overview

- single-source shortest path
- string-matching
- brute-force string matching algorithm
- Knuth-Morris-Pratt pattern matching
- material
path

weighted and directed graph

path from $v_0$ to $v_n$:
list of vertices $p = (v_0, v_1, \ldots, v_n)$

weight of $p$:
sum of its edges $\sum_{i=1}^{n} w(v_{i-1}, v_i)$

shortest-path weight $\delta(u, v)$
$\min\{w(p) \mid p \text{ path from } u \text{ to } v\}$ if there is a path from $u$ to $v$
$\infty$ otherwise

shortest path from $u$ to $v$:
path with weight $\delta(u, v)$

negative weights

if on the path from $s$ to $v$ there is a negative weight-cycle
then distance from $s$ to $v$ is $-\infty$

either assume positive weights
or produce correct answer as long as no negative-weight cycle reachable

single source shortest-path algorithms

given a (weighted and directed) graph and a source node

give for every node $v$ a shortest path from the source to $v$
or: single pair shortest path
or: all pair shortest path

input: graph $G$ and start-vertex $s$ in $G$
set all distances $d$ except to $\infty$, except for $s.d$
so far no predecessor $\pi$

Algorithm initialize($G, s$):
for every $v \in V$
$v.d := \infty$
$v.\pi := \text{nil}$
$s.d = 0$
in $\Theta(|V|)$
can we improve ‘price’ of going to $v$ by going to $u$ and then use $(u, v)$?

input: nodes $u$ and $v$ and weight function $w$

Algorithm relax$(u, v, w)$:

$$\text{if } v.d > u.d + w(u, v) \text{ then}$$

$$v.d := u.d + w(u, v)$$

$$v.\pi := u$$

in $O(1)$

Dijkstra’s algorithm: time complexity

initialize in $\Theta(|V|)$

init $S$ in constant time

init $Q$: build priority queue

in while-loop $|V|$ times extract-min

in while-loop $|V|$ times update of $S$ (which is in constant time)

for-loop is executed in total (!) $|E|$ times with inside update key of $v$

crucial: how we implement the priority queue

Dijkstra’s algorithm

input: directed graph $G$ with positive weights $w$ and start vertex $s$ in $G$

Algorithm Dijkstra$(G, w, s)$:

initialize$(G, s)$

$S := \emptyset$

$Q := G.V$

while $Q \neq \emptyset$ do

$u := \text{extractMin}(Q)$

$S := S \cup \{u\}$

for each $v \in G.\text{Adj}[u]$ do

relax$(u, v, w)$

the set $S$ contains vertices for which the weight of a shortest path has been found

Dijkstra’s algorithm: time complexity

priority queue implemented as heap:

extract-min and update key in $O(\log |V|)$

algorithm in $O((|V| + |E|) \cdot \log |V|) = O(|E| \cdot \log |V|)$

priority queue implemented as array with $v$ at index $v$:

extract-min takes in the worst case $|V|$ steps

algorithm in $O(|V|^2 + |E|) = O(|V|^2)$
string

string: sequence of characters given as array \( P[1 \ldots m] \)

empty string: \( \epsilon \)

deexample: \( acaabca \) given as \( [a, c, a, a, b, c, a] \)

prefix: initial part; \( P[1 \ldots k] \) for \( 0 \leq k \leq m \)

deexample: \( acaab \) is prefix of \( acaabca \)

suffix: final part; \( P[k \ldots m] \) for \( 1 \leq k \leq m + 1 \)

deexample: \( bca \) is a suffix of \( acaabca \)

substring: subarray

string-matching problem

setting:
text \( T[1 \ldots n] \) and pattern \( P[1 \ldots m] \),
both arrays with characters from a finite alphabet \( \Sigma \)
valid shift:
\( P \) occurs at shift \( s \) if \( T[1 + s \ldots s + m] = P[1 \ldots m] \)
that is: \( P \) occurs beginning at position \( s + 1 \)
or: \( P \) is a substring of \( T \)

then: \( s \) is a valid shift for \( P \) and \( T \)

string matching problem: find all valid shifts for given \( T \) and \( P \)
string matching: example

does $P = \text{bra}$ occur in $T = \text{adacadabra}$?

brute-force algorithm: pseudo code

**input:** pattern $P[1\ldots m]$ and text $T[1\ldots n]$

**output:** shift of occurrence $s$ of all matches

Algorithm BruteForceMatch($T, P$):

1. $n := T.length$
2. $m := P.length$
3. for $s := 0$ to $n - m$ do
4.     $j := 1$
5.     while $j \leq m$ and $T[s + j] = P[j]$ do
6.         $j := j + 1$
7.     if $j = m + 1$ then
8.         print 'occurs with shift $s$'

brute-force algorithm: analysis

**worst-case example:**

$T = \text{aaaaaaaaaaa}$ en $P = \text{aaa}$

takes $(n - m + 1) \cdot m$ steps

**worst-case time complexity:**

in $O((n - m + 1) \cdot m)$

so if $m$ roughly half of $n$, then in $O(n^2)$

**pre-processing:**

nothing
overview

- single-source shortest path
- string-matching
- brute-force string matching algorithm
- Knuth-Morris-Pratt pattern matching
- material

Knuth-Morris-Pratt algorithm: idea

compare $P$ with $T$ from left to right

if mismatch $T[pos] \neq P[i]$: what is maximal shift?

```
. . a b a a b x . . . .
```

```
 a b a a b b
```

```
 a b a a b a
```

don’t do this again

continue here

KMP algorithm: failure or prefix function

$P = ababaca$

```
  i  1  2  3  4  5  6  7
  P[i] a b a b a c a
  f(i)  0  0  1  2  3  0  1
```

$f(i)$ tells us how many of the $i$ succeeded matches can be reused

Knuth-Morris-Pratt algorithm

Algorithm KMPMatch($T, P$):

```
i := 1
j := 1
while i \leq n do
  if $P[j] = T[i]$ then
    if $j = m$ then match
      i := i + 1
      j := j + 1
    else
      if $j > 1$ then
        j := f(j - 1) + 1
      else
        i := i + 1
  return fail
```
KMP algorithm: example

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>f(i)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Knuth-Morris-Pratt: time complexity

computing the failure or prefix function

$O(m)$ with $m$ length of pattern $P$

iteration:

the while-loop is traversed at most $2n$ times (asks analysis!)
hence the algorithm is in $O(n + m)$

know without proof

overview

- single-source shortest path
- string-matching
- brute-force string matching algorithm
- Knuth-Morris-Pratt pattern matching
- material

extra materiaal

- Dijkstra’s paper from 1959
- Dijkstra’s reflections on his work from 1959
- Knuth
- Morris
- Pratt
- grep