Lecture 10

More Dynamic Programming

Travelling Salesperson Problem (TSP)

Graph (complete weighted)

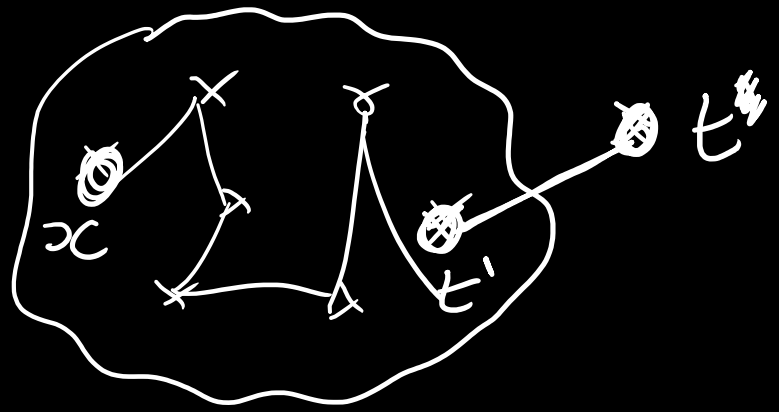
Goal: Visit every vertex once and only once
(find shortest distance)
(and come back to start)

NP-hard !!

Naive algorithm: Try all $n!$ orders $\rightarrow O(n! \cdot n)$

Substructure

Attempt: Subsets of vertices? ↙



Add more information? Arbitrary start point x

Consider paths that end at a particular vertex t

$C(S,t)$ = Min-cost path starts at x, goes through all $v \in S$, ends at t

x: subset, t: end point

Recurrence

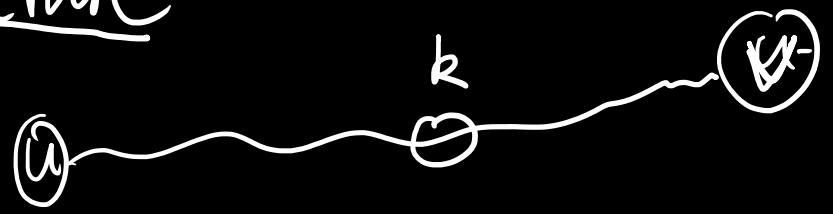
$$C(S,t) = \begin{cases} w(x,t) & \text{if } S = \{x,t\} \\ \min_{t' \in S,\ t' \neq x,t} C(S - \{t\}, t') + w(t',t) \end{cases}$$

Analysis. $O(2^n \cdot n)$ subproblems. $O(n)$ time per sub. $O(2^n \cdot n^2)$

All-pairs shortest paths

Directed weighted graph G.

Optimal substructure



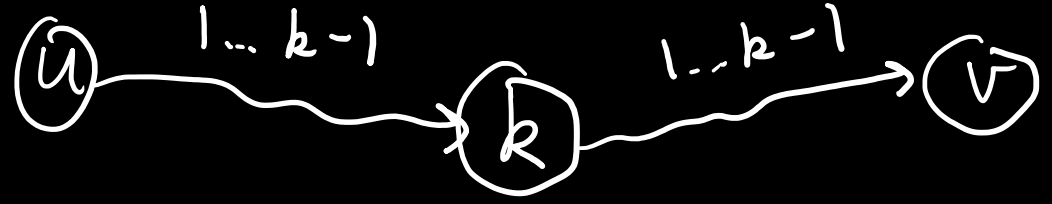
Subproblems

$D[u][v][k]$ = distance from u to v if the path uses only $\{1, 2, \ldots k\}$ as intermediate

Recurrence

don't use k ↓ do use k ↓

$$D[u][v][k] = \min\{D[u][v][k-1],\ D[u][k][k-1] + D[k][v][k-1]\}$$



$$DP[u][v][0] = \begin{cases} 0 & \text{if } u = v \\ w(u,v) & \text{if } (u,v) \in E \\ \infty & \text{otherwise} \end{cases}$$

Analysis: $O(n^3)$ problems. $O(1)$ per. $O(n^3)$ time.

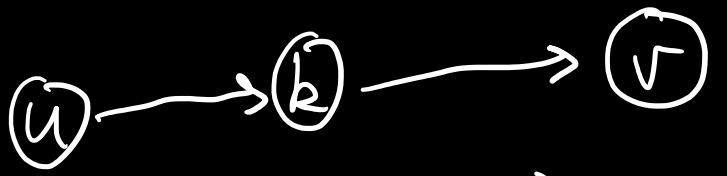
Optimize Space (Floyd-Warshall)

DP[u][v][k] required DP[..][..][k-1]

Only store current & previous value for k

2nd idea: Forget about k.

```
for k = 1 to n do
    for u = 1 to n do
        for v = 1 to n do
            DP[u][v] = min(DP[u][v], DP[u][k] + DP[k][v])
```



$O(n^2)$ space

Longest Increasing Subsequence

Given $a_1\ a_2\ \ldots\ a_n$.

Increasing subsequence is $a_{i_1} < a_{i_2} < \ldots < a_{i_k}$

Goal Find longest one

$1 \times\times\times\ \ 2 \times\times\ \ 5 \times\ \times 10\ \ \times\times 12$

Optimal Substructure

1 x x x 2 x x x 5 x x 10 x x 12 x

Subproblems

$LIS[i]$ = length of LIS of $a_1, \ldots a_i$ that ends at a_i

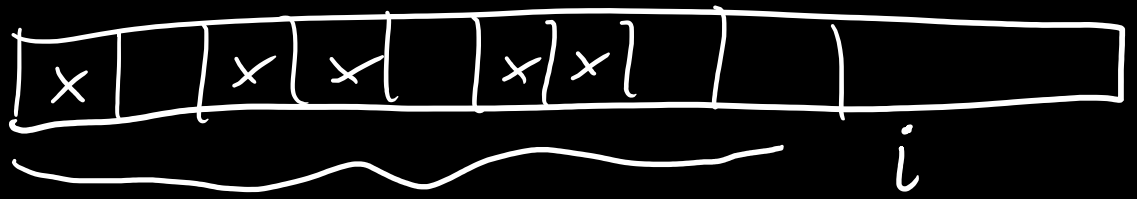
Recurrence

$$LIS[i] = \begin{cases} 0 & \text{if } i = 0 \\ 1 + \max\limits_{\substack{0 \le j < i \\ a_j < a_i}} LIS[j] \end{cases}$$

Analysis

$O(n^2)$ time

Q: Can we do better?



Idea: What if a was sorted?

```
n := size(a)                                   O(n log n) time
b := sorted(a)
LIS : SegTree = (array(int))(n+1, 0)
for i in 0 to n-1 do {
    rank = binary_search(b, a[i])
    LIS.Assign(rank, 1 + LIS.RangeMax(0, rank))
}
return LIS.RangeMax(0, n+1)
```