Lecture 12 — CS124

Dynamic Programming

All pair shortest paths

Dijkstra $O(V^2) \rightarrow O(V^3)$ for all pairs.

$D[i,j]$ = shortest path from i to j

$D_k[i,j]$ = shortest path from i to j only using vertices $1, \ldots, k$ as intermediate nodes

$D_0[i,j] = \infty$ if no edge (i,j)

d_{ij} = length of edge from i to j otherwise

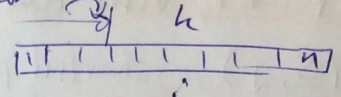
$V = \{1, \ldots, n\}$

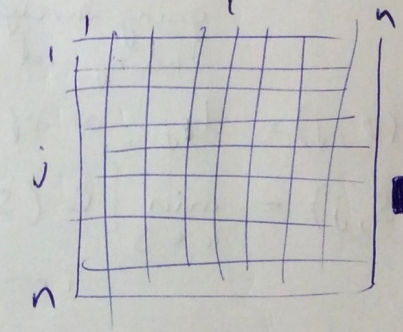
recurrence for $D_k[i,j] = \min\left[D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j] \right]$

case 1: $i \rightsquigarrow j$ (path does not pass through k)

case 2: $i \rightsquigarrow k \rightsquigarrow j$ (path passes through k)

keep two 2D arrays





for each k update 2D array

array $D = (d_{ij})$

```
for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
            D_k[i,j] = min[ D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j] ]
```

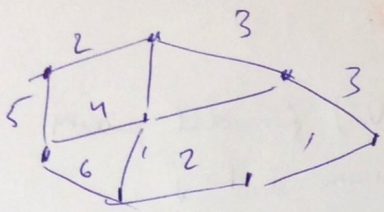
$O(V^3)$ runtime, not better than Dijkstra, but a different formulation

$O(V^2)$ space $\rightarrow$ upto $O(V)$

Traveling Salesman Problem

NP-complete

Vertices = cities



min cost tour:
start at 1, visit all cities and return to 1 at min cost

$(n-1)!$ paths

$O(n!)$ ← in each path compute cost across n vertices

Stirling for large n

$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

With DP we get $O(2^n n^2)$

start at 1, ~~end~~ at 1 — cycle — path

to cycle is hard to think recursively — easier to think recursively

reformulate the problem

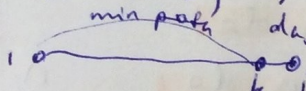
start at 1, visit all vertices

$C(S, j)$ = shortest path starting at 1, going through all vertices of S ending at j

rec

$C(\emptyset, j) = d_{1j}$, ∞ if no edge

$$C(S, j) = \min_{k \in S} \left[C(S - \{k\}, k) + d_{kj} \right]$$



which node is k? try all of them

iter

for all (S, j) in increasing size of S

$[1, j \notin S]$

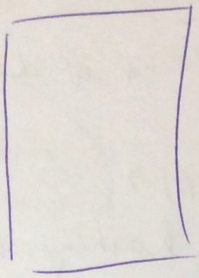
$C(S, j) = \min_{k \in S} \ldots$

precomputed

from path to cycle ~~one~~ more min operation

$$OPT = \min_k \left[C(\{2, \ldots n\} - \{k\}, k) + d_{k1} \right]$$

array



subsets
$S - \{j\}$
but keep
all
subsets of
S for simplicity

$O(2^4 n)$ ← memory cells

min op for each cell
$O(n)$

$\Rightarrow O(2^4 n^2)$ ← runtime

My Summary

DP for strings:
recurrence on string, ↑ reconstruction
cost function ↑ edit dist.

DP for graphs:
recurrence on sequence of nodes ↑ all pairs shortest paths
(subsets of nodes ↑ TSP

Pattern matching — length

P = pattern — $|P|$
D = document — $|D|$
find pattern in doc.

think as base 10 numbers

Baseline
char by char move pattern
$O(P \cdot D)$ ← problem for large patterns

Hashing $O(D)$

Hash pattern (eg. 16 bits)

Hash blocks of doc and compare

① why isn't hashing O(|P|)? → convenient "Hash"
↳ can we make hashing O(1)?

② False positives? → bound FPs

Pick a prime p. → later discussed, assume as subroutine

hash: number mod p

① Rolling

17935

6 38617935 → ....

17935 mod 251 = 114

38617 mod 251 = 107

38617 mod 251

p = 251

$N \to N'$

a ~~~ ~~~ b

$N' = (N - 10^{|P|-1} \cdot a) \cdot 10 + b$

mod p

$N' = (N - (10^{|P|-1} \bmod p) \cdot a) \cdot 10 + b) \bmod p$

const precomputable

② bound FP rate
pick random prime

$\pi(x)$ = # of primes $\le x$

$\pi(x) \sim \frac{x}{\ln x}$

10^{100} $\pi(10^{100}) \sim \frac{10^{100}}{100 \ln 10}$

fraction of primes $\frac{1}{250}$

const. time Fingerprinting

Single FP coll

consider FP coll (a non-pallel is 0 mod p)

P

D^*

$P - D^* = 0 \bmod p$

D^* chunks corresponding to P

$|P - D^*| \le$ [redacted] $10^{|P|}$

each size $10^{|P|}$

? how many primes can get it to 0 mod p

↳ the number must be product of these primes, at least (each prime can contribute as multiple)

$\Leftarrow$ Suppose primes $p_1, p_2 \dots p_j$ all divide X, then

$2^j \le \prod_{i=1}^{j} p_i \le X$

each prime is at least 2

$j \le \log_2 X$ (take $\log_2$ on both sides)

$$\#\text{ bad Primes} \le \log_2 x = \log_2 10^{|P|} = |P| \log_2 10$$

(# of primes that get a non-matching pattern to 0 mod p)

$$\#\text{ primes (hashing into 64 bits)} = \pi(2^{64})$$

$$P[\text{get a bad prime at a single "non-pattern" hashing}] = \frac{|P| \log_2 10}{\pi(2^{64})}$$

(single FP coin)

$$P[\text{FP in doc}] = \frac{|D||P| \log_2 10}{\pi(2^{64})}$$

(FP across all "non-pattern" hashings in a doc)

with k hashes

(increases work by a constant, but lowers FP rate exponentially)

$$P[\text{FP in doc}] = \left[\frac{|D||P| \log_2 10}{\pi(2^{64})}\right]^k$$