# CS 420: Advanced Algorithm Design and Analysis
# Spring 2015 – Lecture 13

Department of Computer Science
University of British Columbia

February 24, 2015

# Announcements

Assignments...

- Sample solutions to Asst 4 (and Midterm I) have been posted
- Asst5...due Thursday

Midterm II...

- Q/A session...next Tuesday (March 03); 5:30-7:00; DMPT 310
- Exam...Wednesday (March 04); 5:30-7:00; DMPT 310
- ...on material up to and including this Thursday's class

Readings...

- minimum-cost path problems [Erickson, Chapt 21, 22; Cormen+, Chapt 25 26; ...]
- edit-distance [Erickson, Chapt. 5.5, 6]

# Last classes...

## Graph algorithms

- path existence (connectivity) cont.
- optimization (minimum cost/shortest paths)
  - minimum length and minimum cost paths
  - properties of minimum cost paths
  - properties of edge relaxation
  - algorithms for *single source* min-cost paths
    - Bellman Ford
    - Dijkstra
    - comparison of Bellman-Ford and Dijkstra
  - all-pairs min-cost paths
    - Johnson's algorithm, using *edge re-weighting*

# Today...

## Min-cost path problems

- all-pairs of endpoints (cont.)
  - algorithms for dense graphs, using dynamic programming

## Edit distance problems

- dynamic programming solutions
- reformulation as (single source, single destination) min-cost path problem

# Coming up...

## Min-cost path problems

- issues related to real-world constraints
  - robustness (failure tolerance)
  - single-source single destination queries
- issues motion planning (continuous path problems)
  - paths on terrains
  - obstacle avoidance
  - curvature constraints

# Algorithms for all-pairs min-cost paths

Problem: Given $G$, determine $\delta(u, v)$, for all $u, v \in V$.

Johnson's algorithm (or repeated Dijkstra) is particularly efficient ($O(n \cdot (m + n \lg n))$ for *sparse* graphs (graphs with $m << n^2$). What about dense graphs?

There are two straightforward *dynamic programming* solutions in this case, both of which involve solving the problem with progressively relaxed *structural constraints*.

1. path-length constraints
2. intermediate-vertex constraints

In both cases, we will assume the vertices are labeled $1, 2, \ldots, n$.

# All-pairs min-cost paths, by dynamic programming

## path-length constraint relaxation

Let $d_{ij}^{(r)}$ denote the minimum cost path from vertex $i$ to vertex $j$, *containing at most r edges*. Then,

1. $d_{ij}^{(0)} = 0$, if $i = j$ (and $\infty$, otherwise), and $d_{ij}^{(1)} = c(i,j)$
2. $d_{ij}^{(n-1)} = \delta(i,j)$

So, it suffices to see how to construct $d_{ij}^{(r)}$ values given $d_{ij}^{(r-1)}$ values.

*Key observation:*

$$\begin{aligned} d_{ij}^{(r)} &= \min\{d_{ij}^{(r-1)}, \min_{1\le k\le n}\{d_{ik}^{(r-1)} + c(k,j)\}\} \\ &= \min\{d_{ij}^{(r-1)}, \min_{1\le k\le n}\{d_{ik}^{(r-1)} + d_{kj}^{(1)}\}\} \\ &= \min_{1\le k\le n}\{d_{ik}^{(r-1)} + d_{kj}^{(1)}\} \end{aligned}$$

# All-pairs min-cost paths, by dynamic programming

## path-length constraint relaxation (cont.)

This recurrence suggests the following dynamic programming solution:

construct $d_{**}^{(0)}$ and $d_{**}^{(1)}$
**for** $r = 2$ to $n - 1$ **do**
    **for** $i = 1$ to $n$ **do**
        **for** $j = 1$ to $n$ **do**
            $d_{ij}^{(r)} \leftarrow \min_{1 \leq k \leq n}\{d_{ik}^{(r-1)} + d_{kj}^{(1)}\}$
        **end for**
    **end for**
**end for**

Cost is $O(n^4)$ in total.

# All-pairs min-cost paths, by dynamic programming

## path-length constraint relaxation (cont.)

But...

1. $d_{ij}^{(r)} = d_{ij}^{(n-1)}$, for all $r \geq n-1$; and
2. $d_{ij}^{(2r)} = \min_{1 \leq k \leq n}\{d_{ik}^{(r)} + d_{kj}^{(r)}\}$

So we can compute:

---

construct $d_{**}^{(0)}$ and $d_{**}^{(1)}$
**for** $t = 1$ to $\lceil \lg n \rceil$ **do**
    **for** $i = 1$ to $n$ **do**
        **for** $j = 1$ to $n$ **do**
            $d_{ij}^{(2^t)} = \min_{1 \leq k \leq n}\{d_{ik}^{(2^{t-1})} + d_{kj}^{(2^{t-1})}\}$
        **end for**
    **end for**
**end for**

---

Cost is reduced to $O(n^3 \lg n)$ in total.

# All-pairs min-cost paths, by dynamic programming

## intermediate-vertex constraint relaxation

Let $\hat{d}_{ij}^{(k)}$ denote the minimum cost path from vertex $i$ to vertex $j$, *using intermediate vertices in* $\{1, \ldots, k\}$. Then,

1. $\hat{d}_{ij}^{(0)} = c(i, j)$
2. $\hat{d}_{ij}^{(n)} = \delta(i, j)$

So, again it suffices to see how to construct $\hat{d}_{ij}^{(k)}$ values given $\hat{d}_{ij}^{(k-1)}$ values.
*Key observation:*

$$\hat{d}_{ij}^{(k)} = \min\{\hat{d}_{ij}^{(k-1)},\ \hat{d}_{ik}^{(k-1)} + \hat{d}_{kj}^{(k-1)}\}$$

# All-pairs min-cost paths, by dynamic programming

## intermediate-vertex constraint relaxation (cont.)

This recurrence suggests the following dynamic programming solution:

construct $\hat{d}_{**}^{(0)}$
**for** $k = 1$ to $n$ **do**
  **for** $i = 1$ to $n$ **do**
    **for** $j = 1$ to $n$ **do**
      $\hat{d}_{ij}^{(k)} \leftarrow \min\{\hat{d}_{ij}^{(k-1)},\ \hat{d}_{ik}^{(k-1)} + \hat{d}_{kj}^{(k-1)}\}$
    **end for**
  **end for**
**end for**

Cost is $O(n^3)$ in total.

# All-pairs min-cost paths, by dynamic programming

Note:

- space requirements
- path reconstruction
- similarities with Bellman-Ford…distributed implementation

# Edit distance Problem

Given two strings (over some fixed alphabet):

$$X = x_1 x_2 \ldots x_n$$
$$Y = y_1 y_2 \ldots y_m$$

we want to *transform* $X$ to $Y$ with the fewest primitive operations:

- *delete* a symbol: cost$(x_i, -)$
- *insert* a symbol: cost$(-, y_j)$
- *replace* a symbol: cost$(x_i, y_j)$

Small total cost (edit distance) measures similarity of $X$ and $Y$
(e.g. spell checking, sequence alignment)

# Edit distance Problem

optimal substructure:

Either (i) $x_n$ is matched with $y_m$, (ii) $x_n$ is matched with $-$ (deleted), or (iii) $y_m$ is matched with $-$ (inserted).
Hence, if $\mathrm{ED}[i,j]$ denotes the edit distance of $x_1 \ldots x_i$ to $y_1 \ldots y_j$,:

$$\mathrm{ED}[i,j] = \min \left\{ \begin{array}{l} \mathrm{ED}[i-1,j-1] + \mathrm{cost}(x_i, y_j) \\ \mathrm{ED}[i-1,j] + \mathrm{cost}(x_i, -) \\ \mathrm{ED}[i,j-1] + \mathrm{cost}(-, y_j) \end{array} \right\}$$

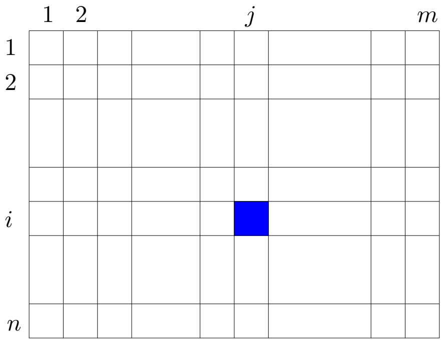
dynamic programming solution:

Knowing all cost values, we can compute $\mathrm{ED}[i,j]$ values in increasing order of $i+j$ (or by row, or column)

# Edit Distance Matrix

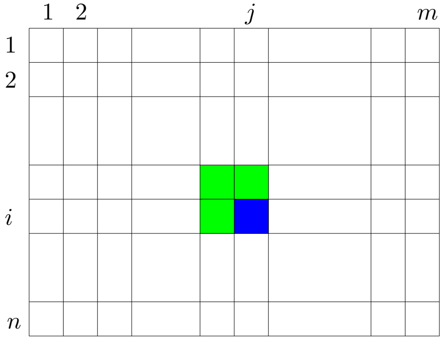
| | $1$ | $2$ | | | | $j$ | | | $m$ |
|---|---|---|---|---|---|---|---|---|---|
| $1$ | | | | | | | | | |
| $2$ | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| $i$ | | | | | | | | | |
| | | | | | | | | | |
| $n$ | | | | | | | | | |

# Edit Distance Matrix




**New entry** depends on only .

# Edit Distance Matrix



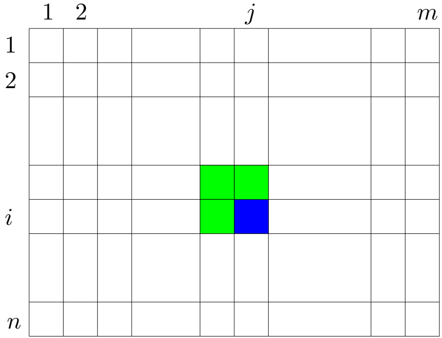

**New entry** depends on only **three neighbouring entries**.

# Edit distance Problem

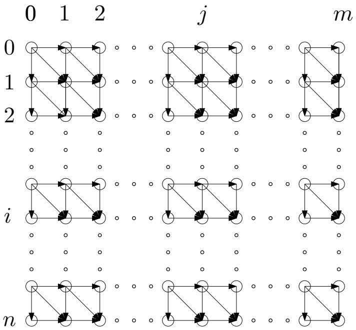
dynamic programming solution:

1. Total cost is $O(nm)$
   - each ED-table entry is computed in $O(1)$ time
2. Space can be reduced to $O(n+m)$
   - it suffices to keep only *two* active columns of ED-table
3. Optimal edit script can be reproduced efficiently
   - by divide and conquer

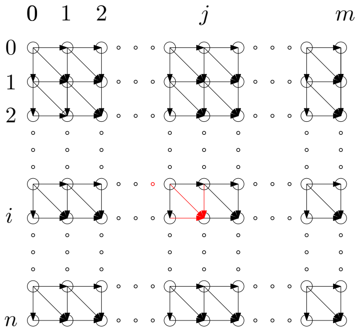
# Edit Distance Matrix



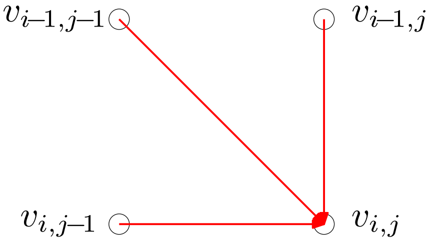
# Edit Distance Graph



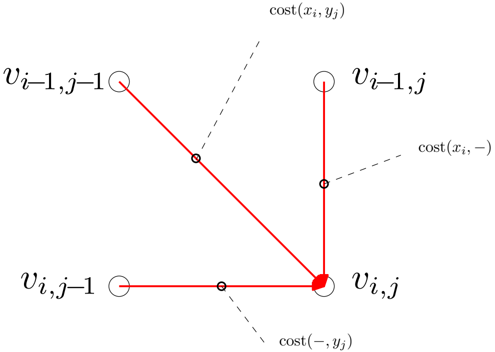
# Edit Distance Graph



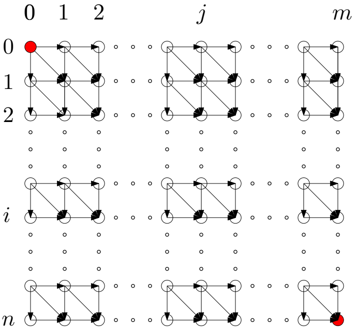
# Edit Distance Graph



# Edit Distance Graph

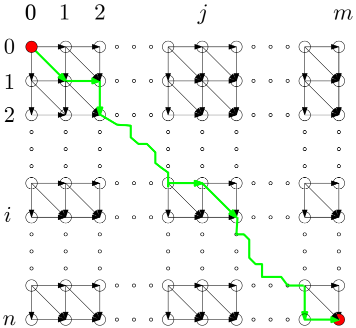


# Edit Distance Graph




We want to find the min-cost path from $v_{0,0}$ to $v_{n,m}$.

# Edit Distance Graph




We want to find the min-cost path from $v_{0,0}$ to $v_{n,m}$.