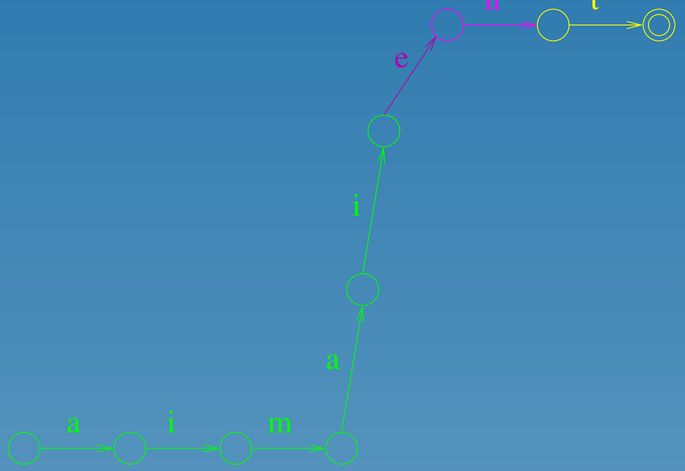
```
function sorted_construction;
    $w' \leftarrow \epsilon$;
    while input not empty do
        $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
        while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        if $i \leq |w'|$ then
            repl_or_reg$(\delta(s, w_i), w'_{i+1\ldots|w'|})$;
        end if;
        while $i \leq |w|$ do
            $\delta(s, w_i) \leftarrow$ new state;
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
    end while;
    repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
    if $v \neq \epsilon$ then
        $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2\ldots|v|})$;
    end if;
    if $\exists_{r \in R}\, r \equiv q$ then
        delete $q$; return $r$;
    else
        $R \leftarrow R \cup \{q\}$; return $q$;
    end if;
end function;
```



aimais

Incremental construction from sorted data – examples

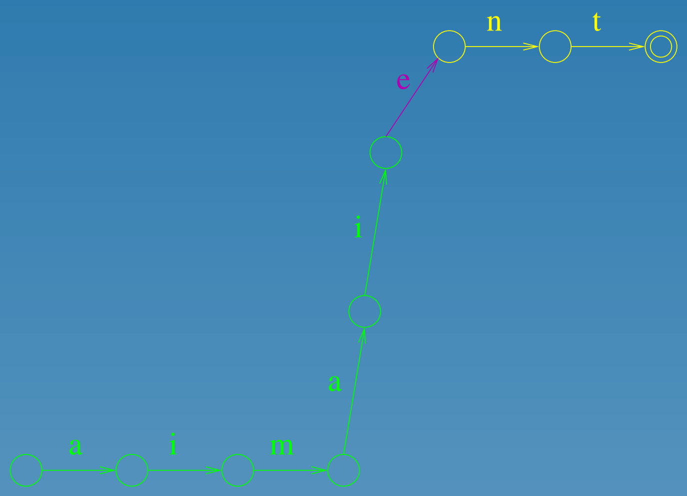
```
function sorted_construction;
   $w' \leftarrow \epsilon$;
   while input not empty do
      $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
      while $i \le |w|$ and $\delta(s, w_i)! = \bot$ do
         $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
      end while;
      if $i \le |w'|$ then
         repl_or_reg($\delta(s, w_i), w'_{i+1...|w'|}$);
      end if;
      while $i \le |w|$ do
         $\delta(s, w_i) \leftarrow$ new state;
         $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
      end while;
      $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
   end while;
   repl_or_reg($q_0, w'$);
end function;
function repl_or_reg($q, v$);
   if $v \neq \epsilon$ then
      $\delta(q, v_1) \leftarrow$ repl_or_reg($\delta(q, v_1), v_{2...|v|}$);
   end if;
   if $\exists_{r \in R}\, r \equiv q$ then
      delete $q$; return $r$;
   else
      $R \leftarrow R \cup \{q\}$; return $q$;
   end if;
end function;
```



aimais

Incremental construction from sorted data – examples

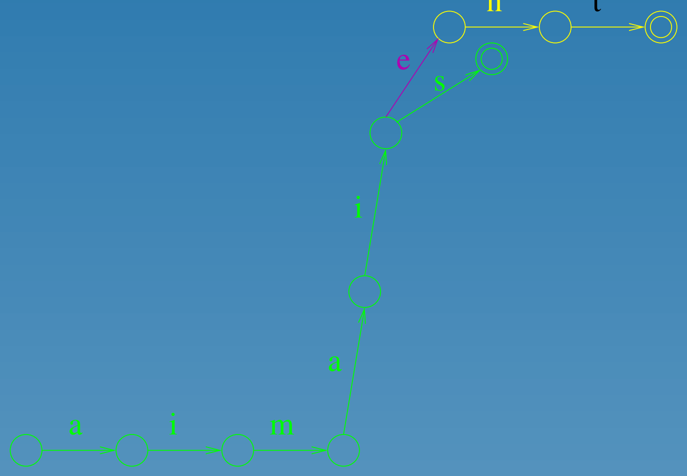
```
function sorted_construction;
   $w' \leftarrow \epsilon$;
   while input not empty do
      $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
      while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
         $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
      end while;
      if $i \leq |w'|$ then
         repl_or_reg$(\delta(s, w_i), w'_{i+1 \ldots |w'|})$;
      end if;
      while $i \leq |w|$ do
         $\delta(s, w_i) \leftarrow$ new state;
         $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
      end while;
      $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
   end while;
   repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
   if $v \neq \epsilon$ then
      $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2 \ldots |v|})$;
   end if;
   if $\exists_{r \in R}\, r \equiv q$ then
      delete $q$; return $r$;
   else
      $R \leftarrow R \cup \{q\}$; return $q$;
   end if;
end function;
```



aimais

Incremental construction from sorted data – examples

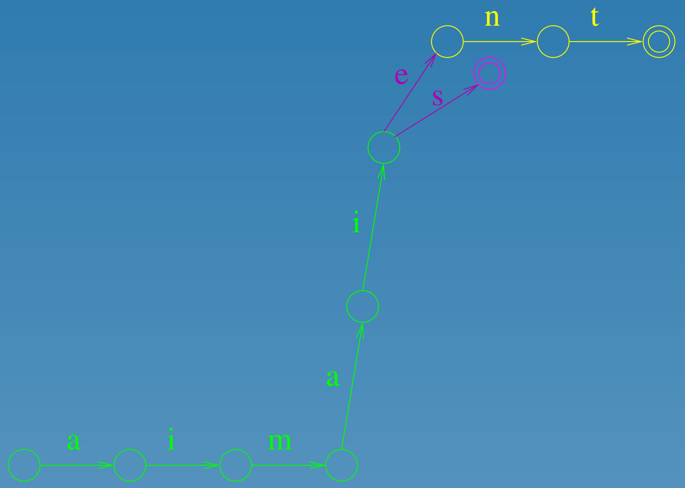
```
function sorted_construction;
    $w' \leftarrow \epsilon$;
    while input not empty do
        $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
        while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        if $i \leq |w'|$ then
            repl_or_reg$(\delta(s, w_i), w'_{i+1\ldots|w'|})$;
        end if;
        while $i \leq |w|$ do
            $\delta(s, w_i) \leftarrow$ new state;
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
    end while;
    repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
    if $v \neq \epsilon$ then
        $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2\ldots|v|})$;
    end if;
    if $\exists_{r \in R}\, r \equiv q$ then
        delete $q$; return $r$;
    else
        $R \leftarrow R \cup \{q\}$; return $q$;
    end if;
end function;
```



aimais

Incremental construction from sorted data – examples

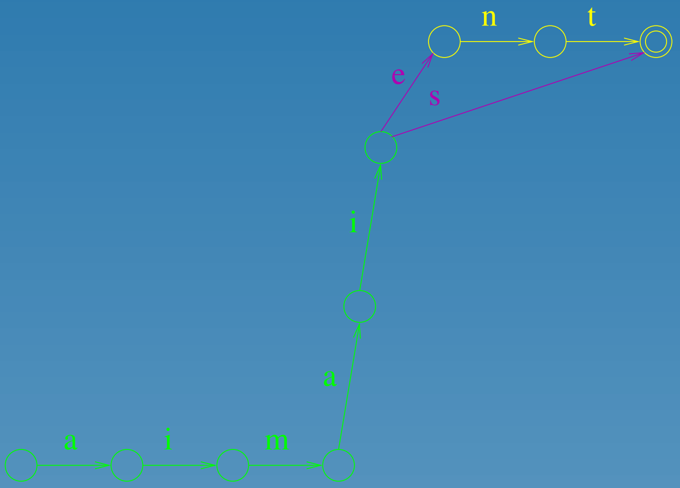
```
function sorted_construction;
   $w' \leftarrow \epsilon$;
   while input not empty do
      $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
      while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
         $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
      end while;
      if $i \leq |w'|$ then
         repl_or_reg($\delta(s, w_i), w'_{i+1\ldots|w'|}$);
      end if;
      while $i \leq |w|$ do
         $\delta(s, w_i) \leftarrow$ new state;
         $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
      end while;
      $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
   end while;
   repl_or_reg($q_0, w'$);
end function;
function repl_or_reg($q, v$);
   if $v \neq \epsilon$ then
      $\delta(q, v_1) \leftarrow$ repl_or_reg($\delta(q, v_1), v_{2\ldots|v|}$);
   end if;
   if $\exists_{r \in R}\, r \equiv q$ then
      delete $q$; return $r$;
   else
      $R \leftarrow R \cup \{q\}$; return $q$;
   end if;
end function;
```



aimait

Incremental construction from sorted data – examples

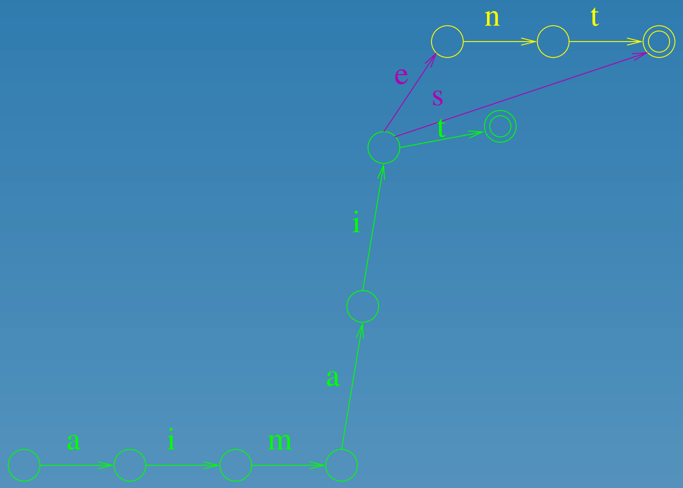
```
function sorted_construction;
    $w' \leftarrow \epsilon$;
    while input not empty do
        $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
        while $i \leq |w|$ and $\delta(s, w_i) != \perp$ do
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        if $i \leq |w'|$ then
            repl_or_reg$(\delta(s, w_i), w'_{i+1\ldots|w'|})$;
        end if;
        while $i \leq |w|$ do
            $\delta(s, w_i) \leftarrow$ new state;
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
    end while;
    repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
    if $v \neq \epsilon$ then
        $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2\ldots|v|})$;
    end if;
    if $\exists_{r \in R}\, r \equiv q$ then
        delete $q$; return $r$;
    else
        $R \leftarrow R \cup \{q\}$; return $q$;
    end if;
end function;
```



aimait

Incremental construction from sorted data – examples

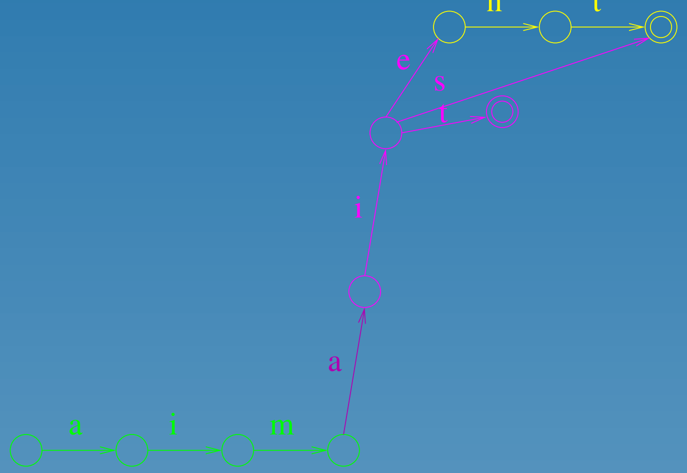
```
function sorted_construction;
   $w' \leftarrow \epsilon$;
   while input not empty do
      $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
      while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
         $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
      end while;
      if $i \leq |w'|$ then
         repl_or_reg$(\delta(s, w_i), w'_{i+1\ldots|w'|})$;
      end if;
      while $i \leq |w|$ do
         $\delta(s, w_i) \leftarrow$ new state;
         $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
      end while;
      $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
   end while;
   repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
   if $v \neq \epsilon$ then
      $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2\ldots|v|})$;
   end if;
   if $\exists_{r \in R}\, r \equiv q$ then
      delete $q$; return $r$;
   else
      $R \leftarrow R \cup \{q\}$; return $q$;
   end if;
end function;
```



aimait

Incremental construction from sorted data – examples

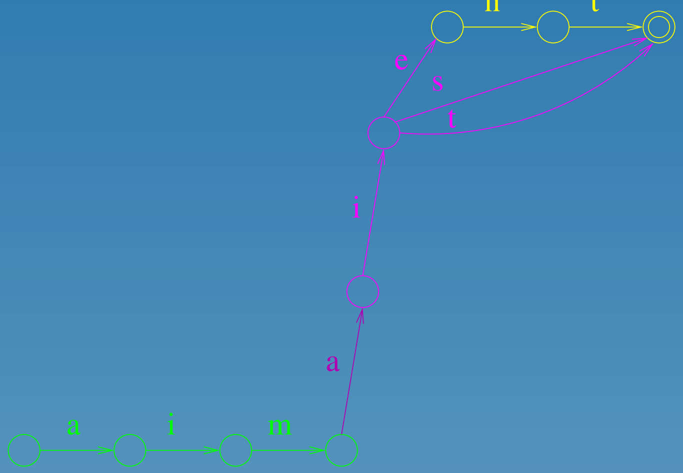
```
function sorted_construction;
  $w' \leftarrow \epsilon$;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
    while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
      $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    if $i \leq |w'|$ then
      repl_or_reg$(\delta(s, w_i), w'_{i+1\ldots|w'|})$;
    end if;
    while $i \leq |w|$ do
      $\delta(s, w_i) \leftarrow$ new state;
      $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
  end while;
  repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
  if $v \neq \epsilon$ then
    $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2\ldots|v|})$;
  end if;
  if $\exists_{r \in R}\, r \equiv q$ then
    delete $q$; return $r$;
  else
    $R \leftarrow R \cup \{q\}$; return $q$;
  end if;
end function;
```



aime

Incremental construction from sorted data – examples

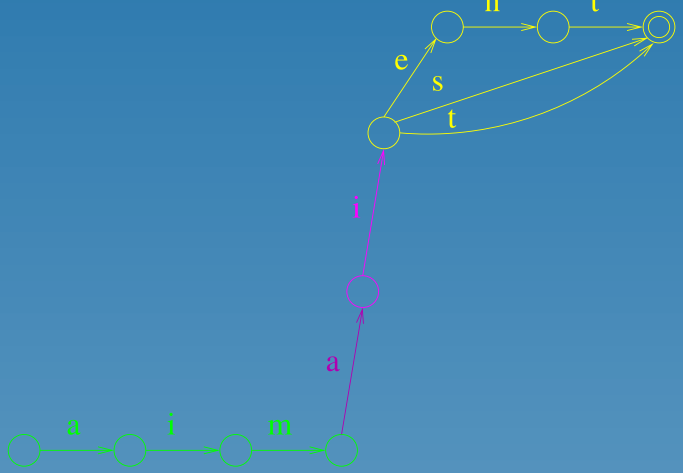
```
function sorted_construction;
    $w' \leftarrow \epsilon$;
    while input not empty do
        $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
        while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        if $i \leq |w'|$ then
            repl_or_reg$(\delta(s, w_i), w'_{i+1\ldots|w'|})$;
        end if;
        while $i \leq |w|$ do
            $\delta(s, w_i) \leftarrow$ new state;
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
    end while;
    repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
    if $v \neq \epsilon$ then
        $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2\ldots|v|})$;
    end if;
    if $\exists_{r \in R}\, r \equiv q$ then
        delete $q$; return $r$;
    else
        $R \leftarrow R \cup \{q\}$; return $q$;
    end if;
end function;
```



aime

Incremental construction from sorted data – examples

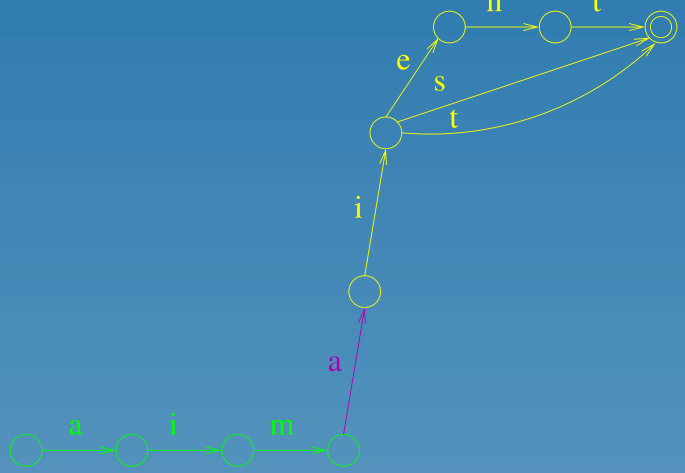
```
function sorted_construction;
    $w' \leftarrow \epsilon$;
    while input not empty do
        $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
        while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        if $i \leq |w'|$ then
            repl_or_reg($\delta(s, w_i), w'_{i+1\ldots|w'|}$);
        end if;
        while $i \leq |w|$ do
            $\delta(s, w_i) \leftarrow$ new state;
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
    end while;
    repl_or_reg($q_0, w'$);
end function;
function repl_or_reg($q, v$);
    if $v \neq \epsilon$ then
        $\delta(q, v_1) \leftarrow$ repl_or_reg($\delta(q, v_1), v_{2\ldots|v|}$);
    end if;
    if $\exists_{r \in R}\, r \equiv q$ then
        delete $q$; return $r$;
    else
        $R \leftarrow R \cup \{q\}$; return $q$;
    end if;
end function;
```



aime

Incremental construction from sorted data – examples

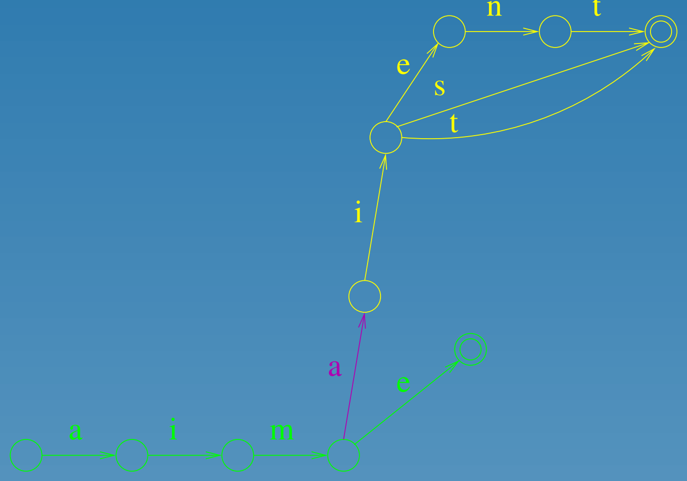
```
function sorted_construction;
   $w' \leftarrow \epsilon$;
   while input not empty do
      $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
      while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
         $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
      end while;
      if $i \leq |w'|$ then
         repl_or_reg$(\delta(s, w_i), w'_{i+1\ldots|w'|})$;
      end if;
      while $i \leq |w|$ do
         $\delta(s, w_i) \leftarrow$ new state;
         $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
      end while;
      $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
   end while;
   repl_or_reg$(q_0, w')$;
end function;
function repl_or_reg$(q, v)$;
   if $v \neq \epsilon$ then
      $\delta(q, v_1) \leftarrow$ repl_or_reg$(\delta(q, v_1), v_{2\ldots|v|})$;
   end if;
   if $\exists_{r \in R}\, r \equiv q$ then
      delete $q$; return $r$;
   else
      $R \leftarrow R \cup \{q\}$; return $q$;
   end if;
end function;
```



aime

Incremental construction from sorted data – examples

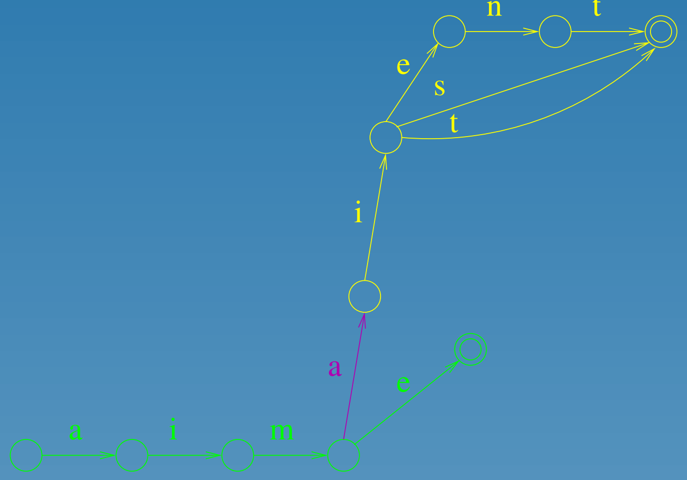
```
function sorted_construction;
    $w' \leftarrow \epsilon$;
    while input not empty do
        $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
        while $i \le |w|$ and $\delta(s, w_i)! = \perp$ do
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        if $i \le |w'|$ then
            repl_or_reg($\delta(s, w_i), w'_{i+1\ldots|w'|}$);
        end if;
        while $i \le |w|$ do
            $\delta(s, w_i) \leftarrow$ new state;
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
    end while;
    repl_or_reg($q_0, w'$);
end function;
function repl_or_reg($q, v$);
    if $v \neq \epsilon$ then
        $\delta(q, v_1) \leftarrow$ repl_or_reg($\delta(q, v_1), v_{2\ldots|v|}$);
    end if;
    if $\exists_{r \in R}\, r \equiv q$ then
        delete $q$; return $r$;
    else
        $R \leftarrow R \cup \{q\}$; return $q$;
    end if;
end function;
```



aime

Incremental construction from sorted data – examples

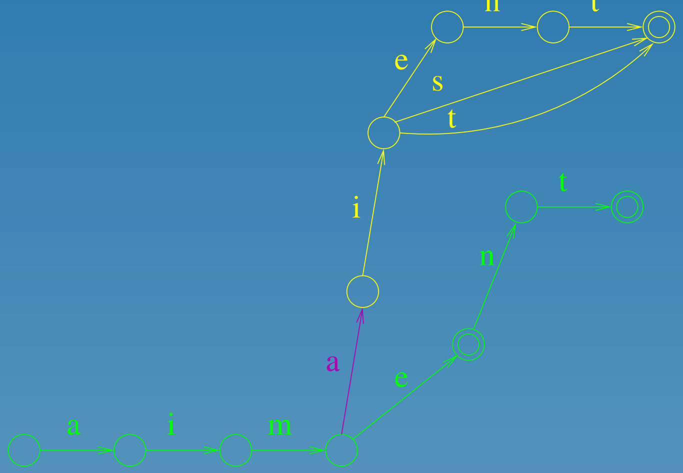
```
function sorted_construction;
  $w' \leftarrow \epsilon$;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
    while $i \le |w|$ and $\delta(s, w_i)! = \perp$ do
      $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    if $i \le |w'|$ then
      repl_or_reg($\delta(s, w_i), w'_{i+1\ldots|w'|}$);
    end if;
    while $i \le |w|$ do
      $\delta(s, w_i) \leftarrow$ new state;
      $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
  end while;
  repl_or_reg($q_0, w'$);
end function;
function repl_or_reg($q, v$);
  if $v \neq \epsilon$ then
    $\delta(q, v_1) \leftarrow$ repl_or_reg($\delta(q, v_1), v_{2\ldots|v|}$);
  end if;
  if $\exists_{r \in R}\, r \equiv q$ then
    delete $q$; return $r$;
  else
    $R \leftarrow R \cup \{q\}$; return $q$;
  end if;
end function;
```



aiment

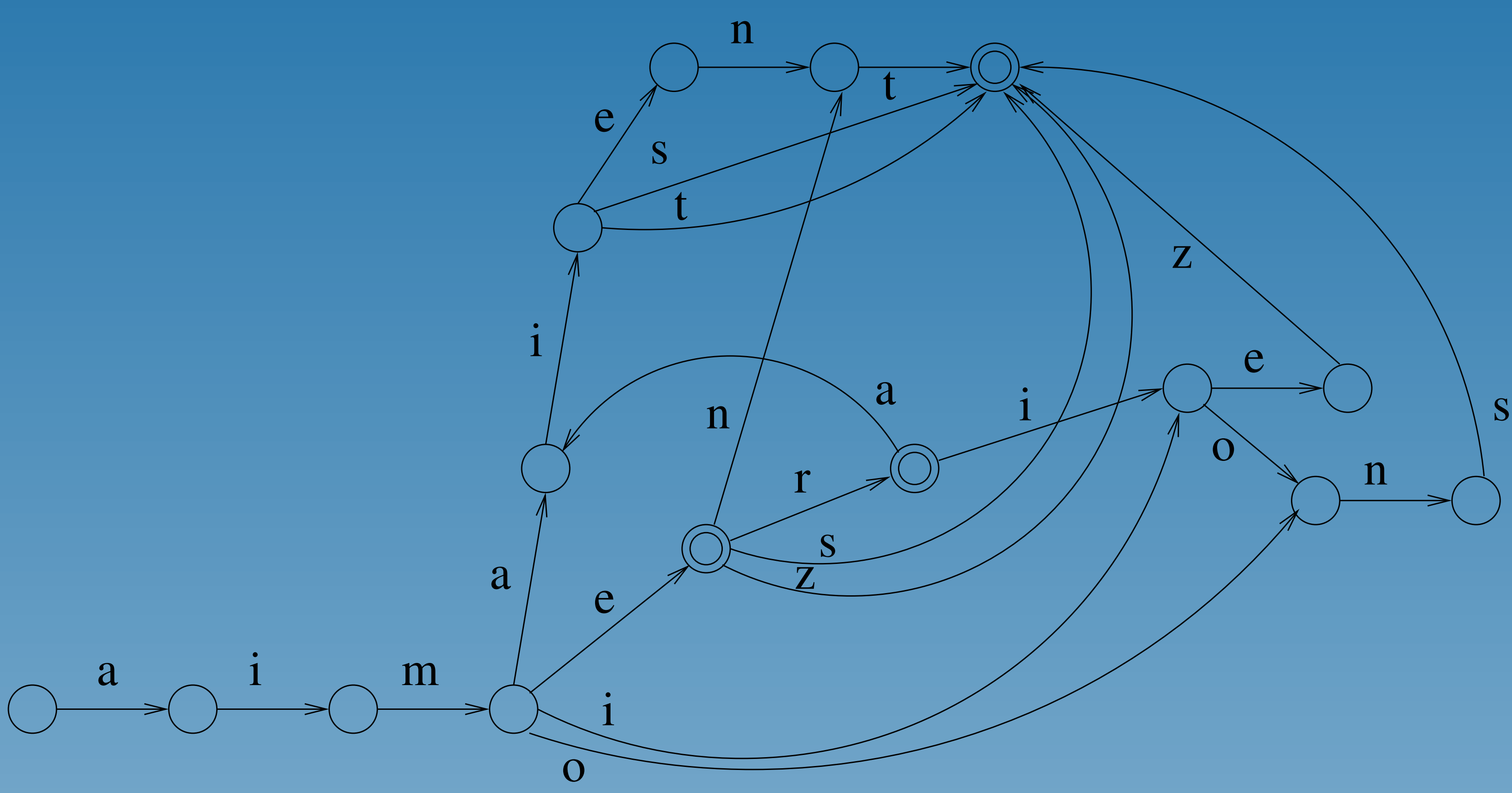
Incremental construction from sorted data – examples

```
function sorted_construction;
    $w' \leftarrow \epsilon$;
    while input not empty do
        $s \leftarrow q_0$; $i \leftarrow 1$; $w \leftarrow$ next word;
        while $i \leq |w|$ and $\delta(s, w_i)! = \perp$ do
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        if $i \leq |w'|$ then
            repl_or_reg($\delta(s, w_i), w'_{i+1 \ldots |w'|}$);
        end if;
        while $i \leq |w|$ do
            $\delta(s, w_i) \leftarrow$ new state;
            $s \leftarrow \delta(s, w_i)$; $i \leftarrow i + 1$;
        end while;
        $F \leftarrow F \cup \{s\}$; $w' \leftarrow w$
    end while;
    repl_or_reg($q_0, w'$);
end function;
function repl_or_reg($q, v$);
    if $v \neq \epsilon$ then
        $\delta(q, v_1) \leftarrow$ repl_or_reg($\delta(q, v_1), v_{2 \ldots |v|}$);
    end if;
    if $\exists_{r \in R}\, r \equiv q$ then
        delete $q$; return $r$;
    else
        $R \leftarrow R \cup \{q\}$; return $q$;
    end if;
end function;
```

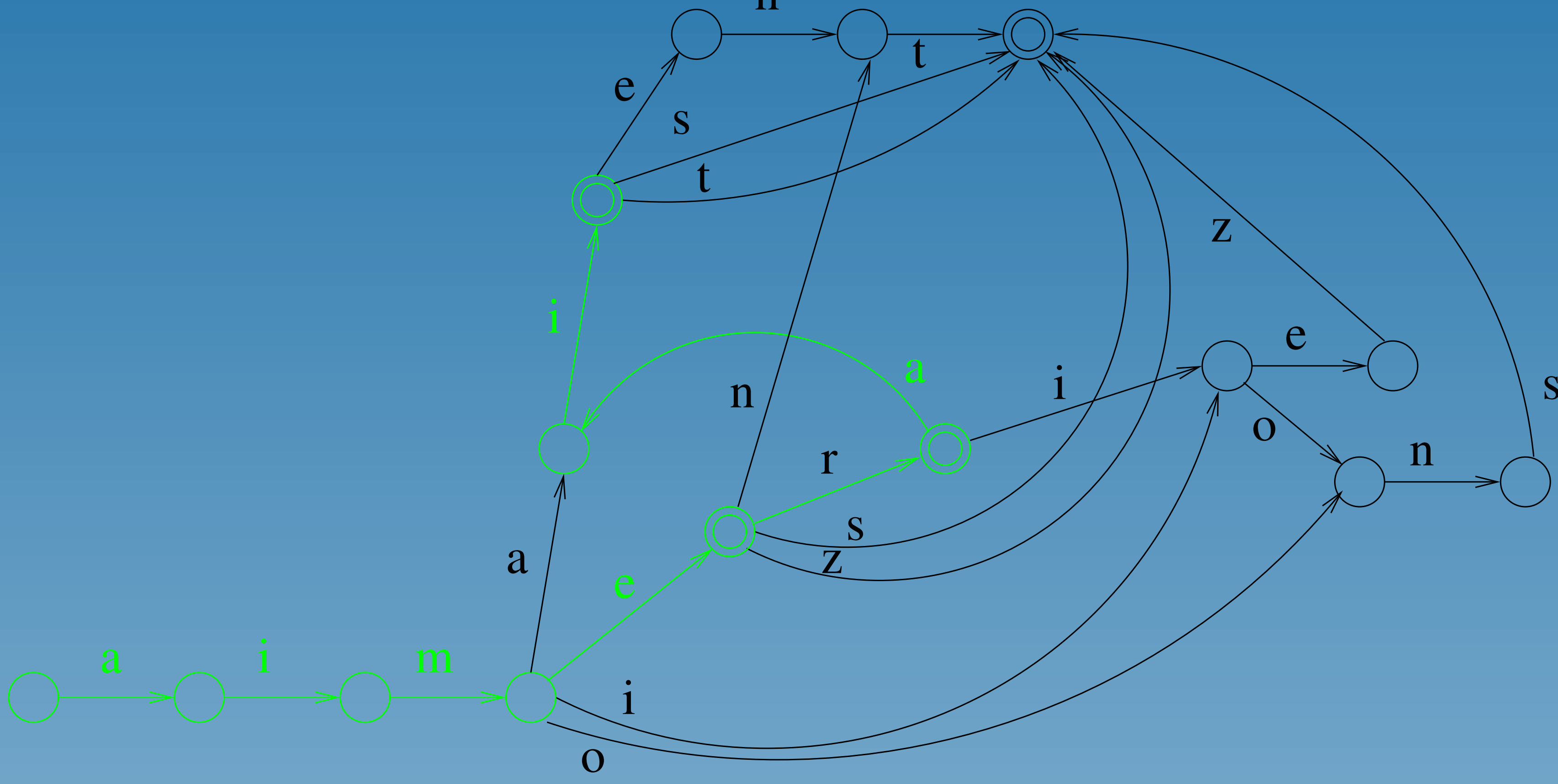


aiment

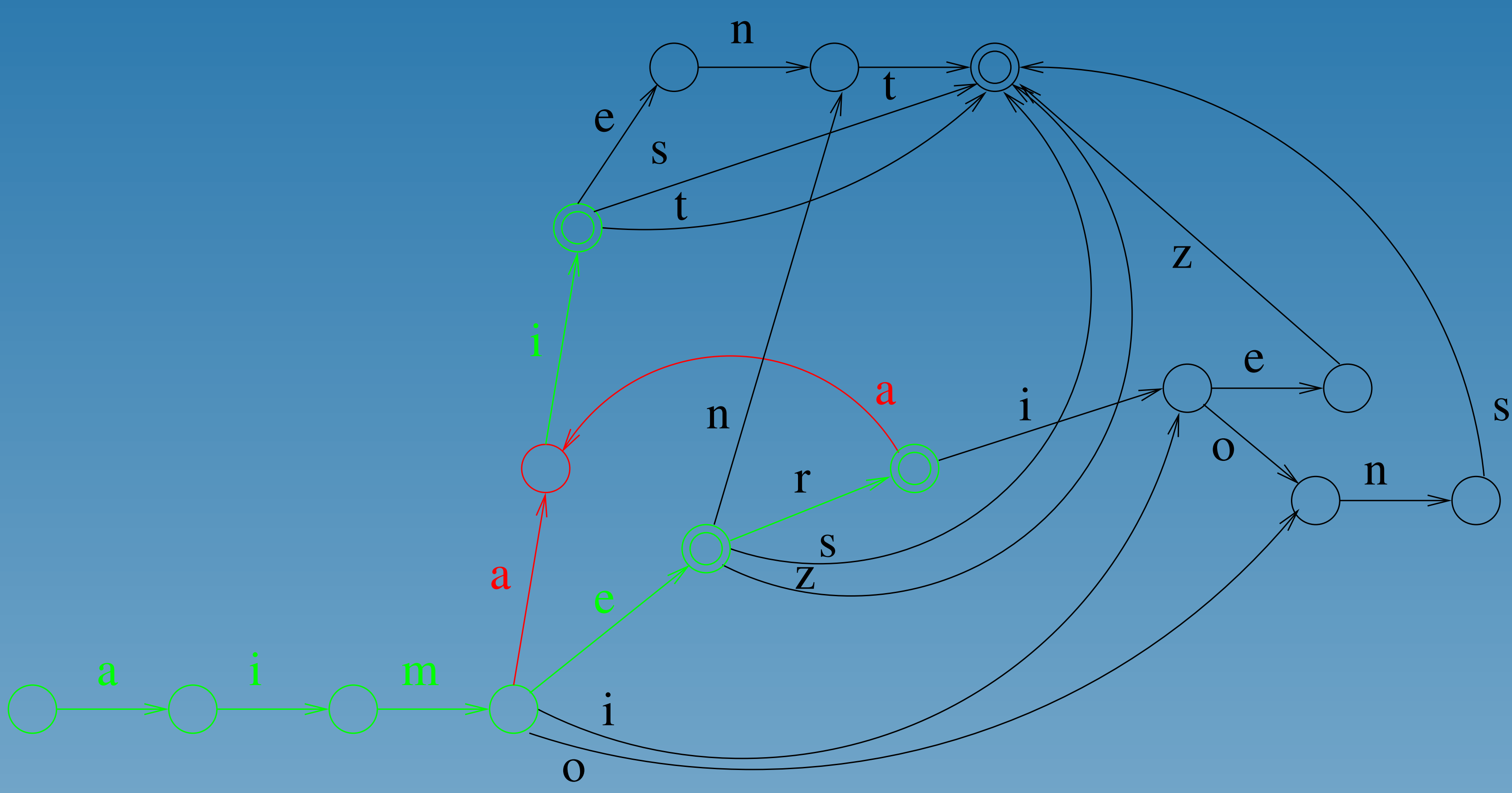
Unsorted data – hidden dangers



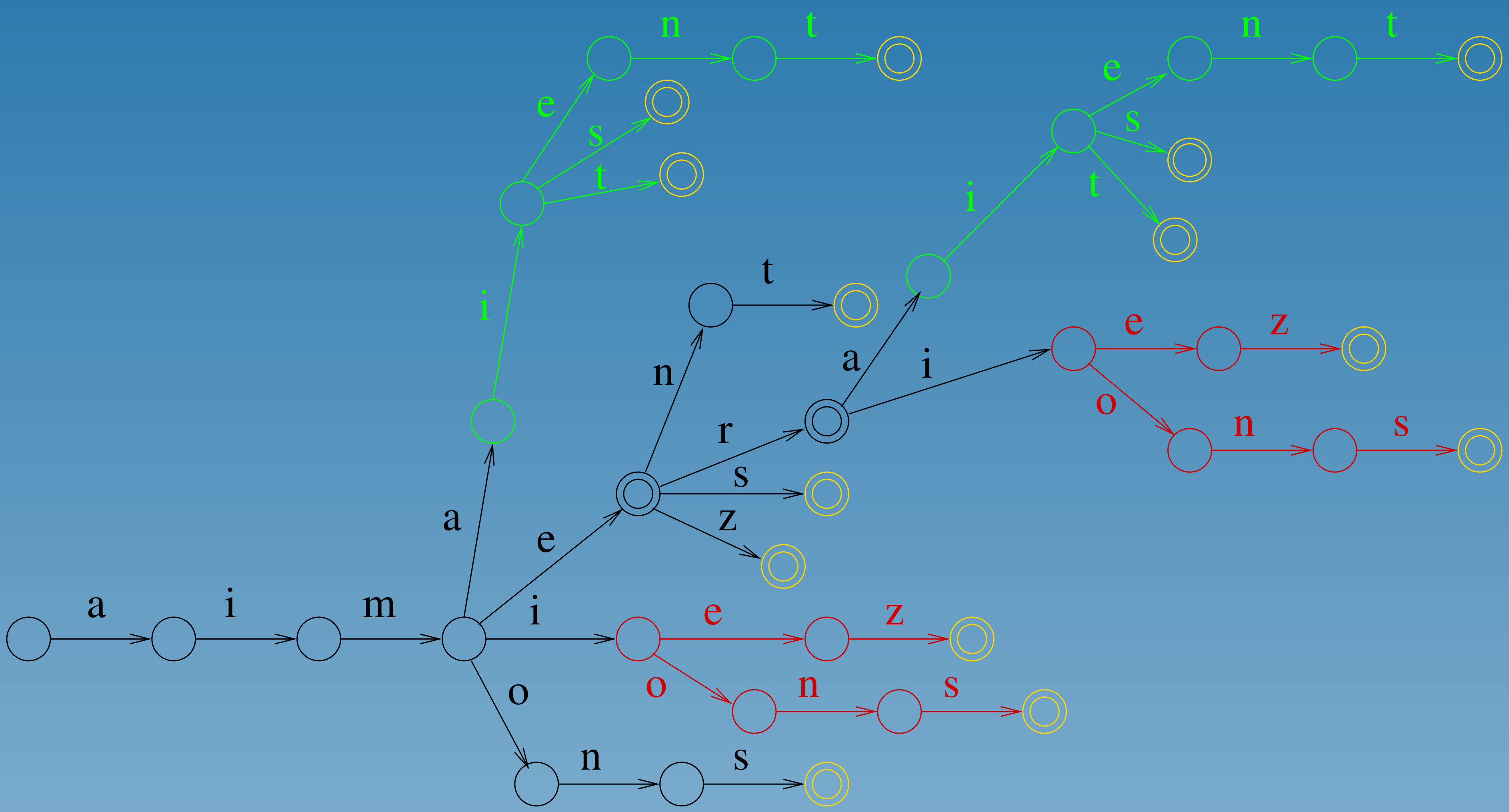
Unsorted data – hidden dangers



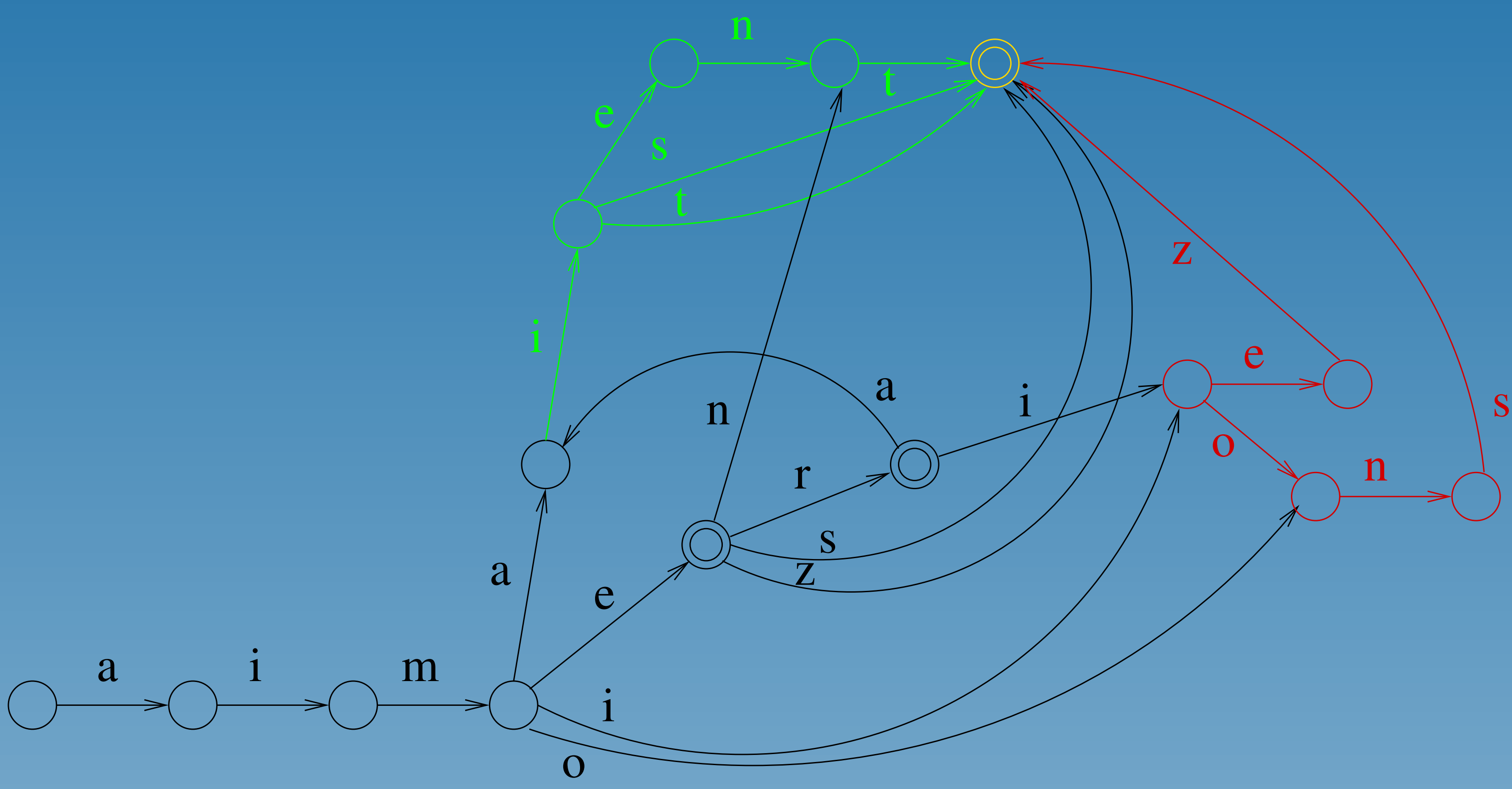
Unsorted data – hidden dangers



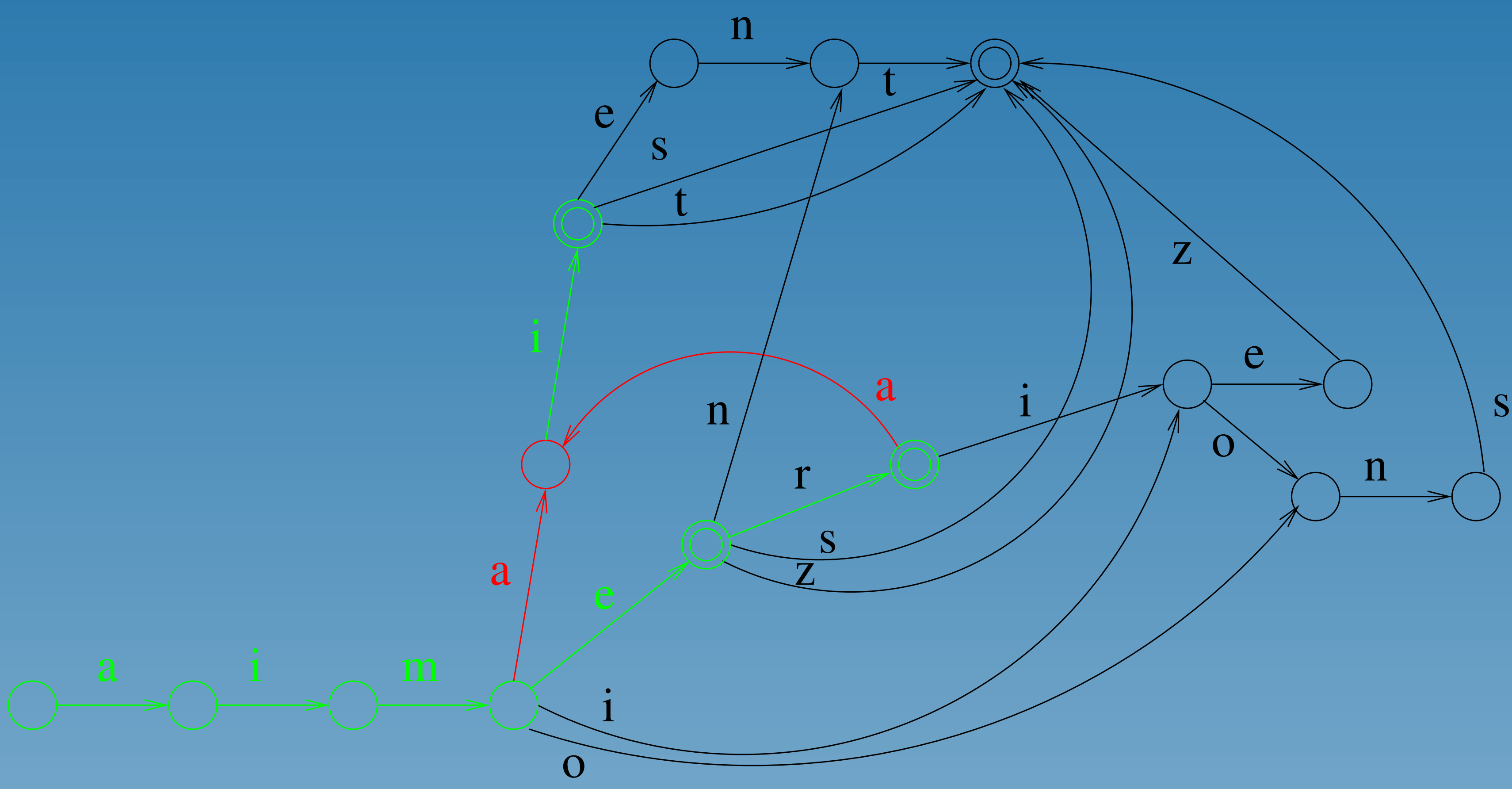
Unsorted data – hidden dangers



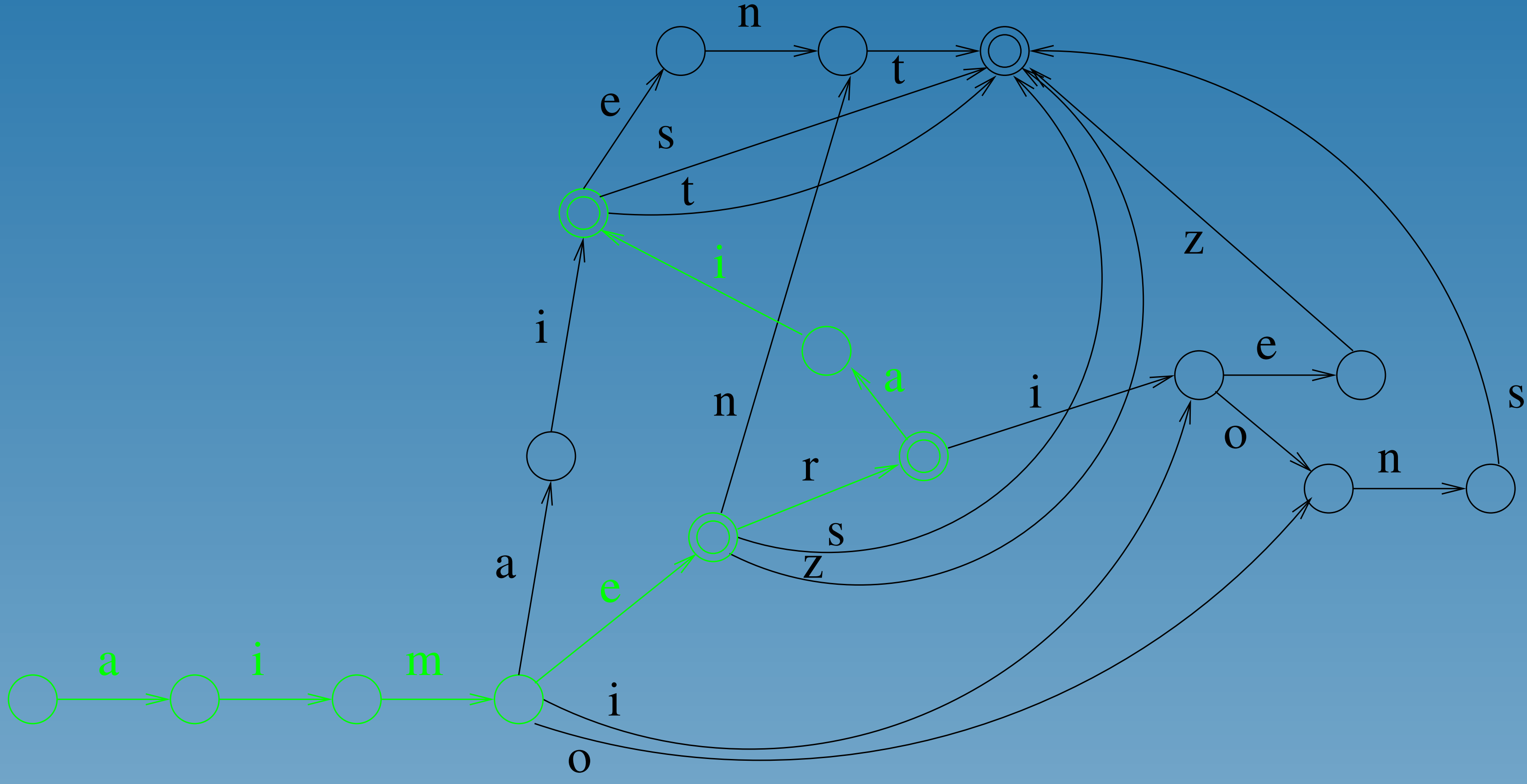
Unsorted data – hidden dangers



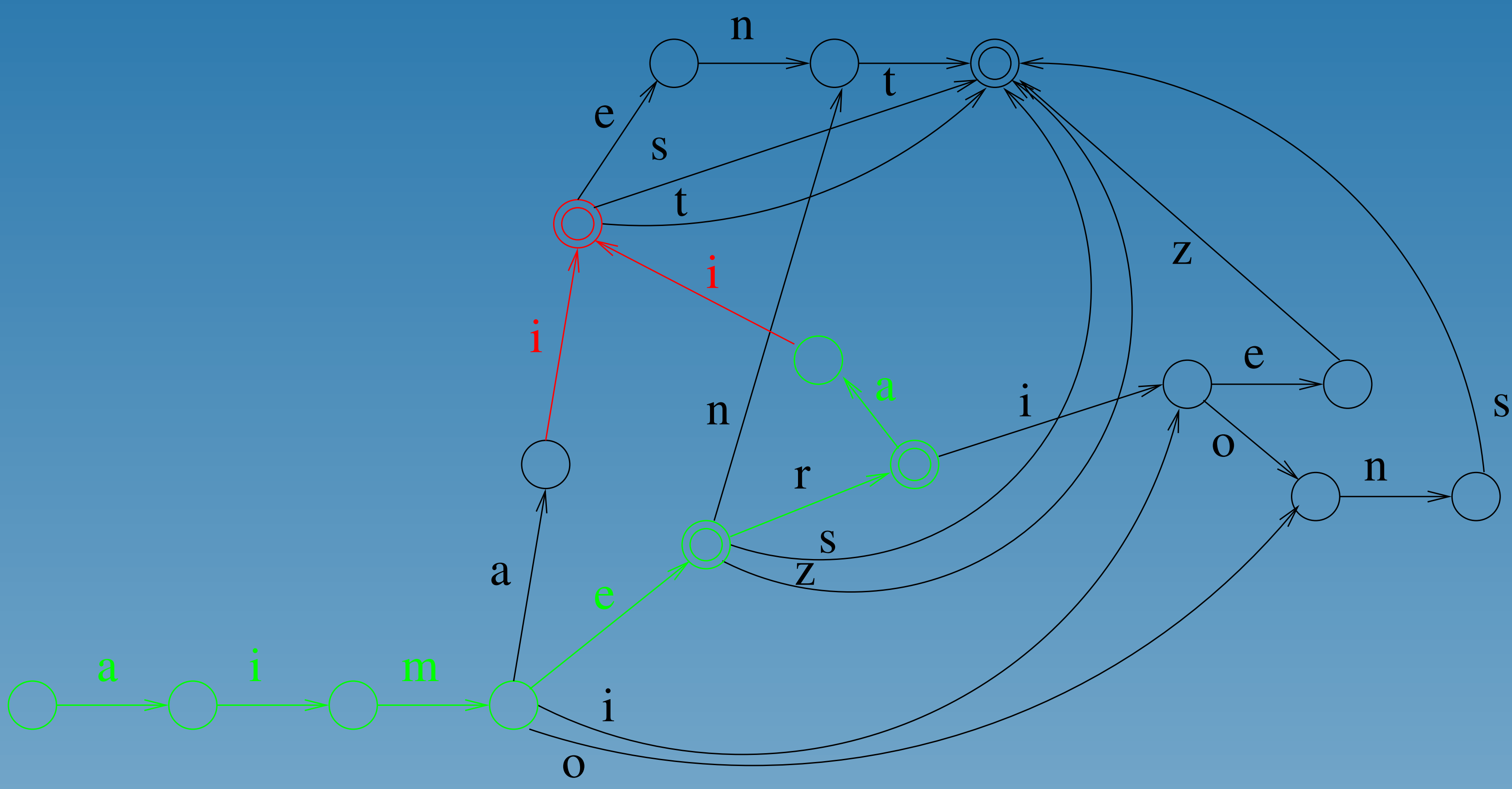
Unsorted data – hidden dangers



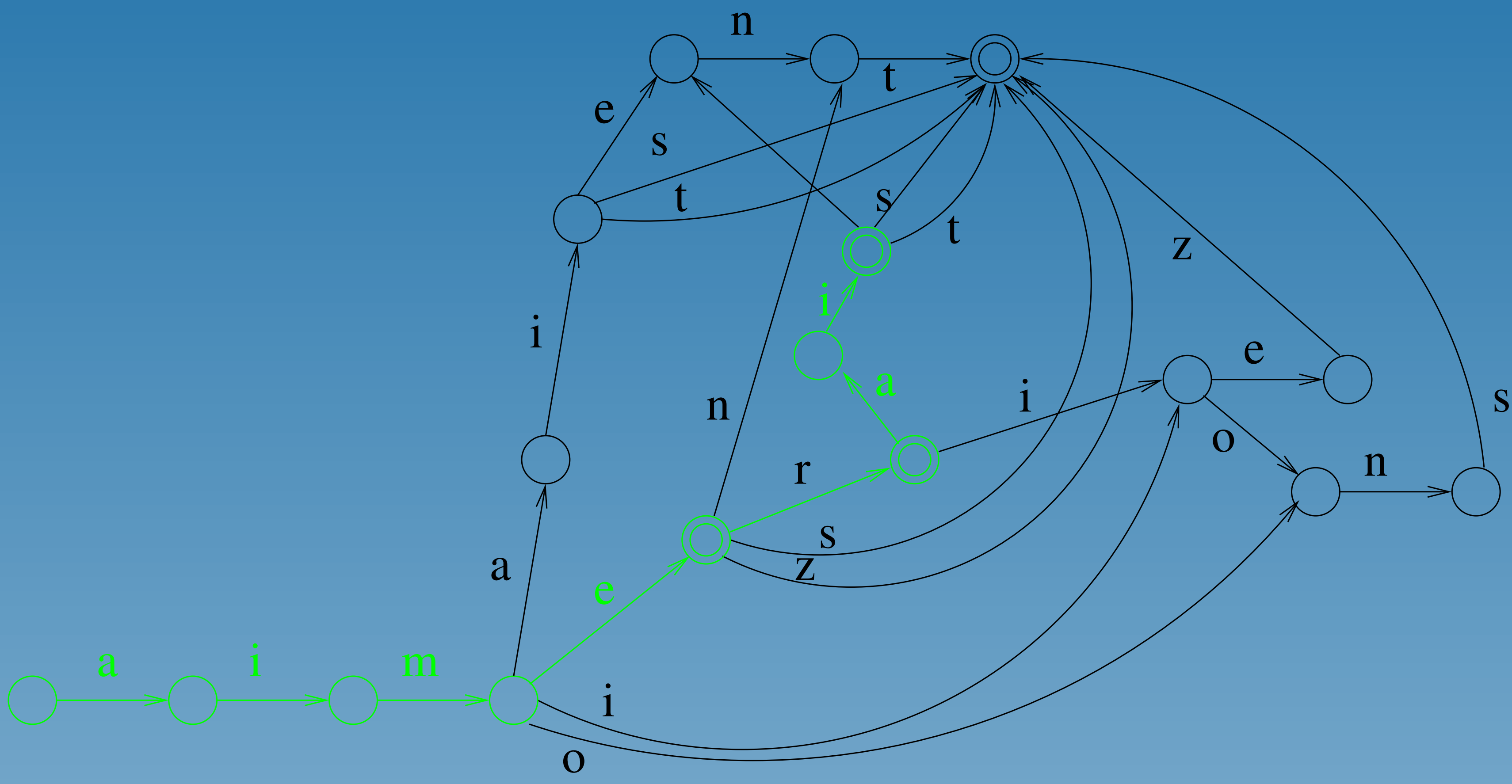
Unsorted data – hidden dangers



Unsorted data – hidden dangers



Unsorted data – hidden dangers



Incremental construction from unsorted data

```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$; end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$; if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```

Incremental construction from unsorted data

```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R} r \equiv s$ then
        if $i = u$ and $i > 0$ then $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$; end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$; if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```

Incremental construction from unsorted data

```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$ and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$; end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$; if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```

Incremental construction from unsorted data

```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$; end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$; if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```

Incremental construction from unsorted data

```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$ and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R} r \equiv s$ then
        if $i = u$ and $i > 0$ then $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$; end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$; if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```

Incremental construction from unsorted data

```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$ and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$; end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$; if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```

Incremental construction from unsorted data

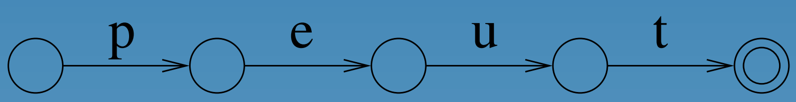
```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then $R \leftarrow R \setminus \{\text{top}(P)\}$; $u \leftarrow u - 1$; end if;
        delete $s$; $\delta(\text{top}(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$; if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```

Incremental construction from unsorted data

```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R} r \equiv s$ then
        if $i = u$ and $i > 0$ then $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$; end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$; if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```

Incr. constr. from unsorted data – examples

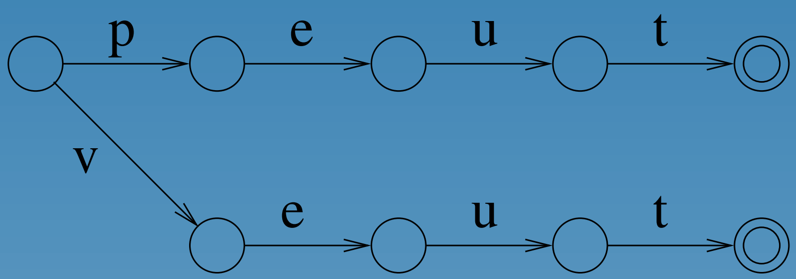
```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$
    and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then
          $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
        end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$;
        if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```



peut

Incr. constr. from unsorted data – examples

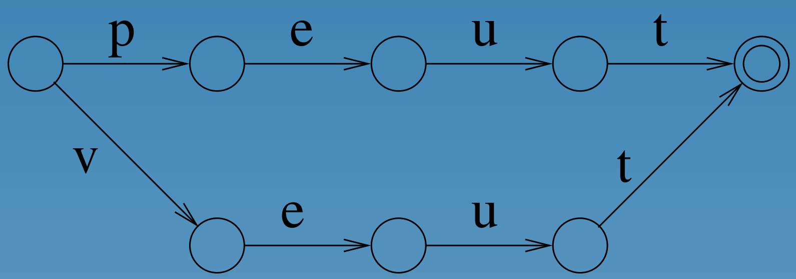
```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$
    and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then
          $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
        end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$;
        if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```



veut

Incr. constr. from unsorted data – examples

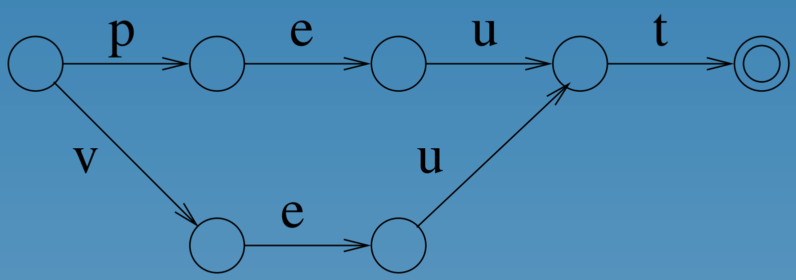
```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$
    and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then
          $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
        end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$;
        if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```



veut

Incr. constr. from unsorted data – examples

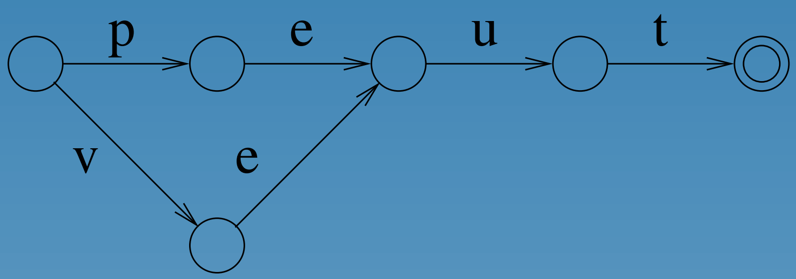
```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$
    and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R} r \equiv s$ then
        if $i = u$ and $i > 0$ then
          $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
        end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$;
        if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```



veut

Incr. constr. from unsorted data – examples

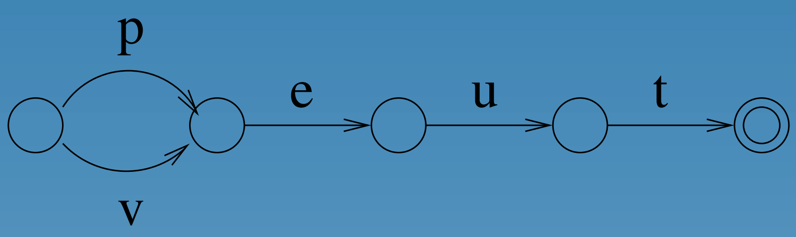
```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$
    and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R} r \equiv s$ then
        if $i = u$ and $i > 0$ then
          $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
        end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$;
        if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```



veut

Incr. constr. from unsorted data – examples

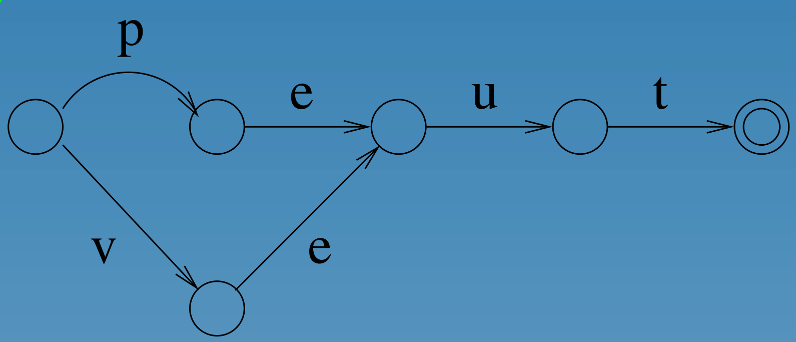
```
function unsorted_construction;
   while input not empty do
      $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
      while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$
      and fanin$(\delta(s, w_i)) \leq 1$ do
         $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
      end while;
      $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
      while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
         $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
         push$(s, P)$; $i \leftarrow i + 1$;
      end while;
      while $i \leq |w|$ do
         $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
         push$(s, P)$; $i \leftarrow i + 1$;
      end while;
      $F \leftarrow F \cup \{s\}$; pop$(P)$;
      while $P$ not empty do
         if $\exists_{r \in R} r \equiv s$ then
            if $i = u$ and $i > 0$ then
               $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
            end if;
            delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
         else
            $R \leftarrow R \cup \{s\}$;
            if $i = u$ then break; end if;
         end if;
         $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
      end while;
      reset $P$;
   end while;
end function;
```



veut

Incr. constr. from unsorted data – examples

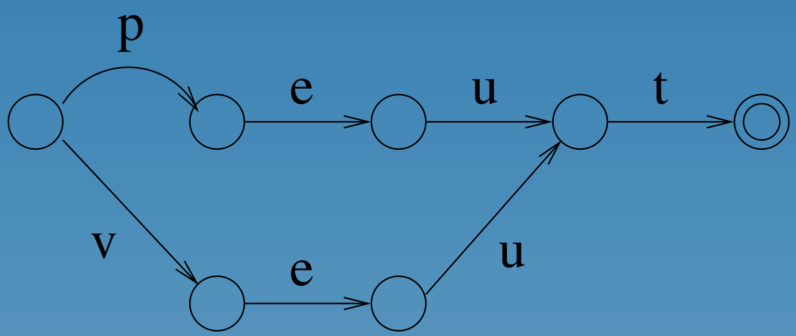
```
function unsorted_construction;
    while input not empty do
        $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
        while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$
        and fanin$(\delta(s, w_i)) \leq 1$ do
            $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
        end while;
        $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
        while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
            $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
            push$(s, P)$; $i \leftarrow i + 1$;
        end while;
        while $i \leq |w|$ do
            $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
            push$(s, P)$; $i \leftarrow i + 1$;
        end while;
        $F \leftarrow F \cup \{s\}$; pop$(P)$;
        while $P$ not empty do
            if $\exists_{r \in R}\, r \equiv s$ then
                if $i = u$ and $i > 0$ then
                    $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
                end if;
                delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
            else
                $R \leftarrow R \cup \{s\}$;
                if $i = u$ then break; end if;
            end if;
            $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
        end while;
        reset $P$;
    end while;
end function;
```



veux

Incr. constr. from unsorted data – examples

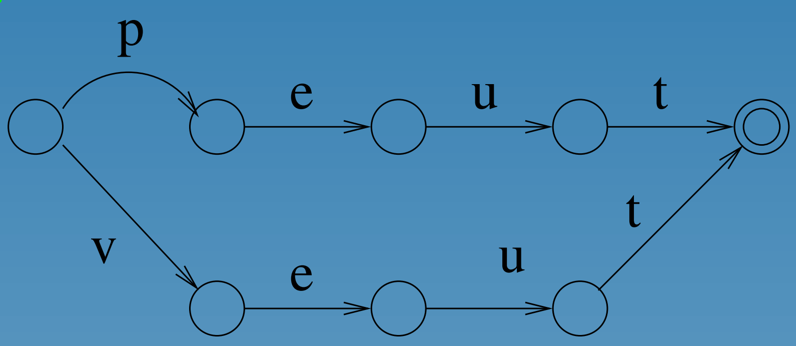
```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$
    and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then
          $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
        end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$;
        if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```



veux

Incr. constr. from unsorted data – examples

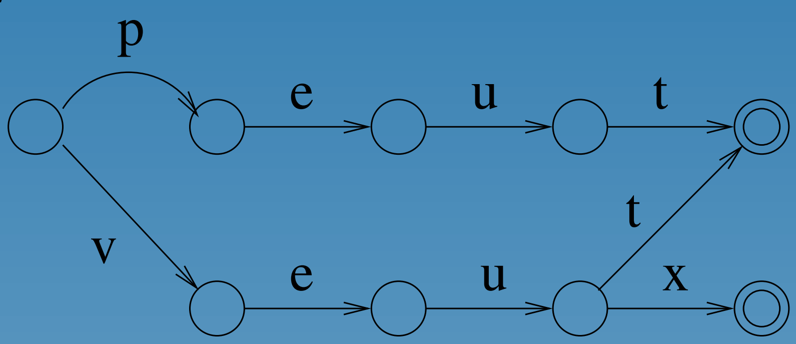
```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$
    and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then
          $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
        end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$;
        if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```



veux

Incr. constr. from unsorted data – examples

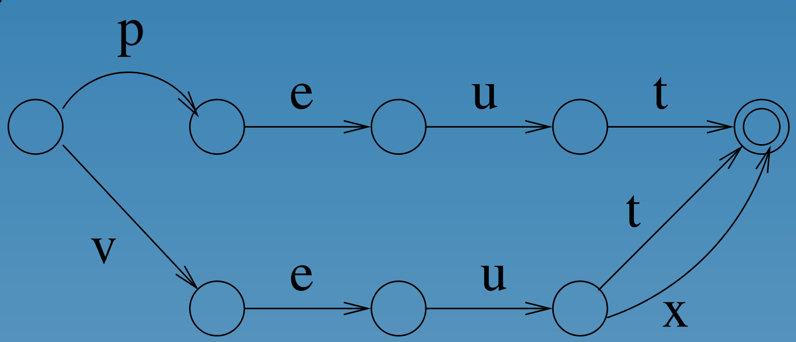
```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$
    and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then
          $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
        end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$;
        if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```



veux

Incr. constr. from unsorted data – examples

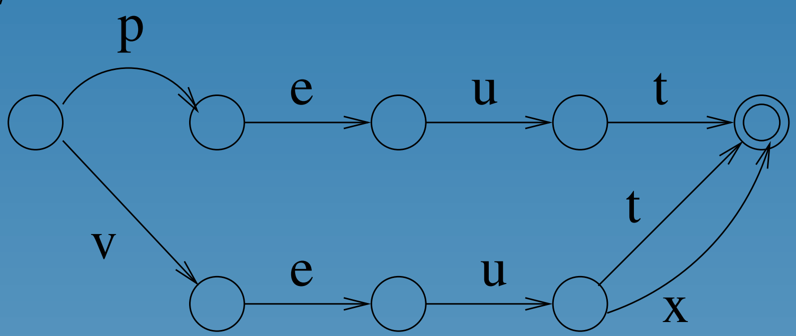
```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$
    and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then
          $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
        end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$;
        if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```



veux

Incr. constr. from unsorted data – examples

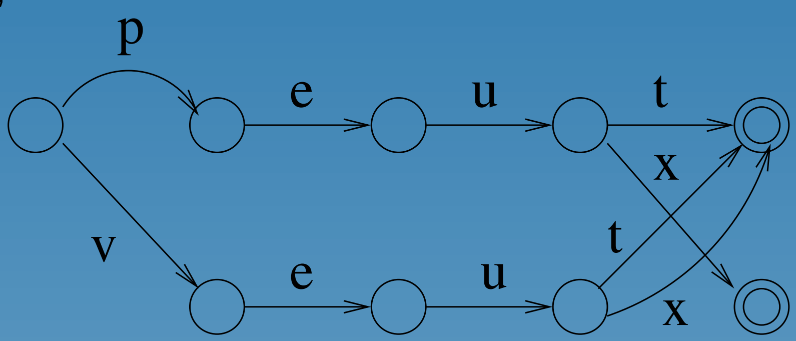
```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$
    and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then
          $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
        end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$;
        if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```



peux

Incr. constr. from unsorted data – examples

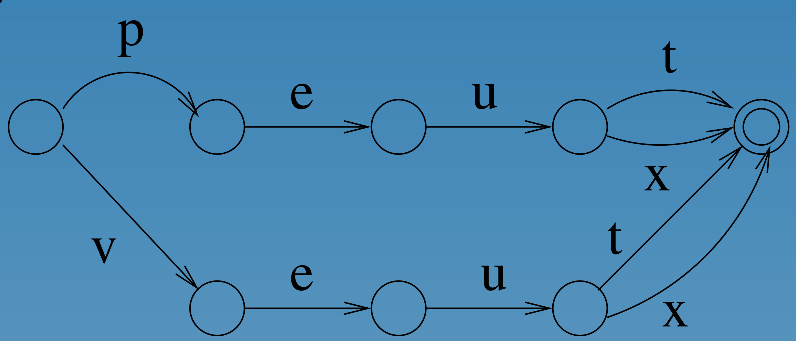
```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$
    and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then
          $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
        end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$;
        if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```



peux

Incr. constr. from unsorted data – examples

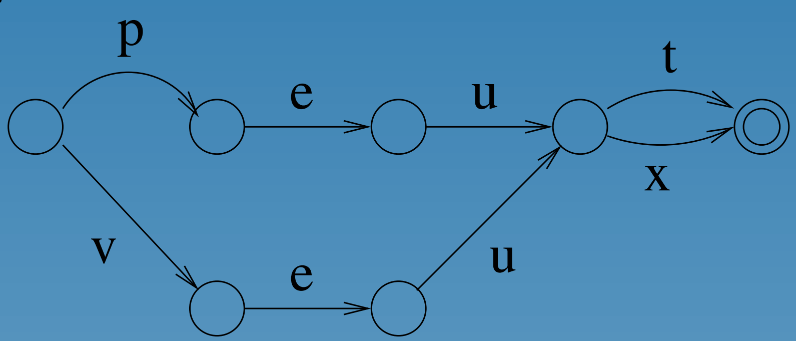
```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$
    and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then
          $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
        end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$;
        if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```



veux

Incr. constr. from unsorted data – examples

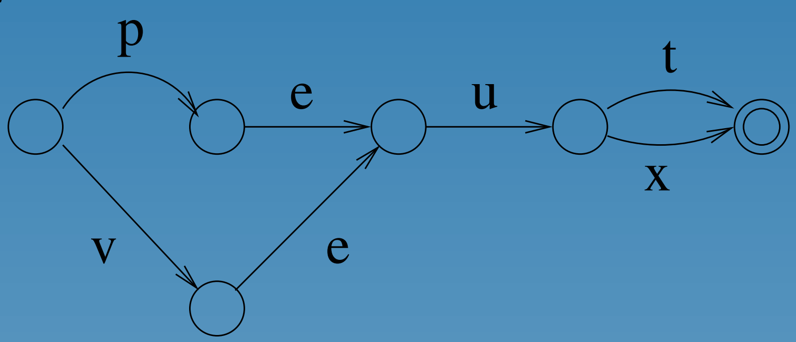
```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$
    and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \perp$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then
          $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
        end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$;
        if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```



veux

Incr. constr. from unsorted data – examples

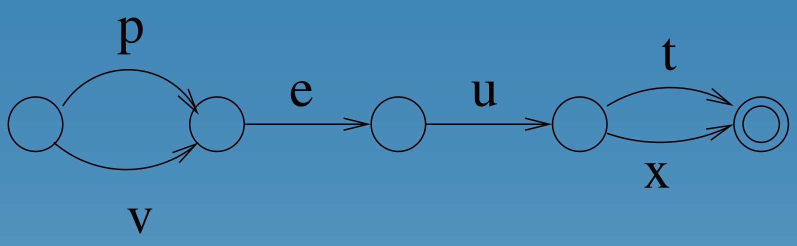
```
function unsorted_construction;
  while input not empty do
    $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$
    and fanin$(\delta(s, w_i)) \leq 1$ do
      $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
    while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$ do
      $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    while $i \leq |w|$ do
      $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
      push$(s, P)$; $i \leftarrow i + 1$;
    end while;
    $F \leftarrow F \cup \{s\}$; pop$(P)$;
    while $P$ not empty do
      if $\exists_{r \in R}\, r \equiv s$ then
        if $i = u$ and $i > 0$ then
          $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
        end if;
        delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
      else
        $R \leftarrow R \cup \{s\}$;
        if $i = u$ then break; end if;
      end if;
      $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
    end while;
    reset $P$;
  end while;
end function;
```



veux

Incr. constr. from unsorted data – examples

```
function unsorted_construction;
   while input not empty do
      $s \leftarrow q_0$; $i \leftarrow 0$; $w \leftarrow$ next word; push$(s, P)$;
      while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$
      and fanin$(\delta(s, w_i)) \leq 1$ do
         $s \leftarrow \delta(s, w_i)$; push$(s, P)$; $i \leftarrow i + 1$;
      end while;
      $R \leftarrow R \setminus \{s\}$; $u \leftarrow i$;
      while $i \leq |w|$ and $\delta(s, w_i) \neq \bot$ do
         $\delta(s, w_i) \leftarrow$ clone$(\delta(s, w_i))$; $s \leftarrow \delta(s, w_i)$;
         push$(s, P)$; $i \leftarrow i + 1$;
      end while;
      while $i \leq |w|$ do
         $s \leftarrow$ new state; $s \leftarrow \delta(s, w_i)$;
         push$(s, P)$; $i \leftarrow i + 1$;
      end while;
      $F \leftarrow F \cup \{s\}$; pop$(P)$;
      while $P$ not empty do
         if $\exists_{r \in R}\, r \equiv s$ then
            if $i = u$ and $i > 0$ then
               $R \leftarrow R \setminus \{$top$(P)\}$; $u \leftarrow u - 1$;
            end if;
            delete $s$; $\delta($top$(P), w_i) \leftarrow r$;
         else
            $R \leftarrow R \cup \{s\}$;
            if $i = u$ then break; end if;
         end if;
         $i \leftarrow i - 1$; $s \leftarrow$ pop$(P)$;
      end while;
      reset $P$;
   end while;
end function;
```



peux

Complexity and performance

- Both algorithms keep intermediate automata (almost) minimal
- Both run in time proportional to input data size
- The algorithm for sorted data is faster but less flexible
- Traditional algorithms are slower and use much more memory
- There are extensions of both algorithms to the case off adding words to a cyclic automaton