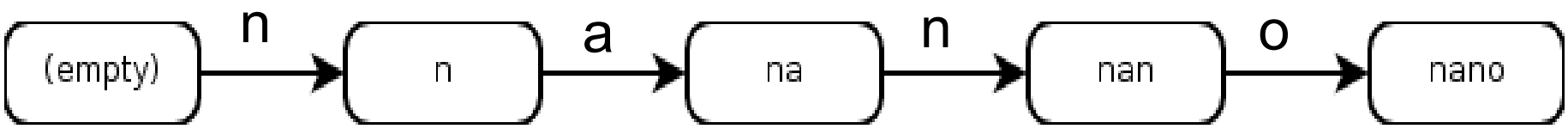
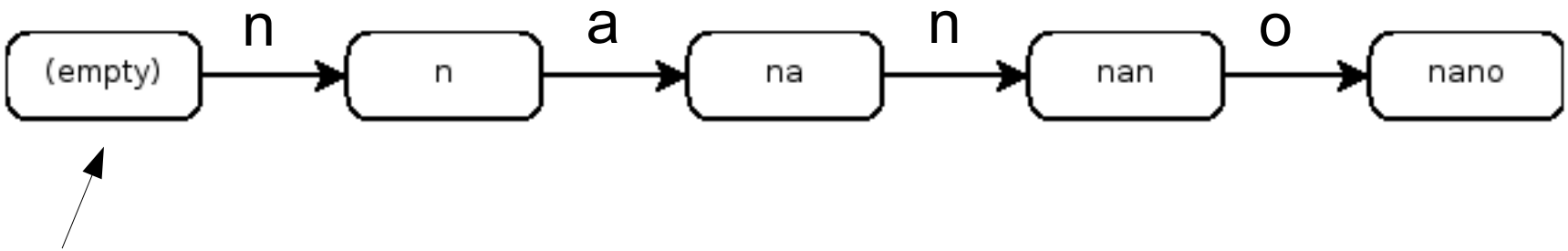
String Algorithms

Matching

Finite State Machine

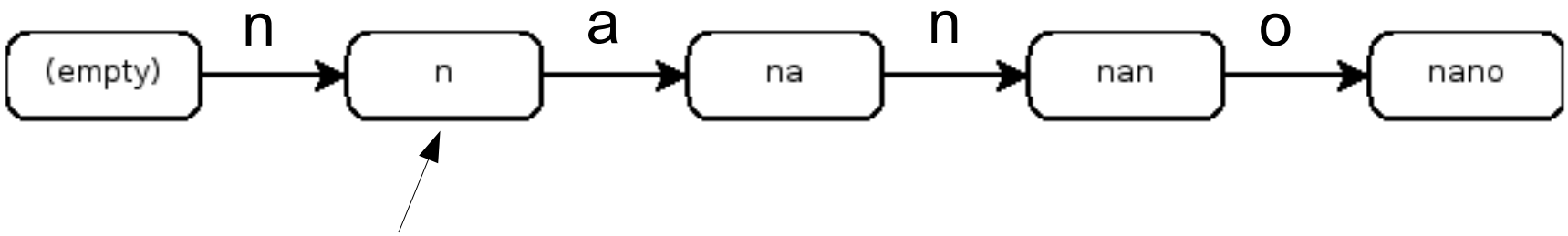


Simulation



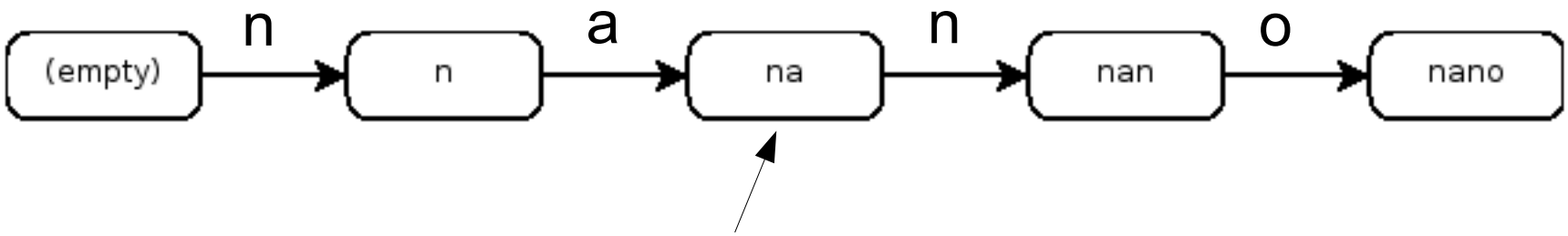
n a o u n a n a n o

Simulation



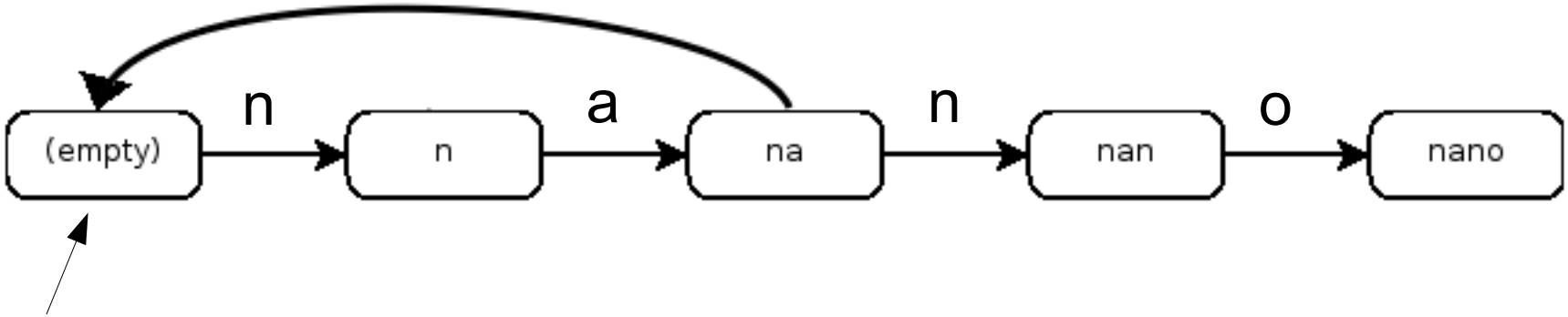
n a o u n a n a n o

Simulation



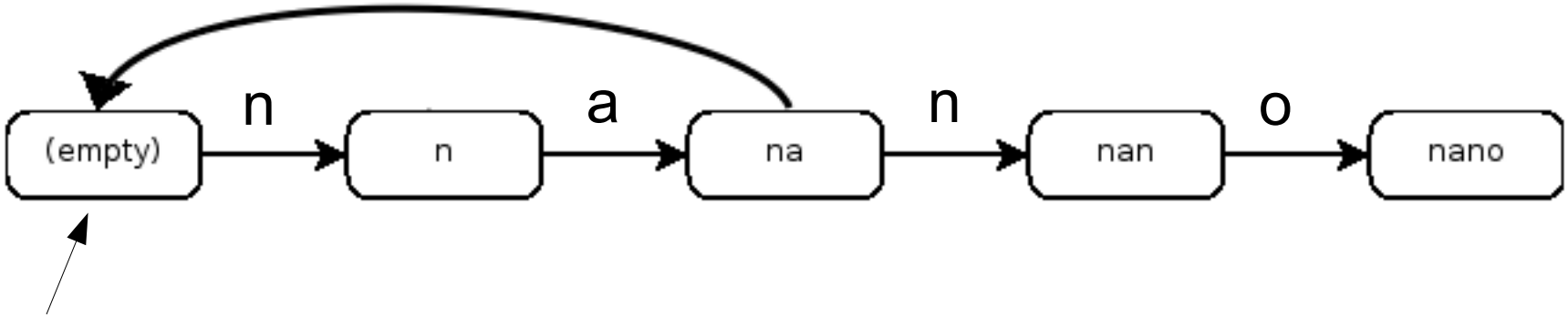
n a o u n a n a n o

Simulation



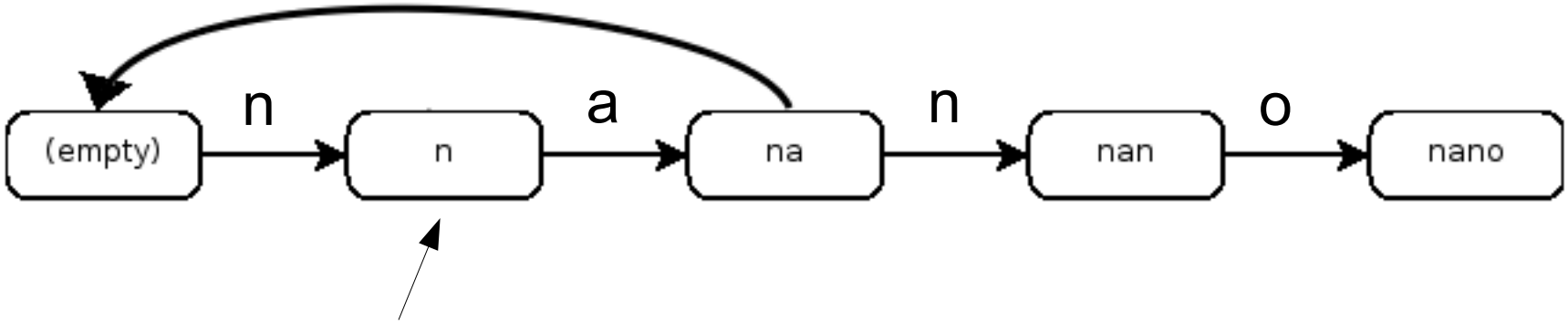
n a o u n a n a n o

Simulation



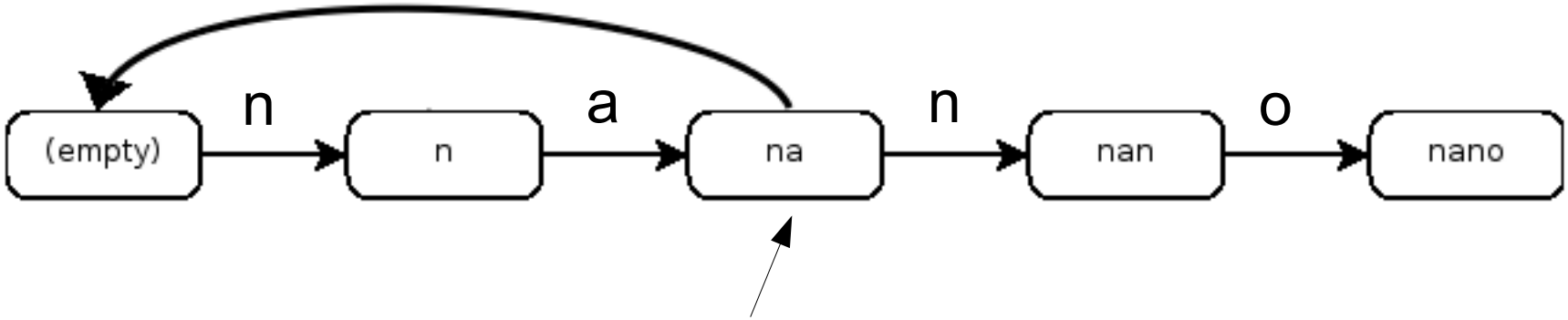
n a o u n a n a n o

Simulation



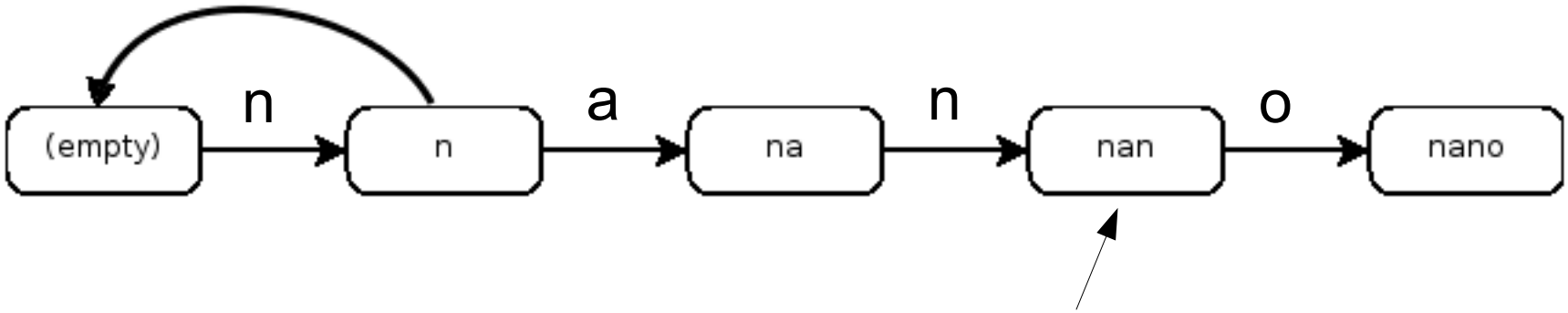
n a o u n a n a n o

Simulation



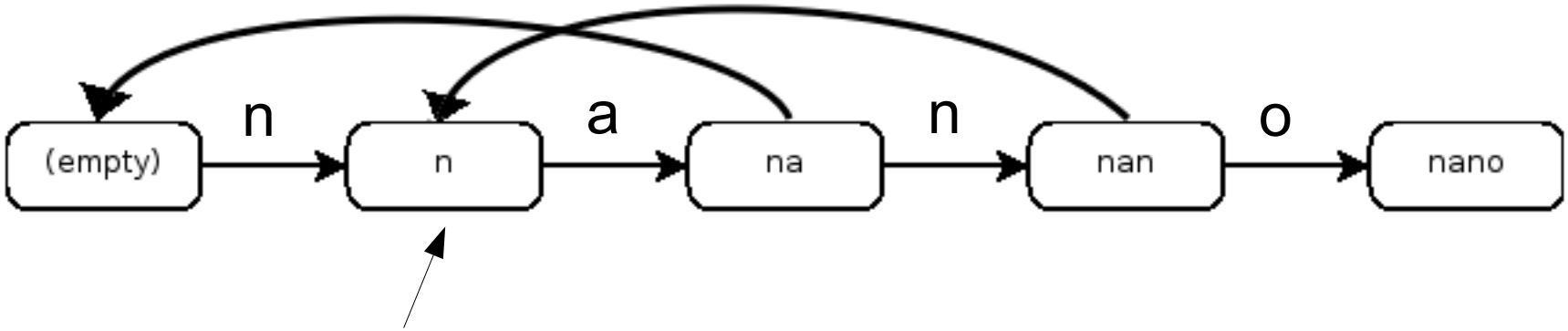
n a o u n a n a n o

Simulation



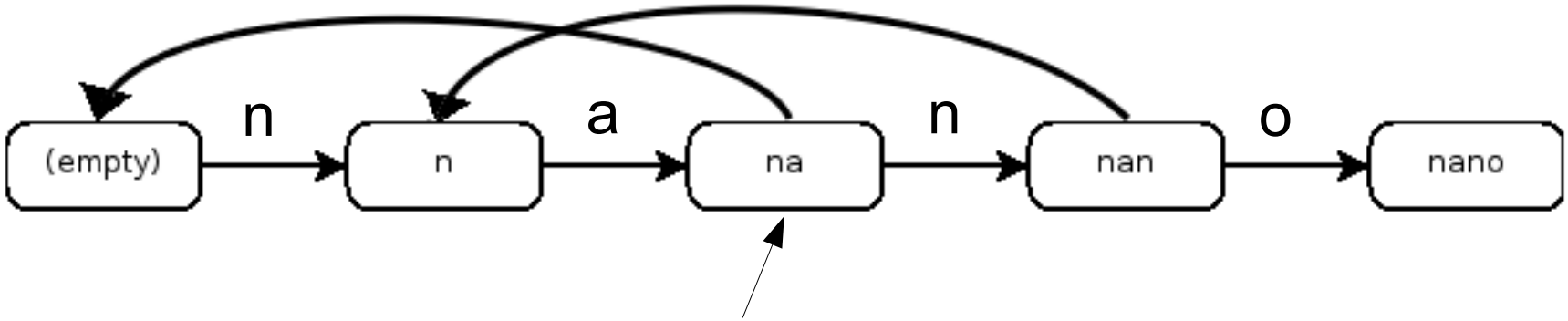
n a o u n a n a n o

Simulation



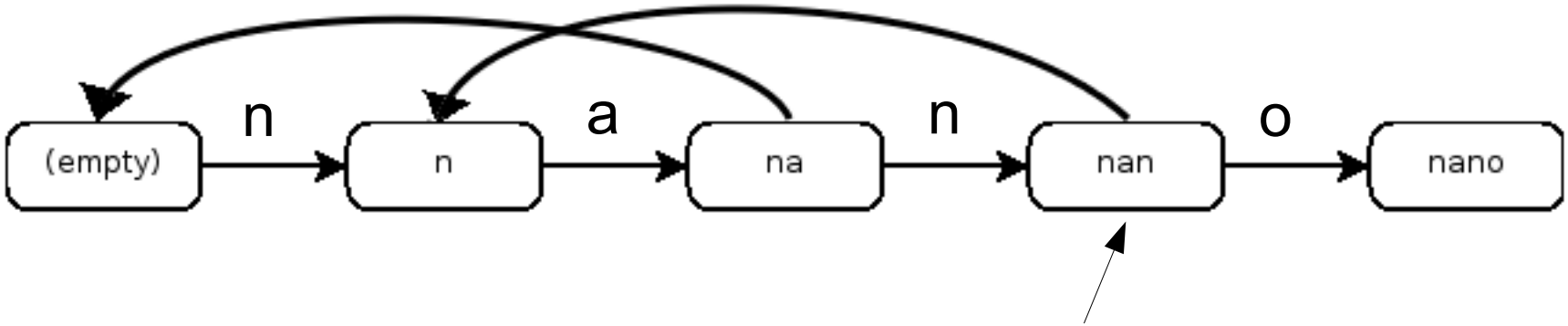
n a o u n a n a n o

Simulation



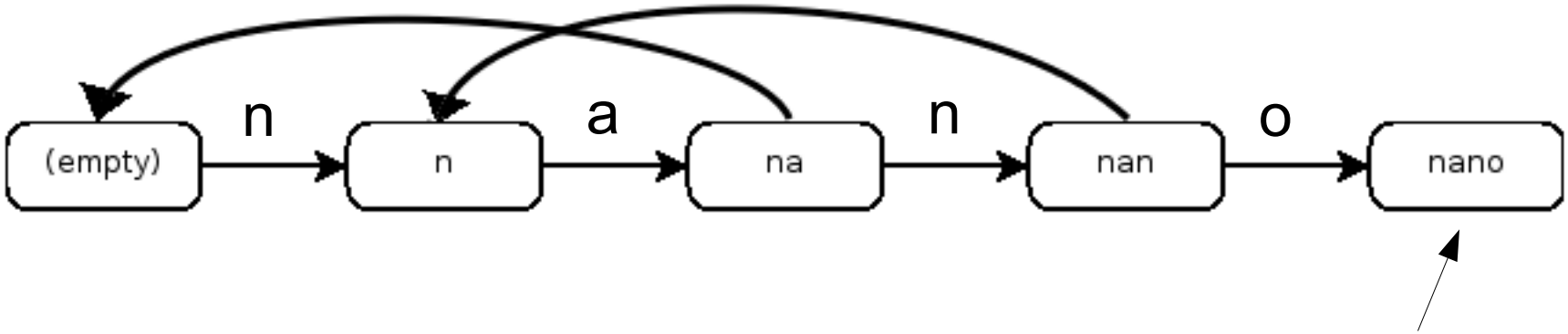
n a o u n a n a n o

Simulation



n a o u n a n a n o

Simulation

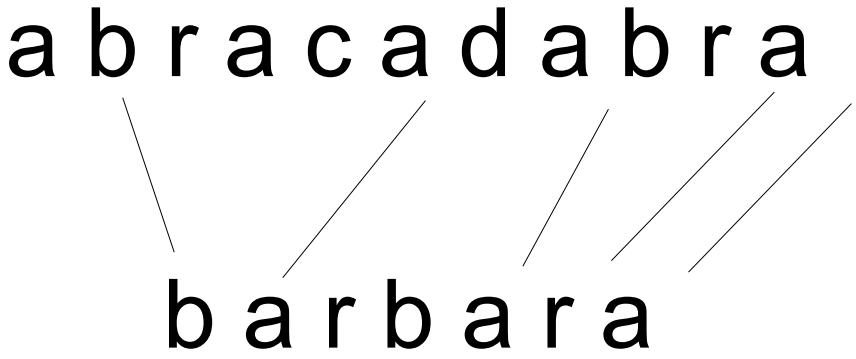


n a o u n a n a n o

String Algorithms

Longest Common Subsequence

Recursive Solution



Recursive Solution

```
int brute_lcs(const char *s1, const char *s2)
{
  if ((*s1 == '\0') || (*s2 == '\0')) {
    return 0;
  }

  if (*s1 == *s2) {
    return 1 + brute_lcs(s1+1, s2+1);
  } else {
    return max(brute_lcs(s1+1, s2), brute_lcs(s1, s2+1));
  }
}
```

Dynamic Programming

```
int lcs_arr[M][N];

int dyn_lcs(const char *s1, const char *s2, int i, int j)
{
  if (lcs_arr[i][j] == 0) {

    if ((s1[i] == '\0') || (s2[j] == '\0')) {
      lcs_arr[i][j] = 1;
    } else if (s1[i] == s2[j]) {
      lcs_arr[i][j] = 1 + dyn_lcs(s1, s2, i+1, i+2) + 1;
    } else {
      lcs_arr[i][j] = max(dyn_lcs(s1, s2, i+1, j),
                          dyn_lcs(s1, s2, i, j+1)) + 1;
    }

  }

  return lcs_arr[i][j] - 1;
}
```

Dynamic Programming

```
int lcs_arr[MAXM][MAXN];

int dyn_lcs(const char *s1, const char *s2)
{
  int i, j;
  int n, m;

  n = strlen(s1);
  m = strlen(s2);

  for (i=n; i>=0; i--) {
    for (j=m; j>=0; j--) {

      if ((s1[i] == '\0') || (s2[j] == '\0')) {
        lcs_arr[i][j] = 0;
      } else if (s1[i] == s2[j]) {
        lcs_arr[i][j] = 1 + lcs_arr[i+1][j+1];
      } else {
        lcs_arr[i][j] = max(lcs_arr[i+1][j],
                            lcs_arr[i][j+1]);
      }

    }
  }

  return lcs_arr[0][0];
}
```

Dynamic Programming

```
for (i=n; i>=0; i--) {
  for (j=m; j>=0; j--) {

    if ((s1[j] == '\0') || (s2[i] == '\0')) {
      row1[j] = 0;
    } else if (s1[j] == s2[i]) {
      row1[j] = 1 + row2[j+1];
    } else {
      row1[j] = max(row1[j+1], row2[j]);
    }
  }

  for (j=0; j<=m; j++) {
    row2[j] = row1[j];
  }
}

ans = row2[0];
```

String Algorithms

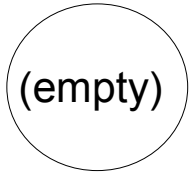
Suffix Tries

Suffix Tries

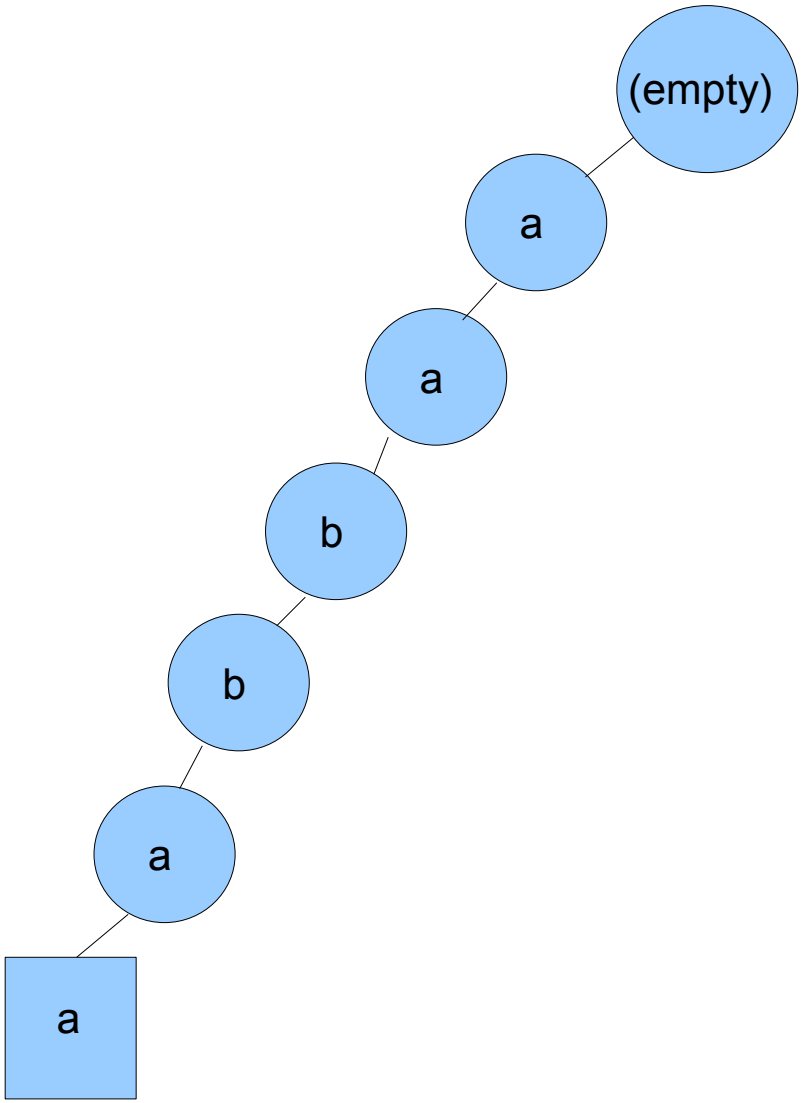
a a b b a a

0. a a b b a a
1. a b b a a
2. b b a a
3. b a a
4. a a
5. a

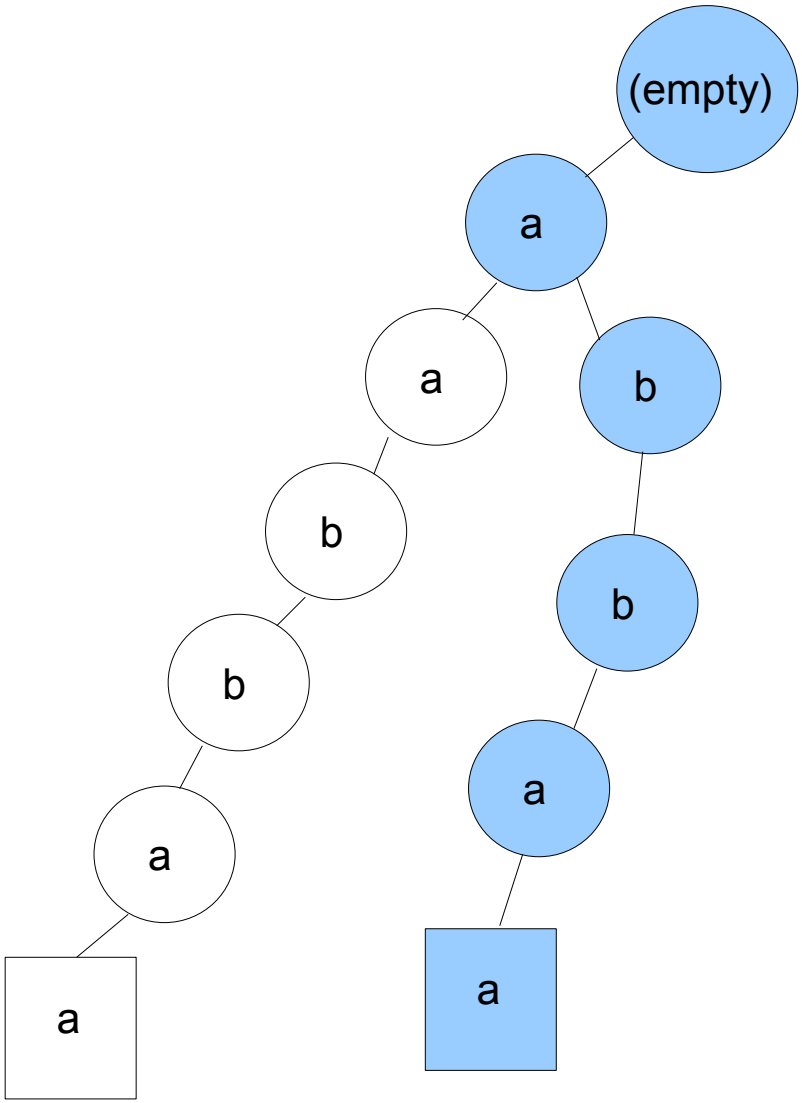
Suffix Tries



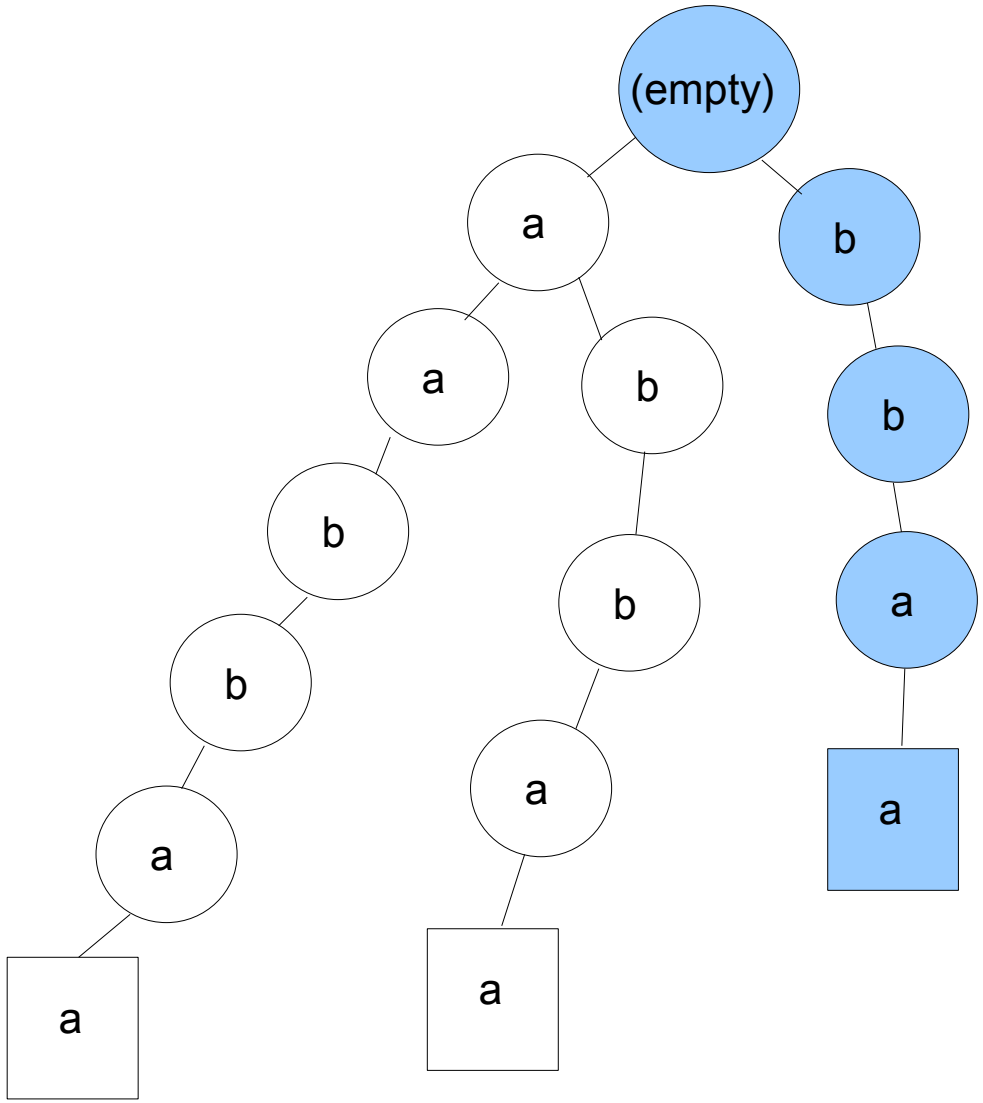
Suffix Tries



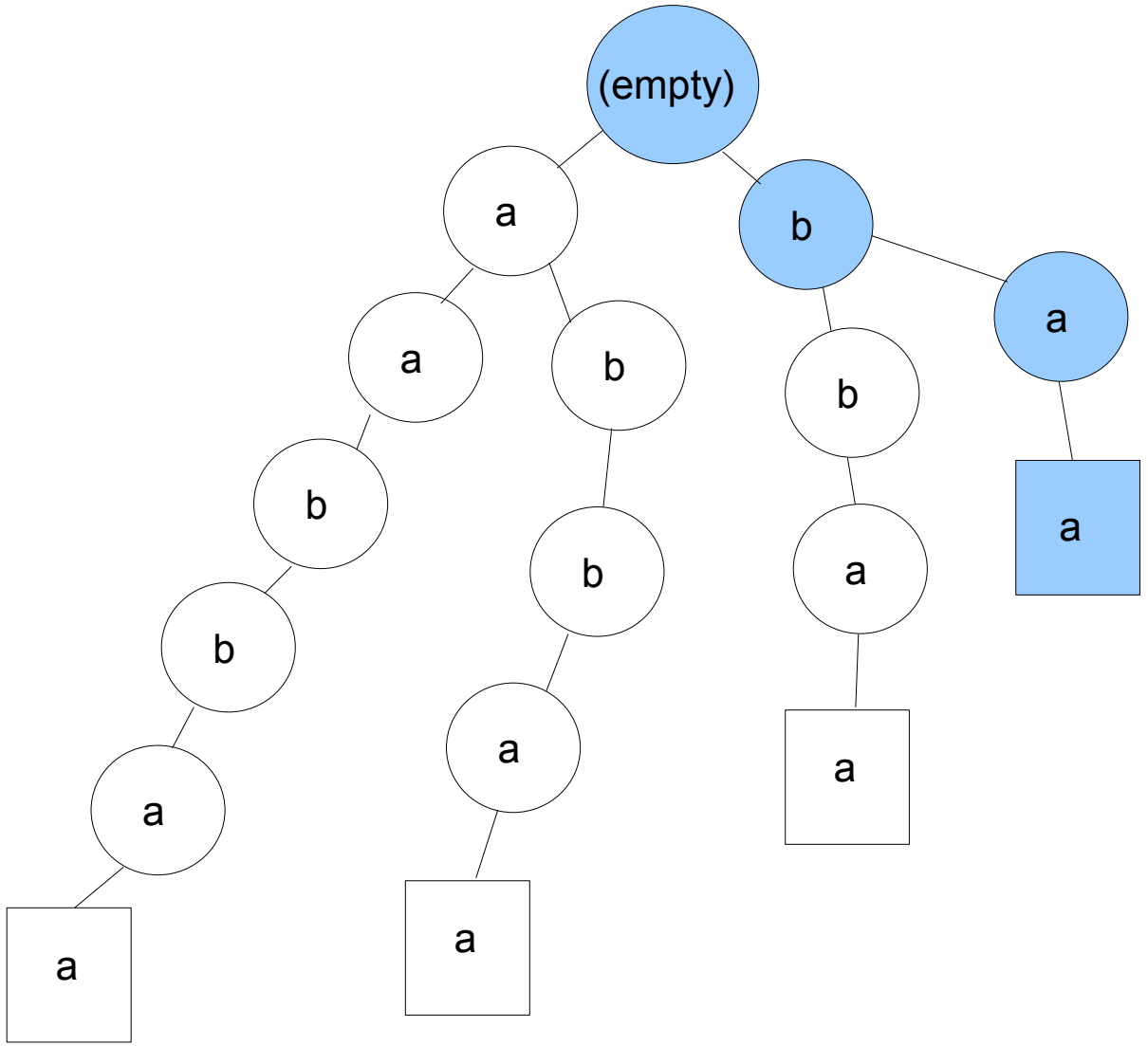
Suffix Tries



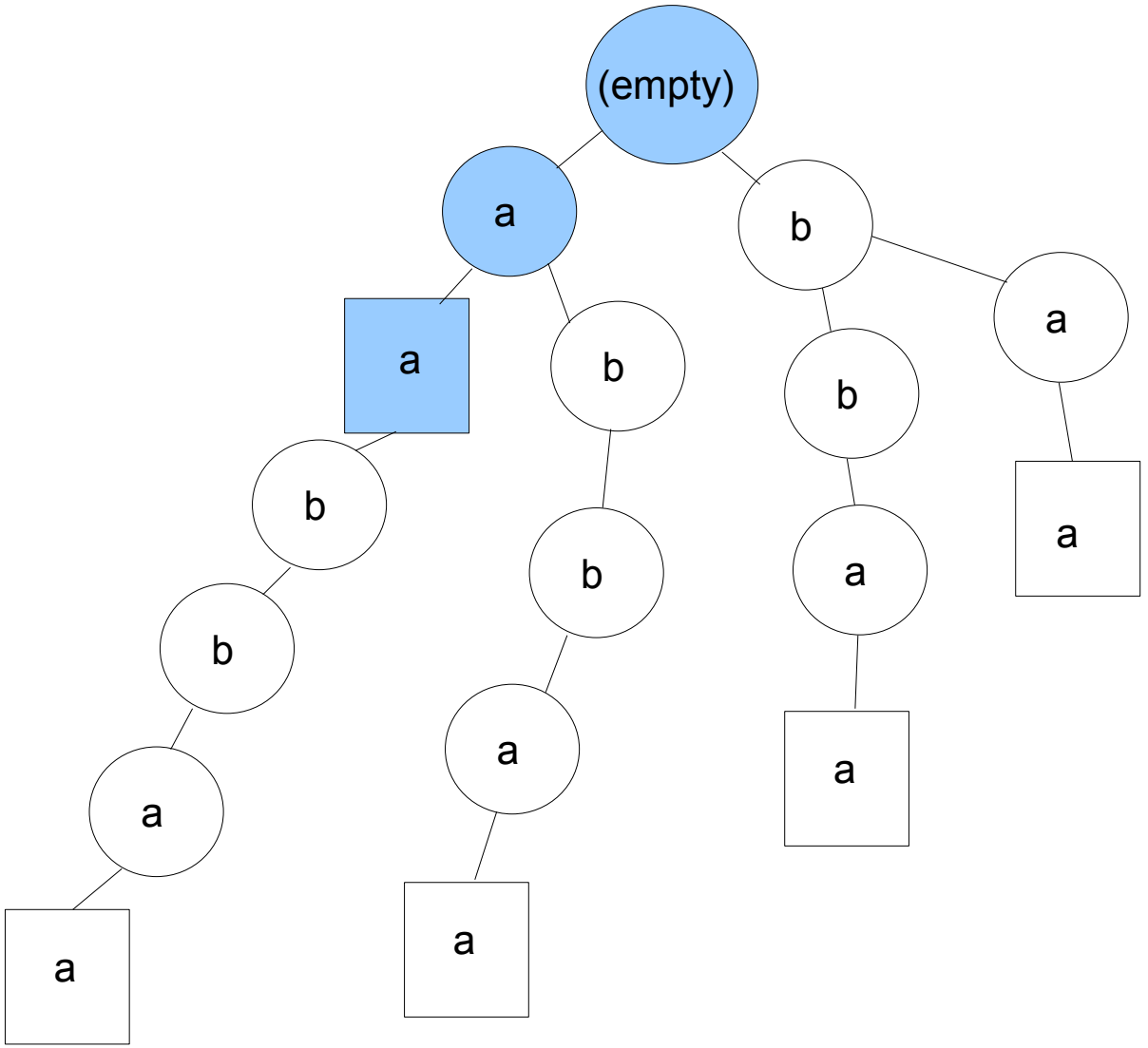
Suffix Tries



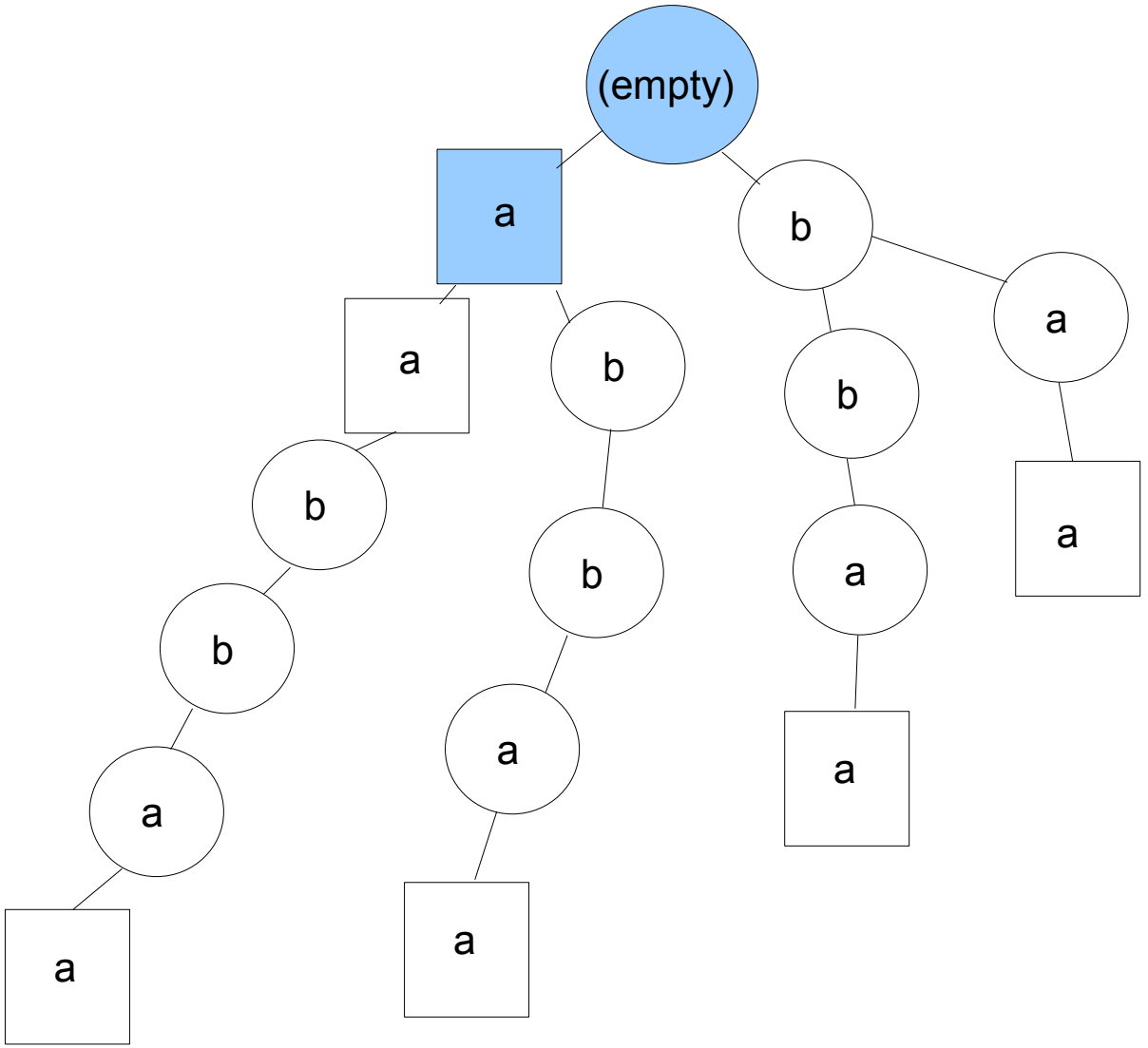
Suffix Tries



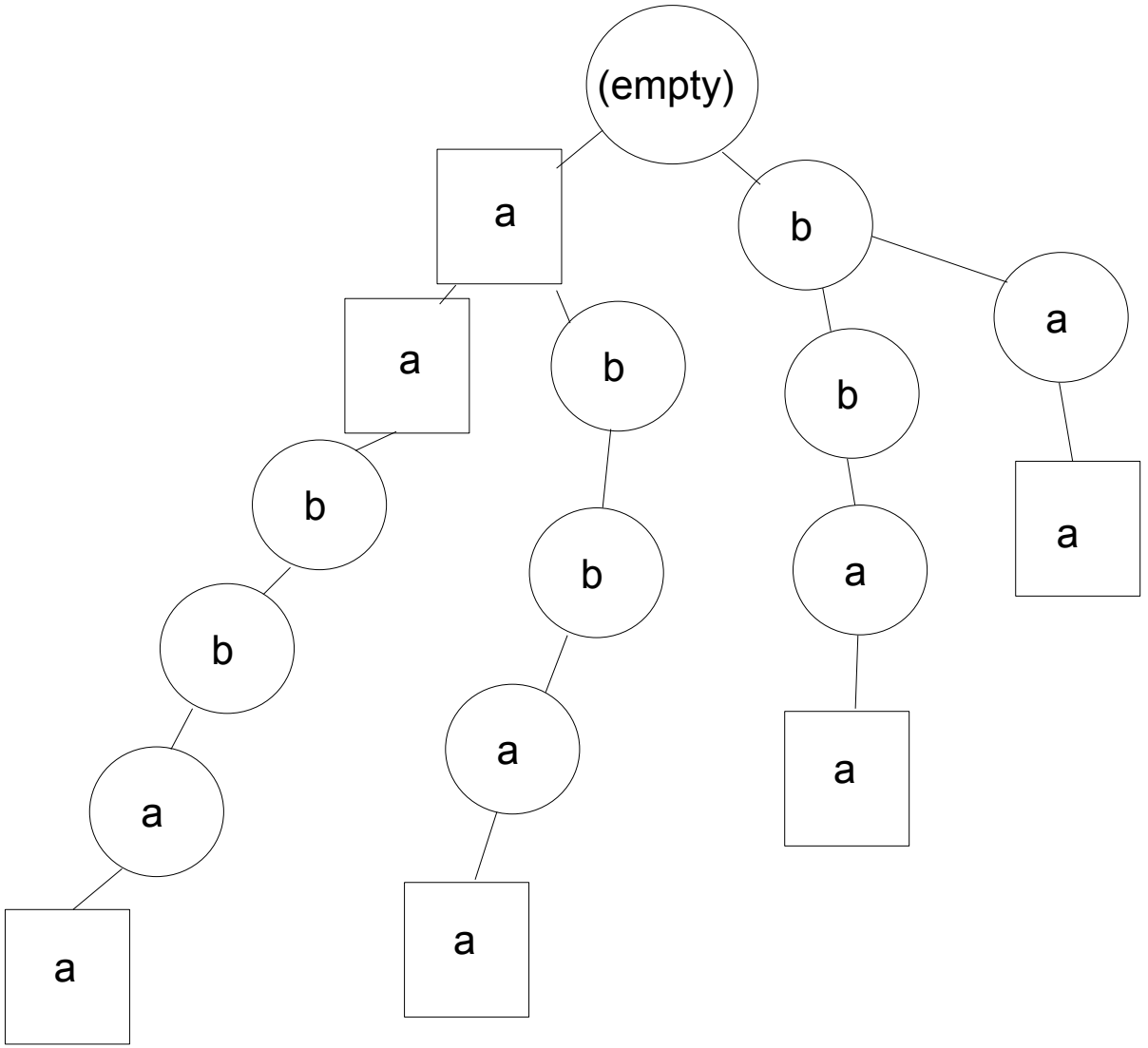
Suffix Tries



Suffix Tries



Suffix Tries



Suffix Array

a a b b a a

0. a a b b a a
1. a b b a a
2. b b a a
3. b a a
4. a a
5. a
6. (empty)

0. (empty)
1. a
2. a a
3. a a b b a a
4. a b b a a
5. b a a
6. b b a a