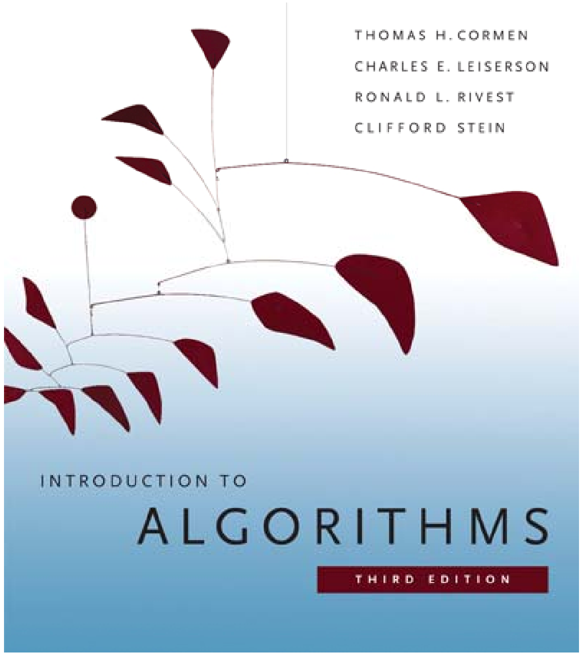
6.006- *Introduction to Algorithms*

Lecture 19



Dynamic Programming II

Prof. Manolis Kellis

CLRS 15.4, 25.1, 25.2

Course VI - 6.006 – Module VI – This is it

Unit	Pset	Week	Date		Lecture (Tuesdays and Thursdays)		Recitation (Wed and Fri)
Intro	PS1	1	Tue Feb 01	1	Introduction and Document Distance	1	Python and Asymptotic Complexity
Binary	Out: 2/1		Thu Feb 03	2	Peak Finding Problem	2	Peak Finding correctness & analysis
Search	Due: Mon 2/14	2	Tue Feb 08	3	Scheduling and Binary Search Trees	3	Binary Search Tree Operations
Trees	HW lab: Sun 2/13		Thu Feb 10	4	Balanced Binary Search Trees	4	Rotations and AVL tree deletions
Hashing	PS2 Out: 2/15	3	Tue Feb 15	5	Hashing I : Chaining, Hash Functions	5	Hash recipes, collisions, Python dicts
	Due: Mon 2/28		Thu Feb 17	6	Hashing II : Table Doubling, Rolling Hash	6	Probability review, Pattern matching
	HW lab:Sun 2/27	4	Tue Feb 22	-	President's Day - Monday Schedule - No Class	-	No recitation
			Thu Feb 24	7	Hashing III : Open Addressing	7	Universal Hashing, Perfect Hashing
Sorting	PS3. Out: 3/1	5	Tue Mar 01	8	Sorting I : Insertion & Merge Sort, Master Theorem	8	Proof of Master Theorem, Examples
	Due: Mon 3/7		Thu Mar 03	9	Sorting II : Heaps	9	Heap Operations
	HW lab: Sun 3/6	6	Tue Mar 08	10	Sorting III: Lower Bounds, Counting Sort, Radix Sort	10	Models of computation
			Wed Mar 09	Q1	Quiz 1 in class at 7:30pm. Covers L1-R10. Review Session on Tue 3/8 at 7:30pm.		
Graphs	PS4. Out: 3/10		Thu Mar 10	11	Searching I: Graph Representation, Depth-1st Search	11	Strongly connected components
and	Due: Fri 3/18	7	Tue Mar 15	12	Searching II: Breadth-1st Search, Topological Sort	12	Rubik's Cube Solving
Search	HW lab:W 3/16		Thu Mar 17	13	Searching III: Games, Network properties, Motifs	13	Subgraph isomorphism
Shortest	PS5	8	Tue Mar 29	14	Shortest Paths I: Introduction, Bellman-Ford	14	Relaxati
Paths	Out: 3/29		Thu Mar 31	15	Shortest Paths II: Bellman-Ford, DAGs	15	Shortest
	Due: Mon 4/11	9	Tue Apr 05	16	Shortest Paths III: Dijkstra	16	Speedin
	HW lab:Sun 4/10		Thu Apr 07	17	Graph applications, Genome Assembly	17	Euler To
Dynamic	PS6	10	Tue Apr 12	18	DP I: Memoization, Fibonacci, Crazy Eights	18	Limits of dynamic programming
Program	Out: Tue 4/12		Wed Apr 13	Q2	Quiz 2 in class at 7:30pm. Covers L11-R17. Review Session on Tue 4/13 at 7:30pm.		
ming	Due: Fri 4/29		Thu Apr 14	19	DP II: Shortest Paths, Genome sequence alignment	19	Edit Distance, LCS, cost functions
	HW lab:W 4/27	11	Tue Apr 19	-	Patriot's Day - Monday and Tuesday Off	-	No recitation
			Thu Apr 21	20	DP III: Text Justification, Knapsack	20	Saving Princess Peach
		12	Tue Apr 26	21	DP IV: Piano Fingering, Vertex Cover, Structured DP	21	Phylogeny
Numbers	PS7 out Thu4/28		Thu Apr 28	22	Numerics I - Computing on large numbers	22	Models of computation return!
Pictures	Due: Fri 5/6	13	Tue May 3	23	Numerics II - Iterative algorithms, Newton's method	23	Computing the nth digit of π
(NP)	HW lab: Wed 5/4		Thu May 5	24	Geometry: Line sweep, Convex Hull	24	Closest pair
		14	Tue May 10	25	Complexity classes, and reductions	25	Undecidability of Life
Beyond			Thu May 12	26	Research Directions (15 mins each) + related classes		
		15	Finals week	Q3	Final exam is cumulative L1-L26. Emphasis on L18-L26. Review Session on Fri 5/13 at 3pm		

Dynamic Programming

Dynamic Programming

- Optimization technique, widely applicable
  - ➢Optimal substructure ➢Overlapping subproblems
- Tuesday: Simple examples, alignment
  - – Fibonacci: top-down vs. bottom-up
  - – Crazy Eights: one-dimensional optimization
- Today: More DP
  - – Alignment: Edit distance, molecular evolution
  - – Back to paths: All Pairs Shortest Paths DP1,DP2
- Next week:
  - – Knapsack (shopping cart) problem
  - – Text Justification
  - – Structured DP: Vertex Cover on trees, phylogeny

Today: Dynamic programming II

- Optimal sub-structure, repeated subproblems
- Review: Simple DP problems
  - Fibonacci numbers: Top-down vs. bottom-up
  - Crazy Eights: One-dimensional optimization
- LCS, Edit Distance, Sequence alignment
  - Two-dimensional optimization: Matrix/path duality
  - Setting up the recurrence, Fill Matrix, Traceback
- All pairs shortest paths (naïve: 2^n. n*BelFo: n^4)
  - Representing solutions. Two ways to set up DP
  - Matrix multiplication: $n^3 \lg n$. Floyd-Warshall: n^3

Hallmarks of optimization problems

Greedy algorithms	**Dynamic Programming**

1. Optimal substructure

An optimal solution to a problem (instance) contains optimal solutions to subproblems.

2. Overlapping subproblems

A recursive solution contains a "small" number of distinct subproblems repeated many times.

Greedy algorithms	Dynamic Programming
3. Greedy choice property *Locally optimal choices lead to globally optimal solution*	***Greedy Choice is not possible*** *Globally optimal solution requires trace back through many choices*

1. Fibonacci Computation

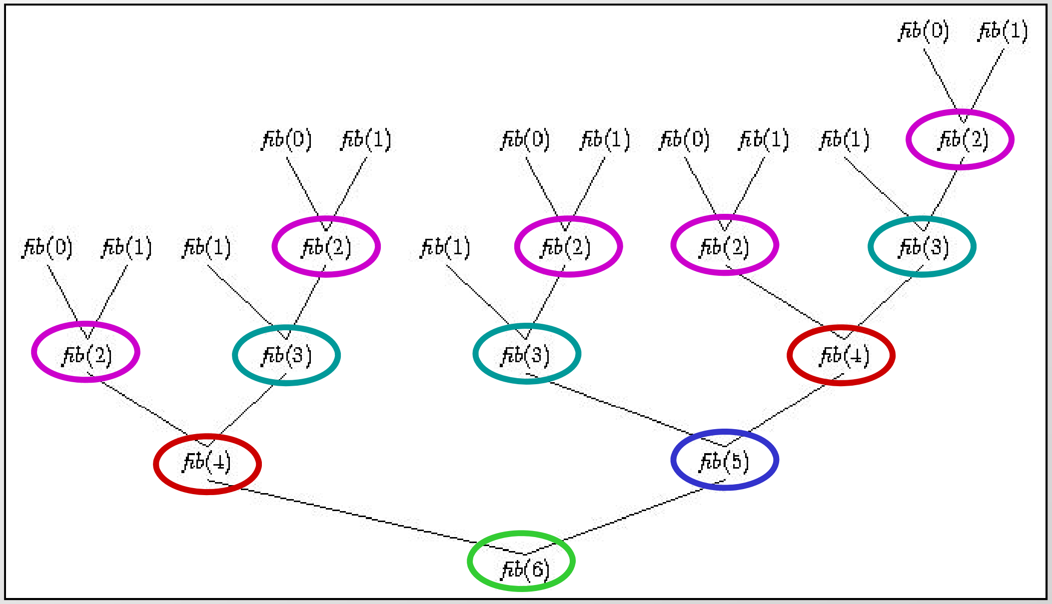
(not really an optimization problem,
but similar intuition applies)

Computing Fibonacci numbers: Top down

- Fibonacci numbers are defined recursively:
  - Python code

```
def fibonacci(n):
    if n==1 or n==2: return 1
    return fibonacci(n-1) + fibonacci(n-2)
```

- Goal: Compute n^{th} Fibonacci number.
  - F(0)=1, F(1)=1, F(n)=F(n-1)+F(n-2)
  - 1,1,2,3,5,8,13,21,34,55,89,144,233,377,…
- Analysis:
  - T(n) = T(n-1) + T(n-2) = (…) = $O(2^n)$



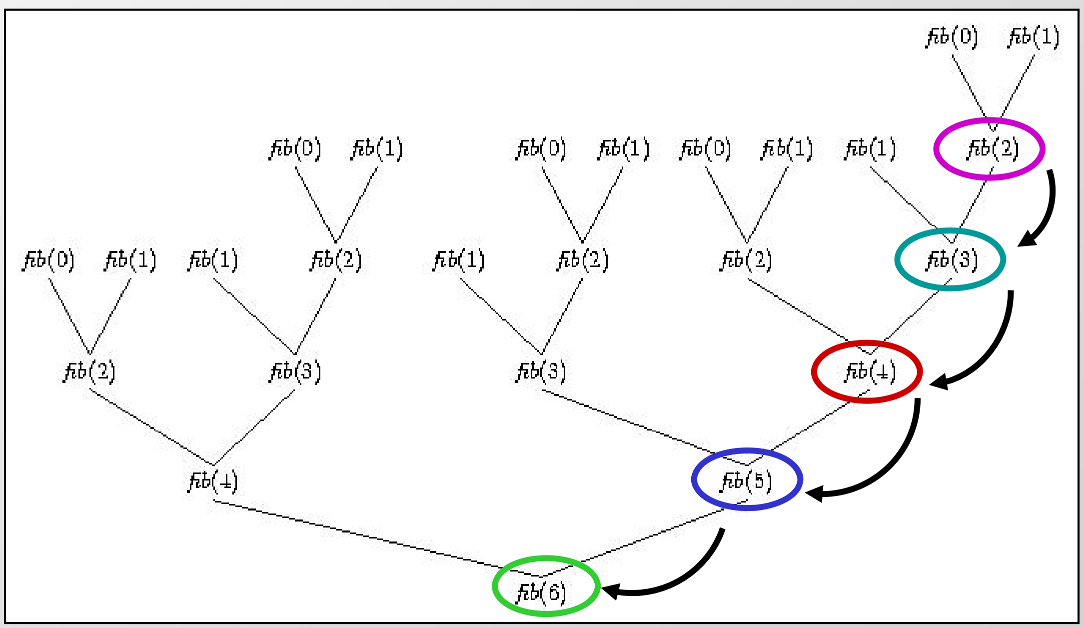
Computing Fibonacci numbers: Bottom up

- Top-down approach
  - Python code

fib_table	
F[1]	1
F[2]	1
F[3]	2
F[4]	3
F[5]	5
F[6]	8
F[7]	13
F[8]	21
F[9]	34
F[10]	55
F[11]	89
F[12]	?

```
def fibonacci(n):
        fib_table[1] = 1
        fib_table[2] = 1
        for i in range(3,n+1):
          fib_table[i] = fib_table[i-1]+fib_table[i-2]
        return fib_table[n]
```

  - Analysis: T(n) = O(n)

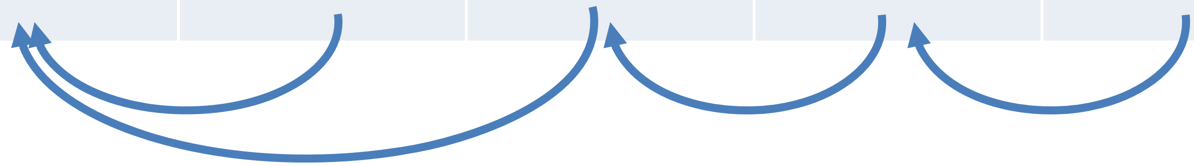


2. Crazy Eights

One-dimensional Optimization

Crazy8: Example computation

	i=1	i=2	i=3	i=4	i=5
c[i]	7♣	7♥	K♣	K♠	8♥
max score[i]	1	2	2	3	4



Input: a sequence of cards c[0]…c[n-1].
Goal: find the longest "trick subsequence" $c[i_1]\ldots c[i_k]$, where $i_1 < i_2 < \ldots < i_k$.

Rules:
- **same rank**
- **or same suit**
- **or one is an 8**

DP solution:
- Bottom-up solving of all tricks ending in ***i***
- Re-use computation by saving solutions
- Remember optimal choice and trace back

Why DP applies to Crazy Eights?

Optimal substructure:

- Optimal trick that uses card i must contain optimal trick that ends in card i.
- Proof (cut-and-paste argument): If not the case, replace sub-optimal trick ending in i with better trick ending in i, leading to better score overall
- Contradiction: original trick was supposedly 'optimal'

Overlapping sub-problems:

- To compute trick ending at i=5, need i=4 and i=3 and i=2 and i=1
- To compute trick ending at i=4, need i=3, i=2, i=1
- etc… ➔ naïve T(n)=T(n-1)+T(n-2)+T(n-3)… ➔ naïve T(n) is exponential in n

- However, only a small number of distinct subproblems exists

	i=1	i=2	i=3	i=4	i=5
c[i]	7♣	7♥	K♣	K♠	8♥
max score[i]	1	2	2	3	4

Dynamic Programming for Crazy Eights

- Setting up dynamic programming
  1. Find ‘matrix’ parameterization
     - One-dimensional array
  2. Make sure sub-problem space is finite! (not exponential)
     - Indeed, just one-dimensional array
  3. Traversal order: sub-results ready when you need them
     - Left-to-right ensures this
  4. Recursion formula: larger problems = F(subparts)
     - Scan entire sub-array completed so far O(n) each step
  5. Remember choices: typically F() includes min() or max()
     - Pointer back to the entry that gave us optimal choice
- Then start computing
  1. Systematically fill in table of results, find optimal score
  2. Trace-back from optimal score, find optimal solution

Today: Dynamic programming II

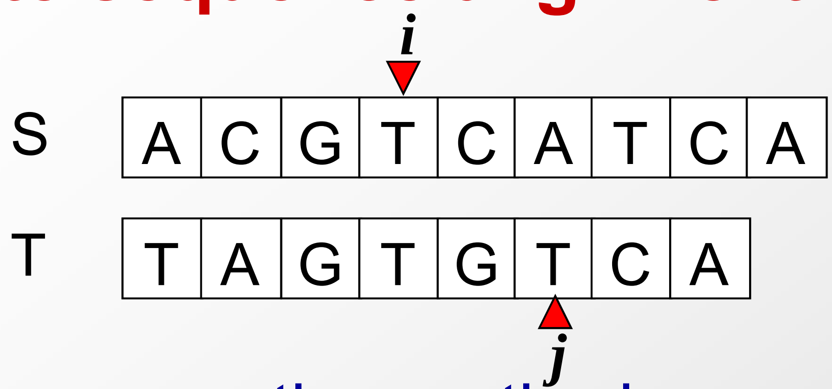
- Optimal sub-structure, repeated subproblems
- Review: Simple DP problems
  - Fibonacci numbers: Top-down vs. bottom-up
  - Crazy Eights: One-dimensional optimization
- LCS, Edit Distance, Sequence alignment
  - Two-dimensional optimization: Matrix/path duality
  - Setting up the recurrence, Fill Matrix, Traceback
- All pairs shortest paths (naïve: 2^n. n*BelFo: n^4)
  - Representing solutions. Two ways to set up DP
  - Matrix multiplication: $n^3 \lg n$. Floyd-Warshall: n^3

3. Sequence Alignment

(aka. Edit Distance, aka. LCS,
Longest common sub**sequence**)

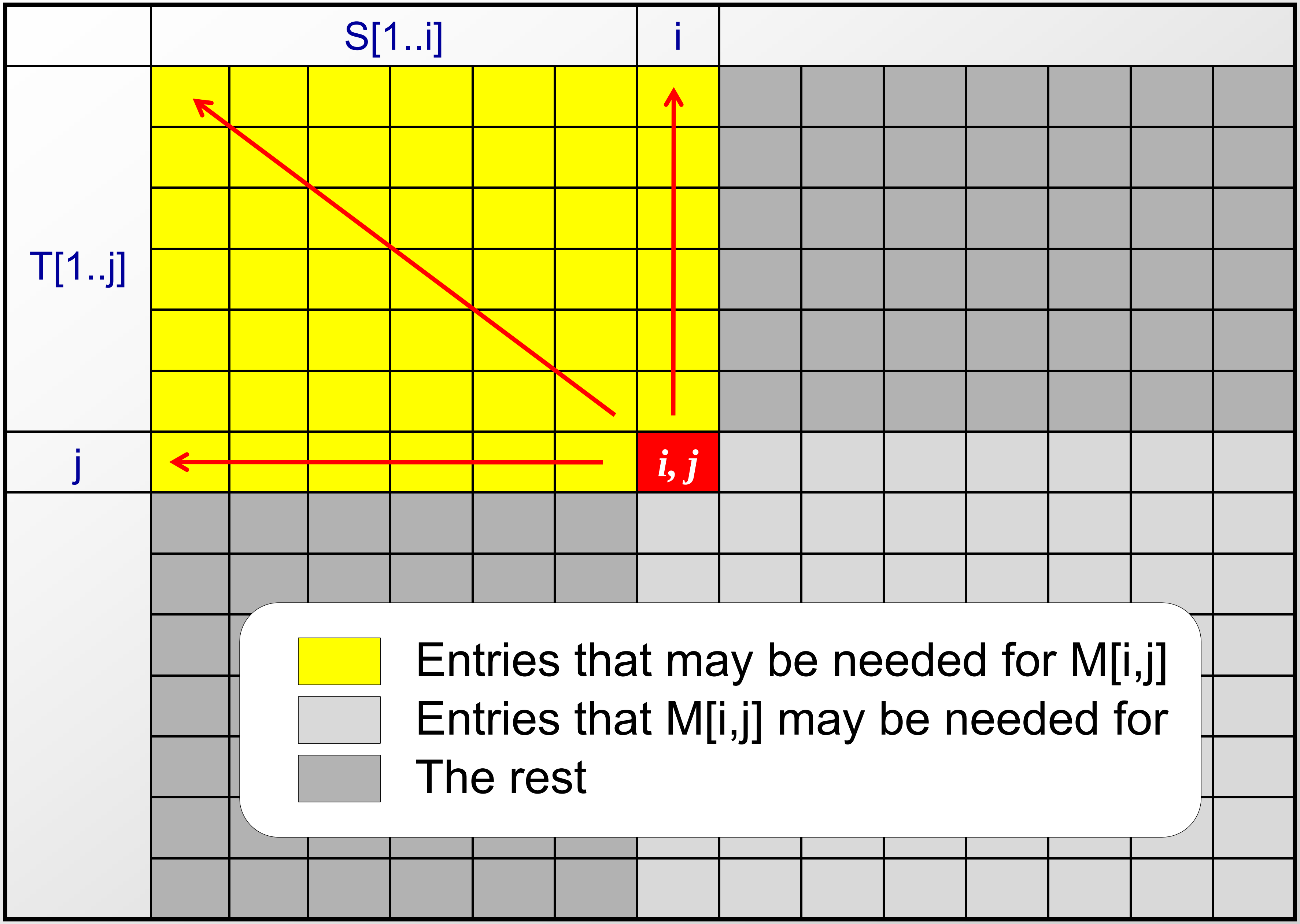
Two-dimensional optimization

Calculate sequence alignment score recursively

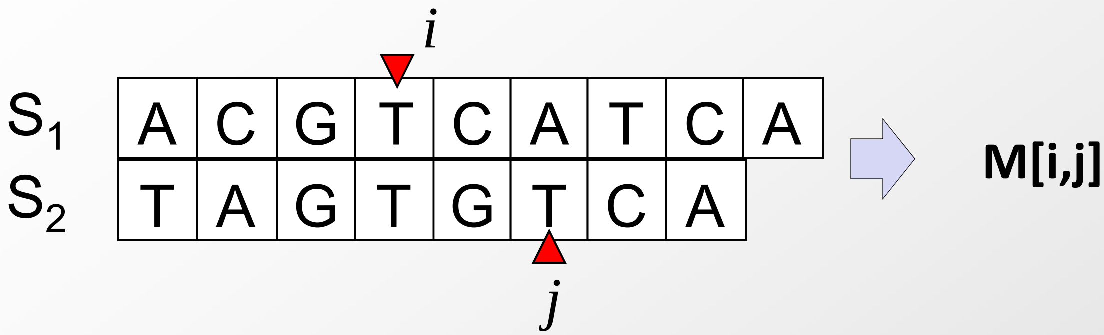


- Naïve enumeration method: exponential # alignments
- Given additive scoring function:
  - Constant cost of mutation / reward of match (e.g. 1,-1)
  - Unit cost of insertion / deletion (e.g. -2)
- Dynamic programming approach:
  - Compute all <u>prefix-to-prefix</u> alignments bottom-up
  - Matrix M[i,j] holds best alignment score S[1..i], T[1..j]
  - Express Score(i,j)=F(previously-computed scores)
  - Entry M[m,n] holds optimal score for full S,T alignment
  - Trace-back choices to obtain the actual alignment

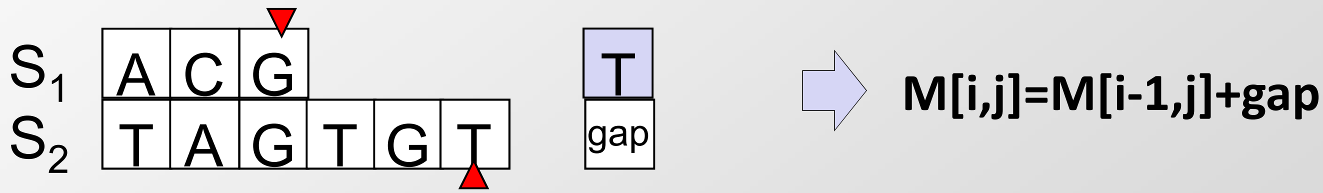
Storing the score of aligning S[1..i] to T[1..j] in M(i,j)



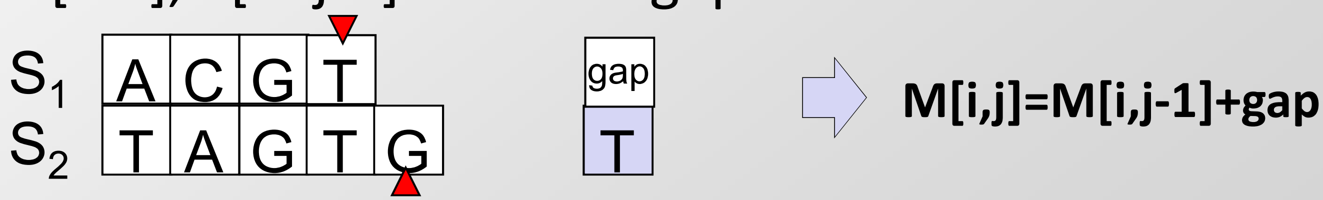
Reusing computation: recursion formula



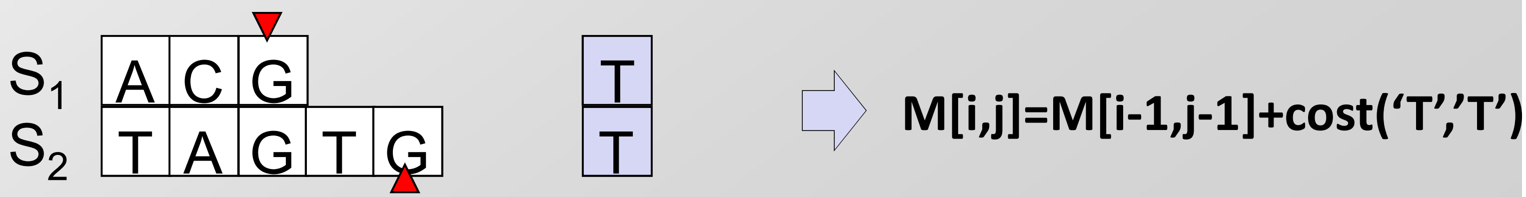
- Score of best alignment of S_1[1..i] and S_2[1..j] is max of:
  - Score of S[1..i-1],T[1..j] + cost of gap in S



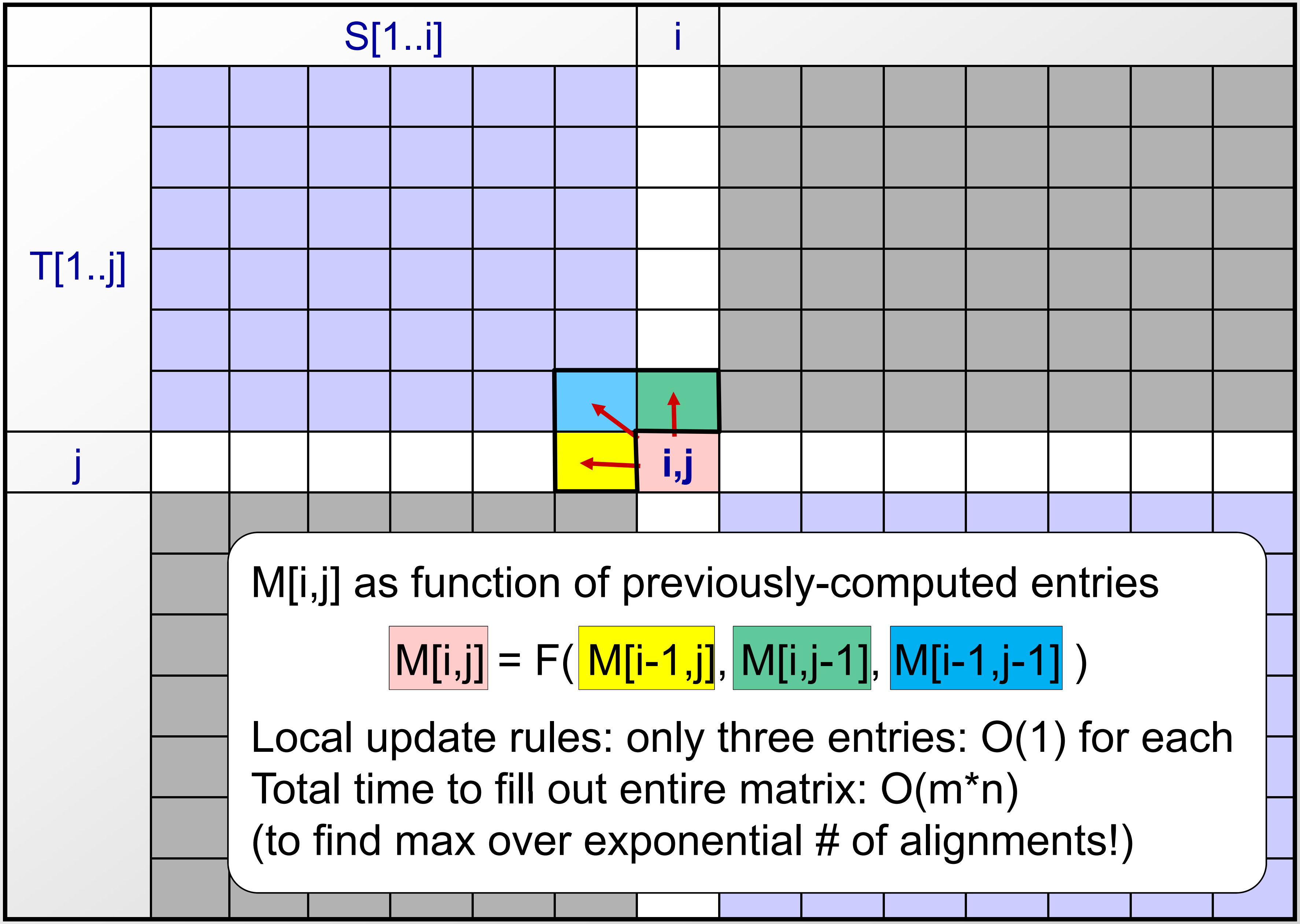
  - Score of S[1..i], T[1..j-1] + cost of gap in T



  - Score of S[1..i-1], T[1..j-1] + match cost of T[i] S[j] chars



(1, 2, 3) Store score of aligning (i,j) in matrix M(i,j)

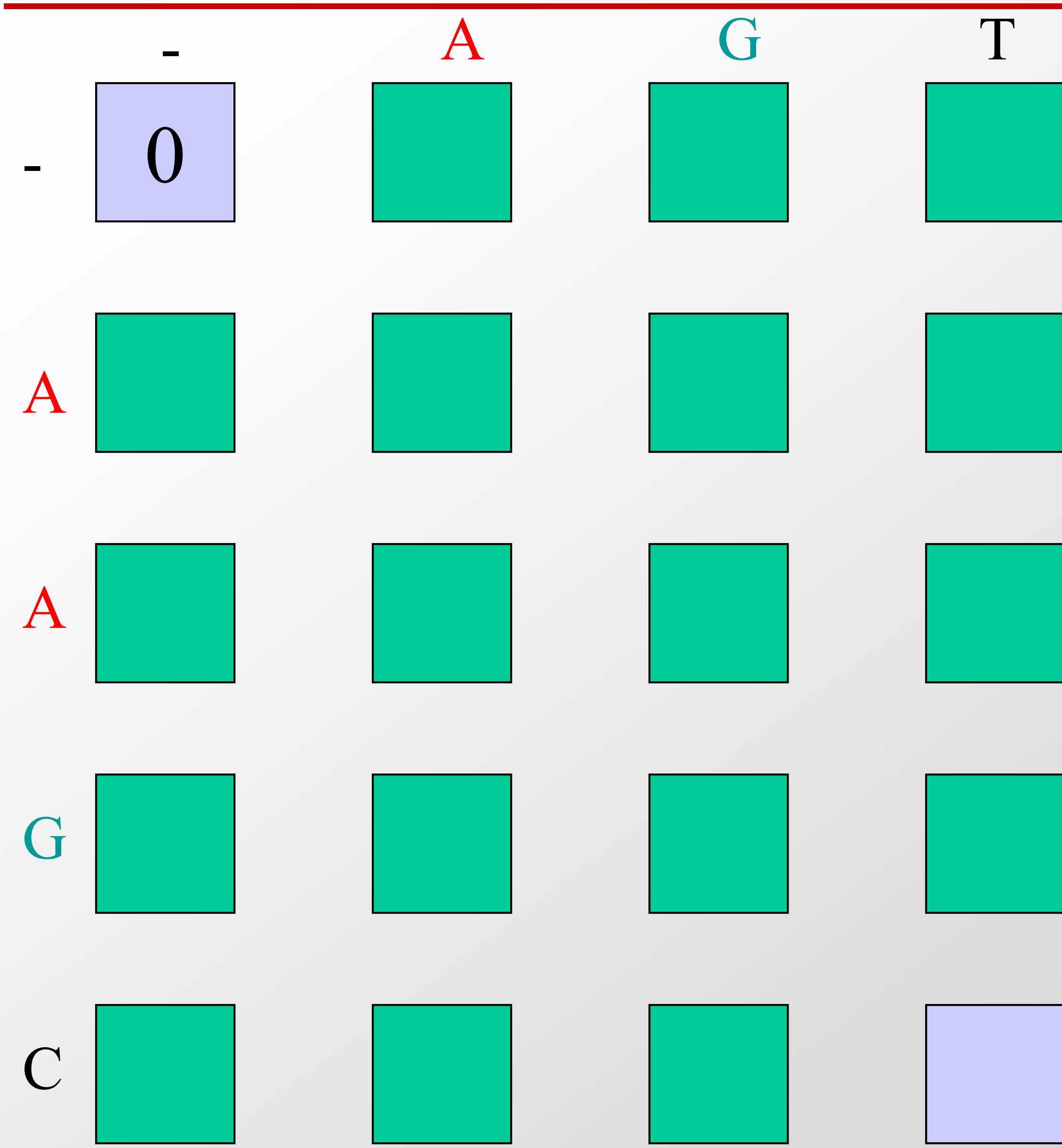


M[i,j] as function of previously-computed entries

M[i,j] = F(M[i-1,j], M[i,j-1], M[i-1,j-1])

Local update rules: only three entries: O(1) for each
Total time to fill out entire matrix: O(m*n)
(to find max over exponential # of alignments!)

Setting up the scoring matrix



Initialization:

- Top left: 0

Update Rule:

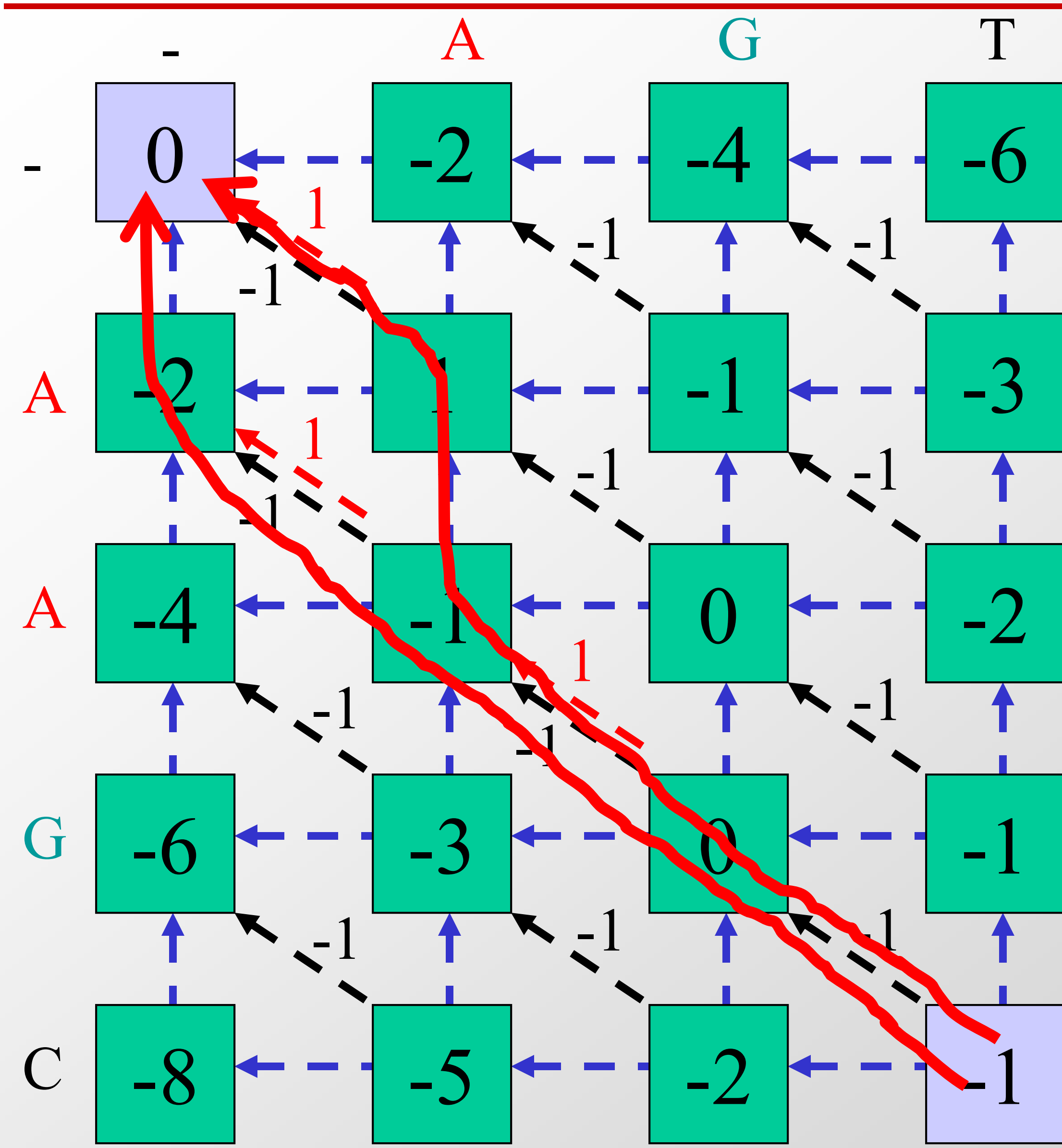
A(*i*,*j*)=max{

}

Termination:

- Bottom right

Setting up graph of scores



Initialization:

- Top left: 0

Update Rule:

A(*i*,*j*)=max{

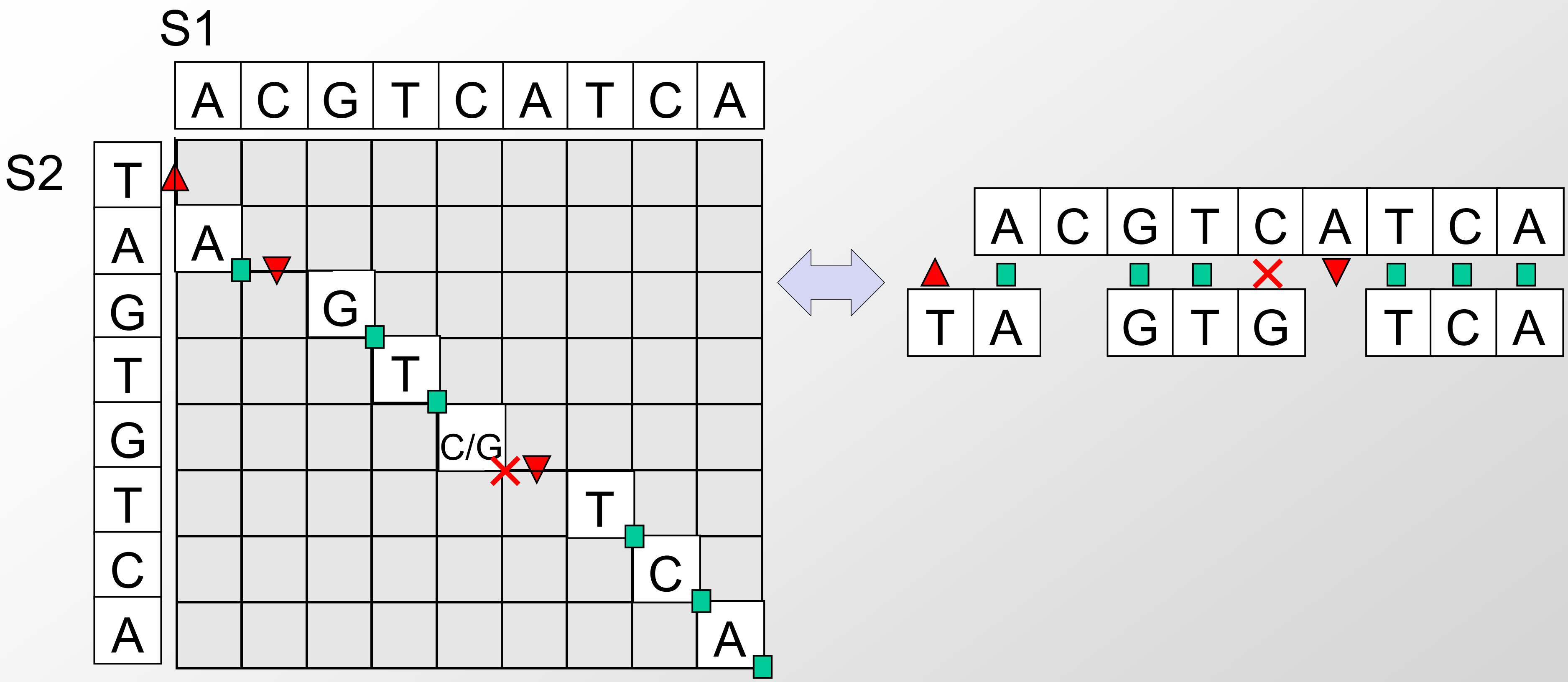
- A(*i*-1 , *j*) - 2 gap
- A(*i* , *j*-1) - 2 gap
- A(*i*-1 , *j*-1) -1 mismatch
- A(*i*-1 , *j*-1)+1 match

}

Termination:

- Bottom right

Trace-back: Path through matrix ⇔ Alignment

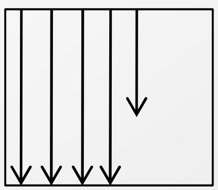


- Fill in entire table, remember best-choice pointers
- M[i,j] gives optimal score for entire alignment S_1 S_2
- Trace-back pointers gives optimal path through M
- Path through matrix corresponds 1-to-1 to alignment

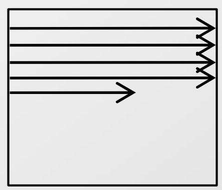
Dynamic Programming for sequence alignment

- Setting up dynamic programming

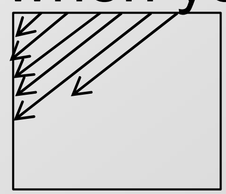
1. Find ‘matrix’ parameterization
    - Prefix parameterization. Score(S[1..i],T[1..j]) → F(i,j)
    - (i,j) only prefixes vs. (i,j,k,l) all substrings → simpler 2-d matrix
2. Make sure sub-problem space is finite! (not exponential)
    - It’s just n^2, quadratic (which is polynomial, not exponential)
3. Traversal order: sub-results ready when you need them



Cols L→R



Rows top→bot



Diags topR→botL

4. Recursion formula: larger problems = F(subparts)
    - Need formula for computing F(i,j) as function of previous results
    - Single increment at a time, only look at F(i-1,j), F(i,j-1), F(i-1,j-1) corresponding to 3 options: gap in S, gap in T, char in both
    - Score in each case depends on gap/match/mismatch penalties
5. Remember choices: typically F() includes min() or max()
    - Remember which of three cells (top,left,diag) led to maximum

Dynamic programming: design choices matter

- Dynamic programming yes, but details do matter
  - Principle: Compute next alignment based on previous alignment
  - Design choices: Make computation even more efficient
- Computing the score of a cell from its neighbors

$$F(i,j) = \max\begin{cases} F(i-1, j) - \text{gap} \\ F(i-1, j-1) + \text{score} \\ F(i, j-1) - \text{gap} \end{cases}$$

1. Parameterization: why prefixes instead of substrings
   – Prefixes allow a single recursion (top-left to bottom-right)
   – Substrings would need two (middle-to-outside top-down)
2. Local update rules, only look at neighboring cells:
   – Linear gap penalty: only neighboring cells O(1)/cell
   – Affine gap penalty: still possible with O(1)/cell
   – General gap penalty: requires O(n)/cell, or $O(n^2)$/cell

Today: Dynamic programming II

- Optimal sub-structure, repeated subproblems
- Review: Simple DP problems
  - Fibonacci numbers: Top-down vs. bottom-up
  - Crazy Eights: One-dimensional optimization
- LCS, Edit Distance, Sequence alignment
  - Two-dimensional optimization: Matrix/path duality
  - Setting up the recurrence, Fill Matrix, Traceback
- All pairs shortest paths (naïve: 2^n. n*BelFo: n^4)
  - Representing solutions. Two ways to set up DP
  - Matrix multiplication: $n^3 \lg n$. Floyd-Warshall: n^3

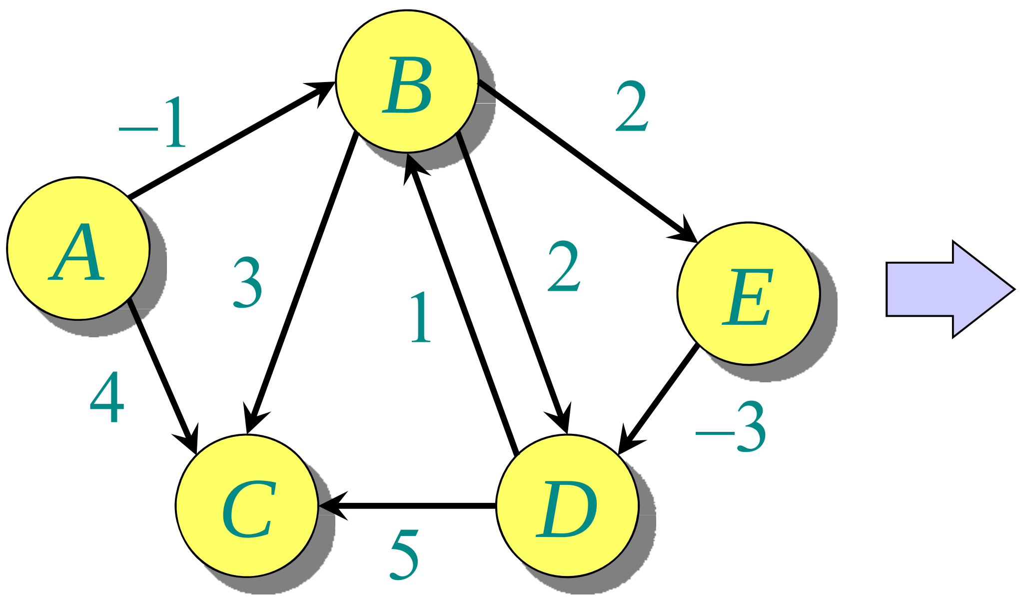
All pairs shortest paths

4. Matrix Multiplication
5. Floyd-Warshall

All-pairs shortest paths (distances)

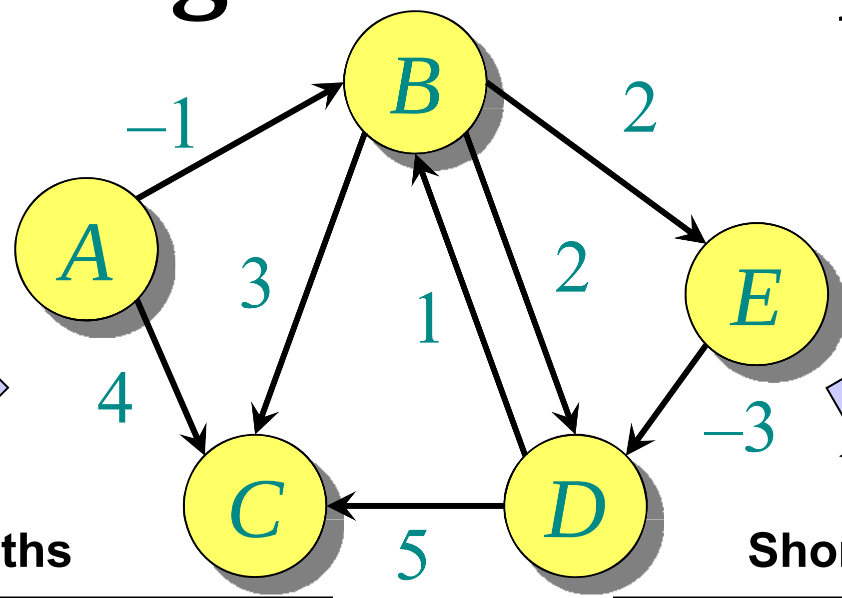
Input: Digraph $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$, with edge-weight function $w : E \to \mathbb{R}$.

Output: $n \times n$ matrix of shortest-path lengths $\delta(i, j)$ for all $i, j \in V$.



	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

Representing all shortest paths soln



A→D:
Cost: -2
Path: ((((A)B)E)D)
A→D
A→E
A→B
A→A

Shortest path lengths

	A	B	C	D	E
A	0	-1	2	-2	1
B	∞	0	3	-1	2
C	∞	∞	0	∞	∞
D	∞	1	5	0	3
E	∞	-2	2	-3	0

Shortest path predecessors

	A	B	C	D	E
A	A	A	B	E	B
B	nil	B	B	E	B
C	nil	nil	C	nil	nil
D	nil	D	D	D	B
E	nil	D	D	D	E

All-pairs shortest path algorithms

- **Idea 1: Run Bellman-Ford** once for each vertex
  - Time: $O(V^2E) = O(n^4)$ in the worst case for dense graphs
- **Idea 2: Dynamic Programming**
  - Build optimal paths from optimal subpaths. (Optimization procedure… greedy doesn't work)
  - **Matrix multiplication**: consider paths of increasing length, iterative over length of the path
  - **Floyd-Warshall:** consider paths involving increasing subsets of vertices, one more vertex at each iteration
- **Idea 3: Graph re-weighing**
  - **Johnson:** graph rewiring to eliminate negative edges, then run Dijkstra's $|V|$ times

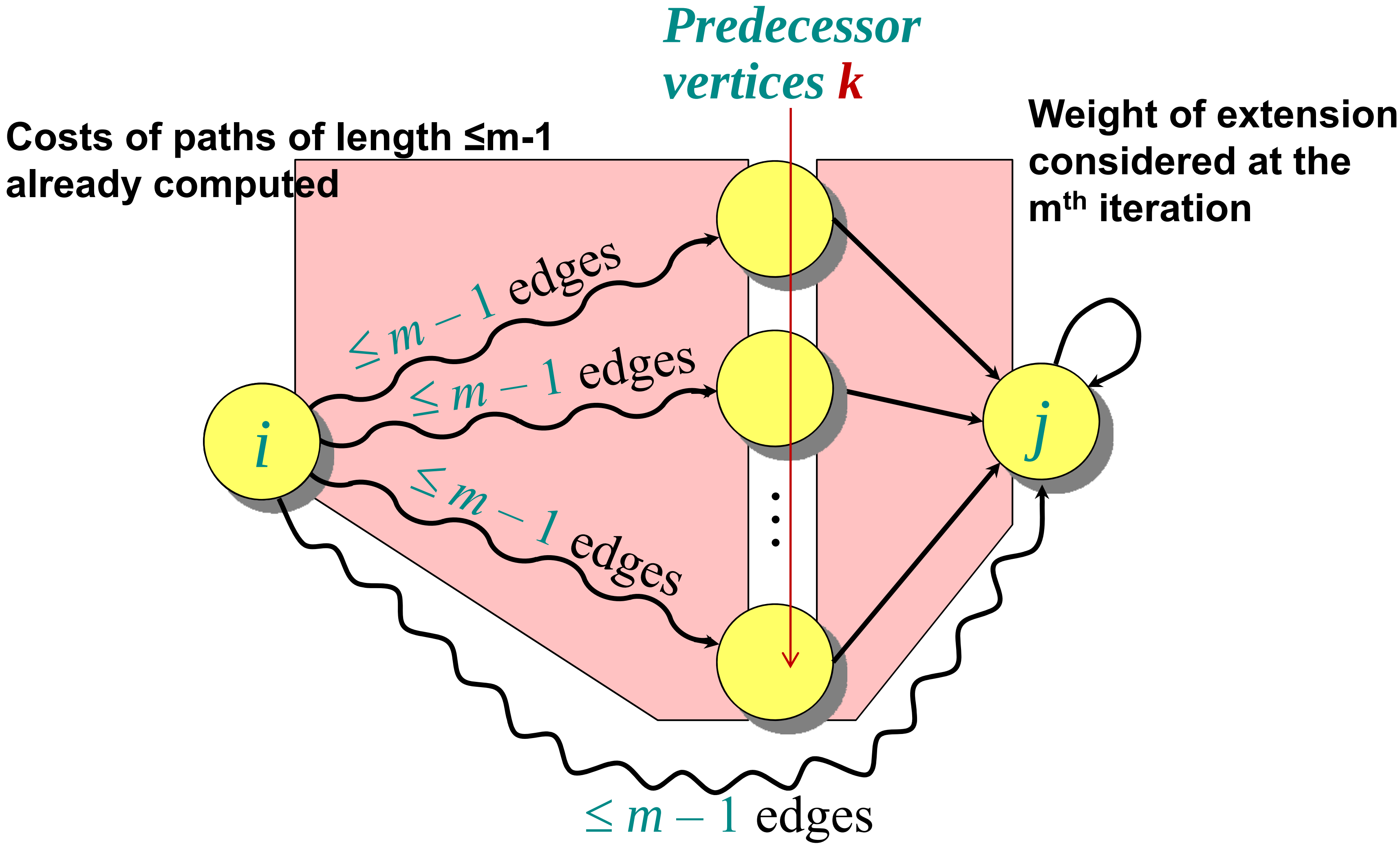
Shortest path algorithms

Setting	Weights	Principle	Algorithm
Single source	$=1$	Greedy	BFS: $O(V+E)$
Single source	≥ 0	Greedy	Dijkstra: $O(E+V \lg V)$
Single source	General	$\|V\|-1$ passes	Bellman-Ford: $O(V \cdot E)$
All pairs	General	DP-length	Matrix Mult: $O(V^3 \lg V)$
All pairs	General	DP-vertices	Floyd-Warshall: $O(V^3)$
All pairs	General	Reweigh	Johnson: $O(V \cdot E+V^2 \lg V)$

4. Matrix Multiplication

Consider paths of increasing length
at each iteration

Intuition: Extend one hop at a time



$$\text{Cost}[\boldsymbol{i} \rightarrow \boldsymbol{j}] = \min_{\boldsymbol{k}} \{\text{Cost}[i \rightarrow k] + \text{EdgeWeight}(k \rightarrow j)\}$$

Already computed

Compute optimal path from optimal subpaths

Consider the $n \times n$ weighted adjacency matrix $A = (a_{ij})$, where $a_{ij} = w(i, j)$ or ∞, and define

$d_{ij}^{(m)}$ = weight of a shortest path from i to j that uses at most m edges.

Claim: We have

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j, \\ \infty & \text{if } i \neq j; \end{cases}$$

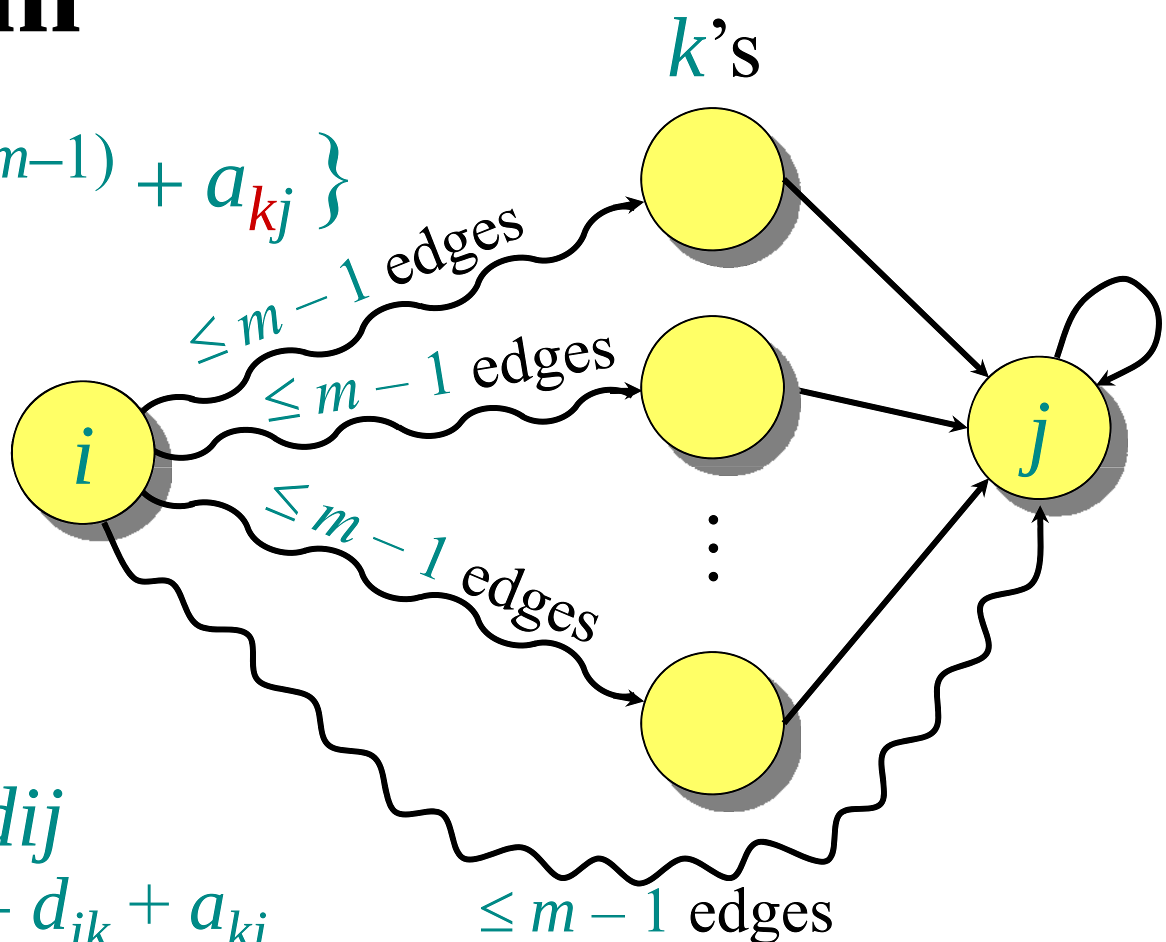
and for $m = 1, 2, \ldots, n - 1$,

$$d_{ij}^{(m)} = \mathbf{min}_k \{ d_{ik}^{(m-1)} + a_{kj} \} .$$

Principle of dynamic programming

Proof of claim

$$d_{ij}^{(m)} = \min_k \{d_{ik}^{(m-1)} + a_{kj}\}$$



Relaxation!

for $k \leftarrow 1$ **to** n
do if $d_{ik} + a_{kj} < dij$
then $d_{ij} \leftarrow d_{ik} + a_{kj}$

Note: No negative-weight cycles implies

$$\delta(i, j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \ldots$$

Since no shortest path has more than n-1 edges.

Matrix multiplication

Compute $C = A \cdot B$, where C, A, and B are $n \times n$ matrices:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \,.$$

Time $= \Theta(n^3)$ using the standard algorithm.

What if we map "$+$" $\to$ "min" and "$\cdot$" $\to$ "$+$"?

$$c_{ij} = \min_k \{a_{ik} + b_{kj}\}.$$

Thus, $D^{(m)} = D^{(m-1)}$ "$\times$" A.

$$\text{Identity matrix} = \mathrm{I} = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix} = D^0 = (d_{ij}{}^{(0)}).$$

Matrix multiplication (continued)

The (min, +) multiplication is ***associative***, and with the real numbers, it forms an algebraic structure called a ***closed semiring***.

Consequently, we can compute

$$
\begin{aligned}
D^{(1)} &= D^{(0)} \cdot A = A^1 \\
D^{(2)} &= D^{(1)} \cdot A = A^2 \\
&\vdots \\
D^{(n-1)} &= D^{(n-2)} \cdot A = A^{n-1} ,
\end{aligned}
$$

yielding $D^{(n-1)} = (\delta(i, j))$.

Time $= \Theta(n \cdot n^3) = \Theta(n^4)$. No better than $n \times$ B-F.

Improved matrix multiplication algorithm

Repeated squaring: $A^{2k} = A^k \times A^k$.

Compute $\underbrace{A^2, A^4, \ldots, A^{2^{\lceil \lg(n-1) \rceil}}}_{O(\lg n) \text{ squarings}}$.

Note: $A^{n-1} = A^n = A^{n+1} = \cdots$. (no need to worry about odd/even split)

Time $= \Theta(n^3 \lg n)$.

To detect negative-weight cycles, check the diagonal for negative values in $O(n)$ additional time.

Shortest path algorithms

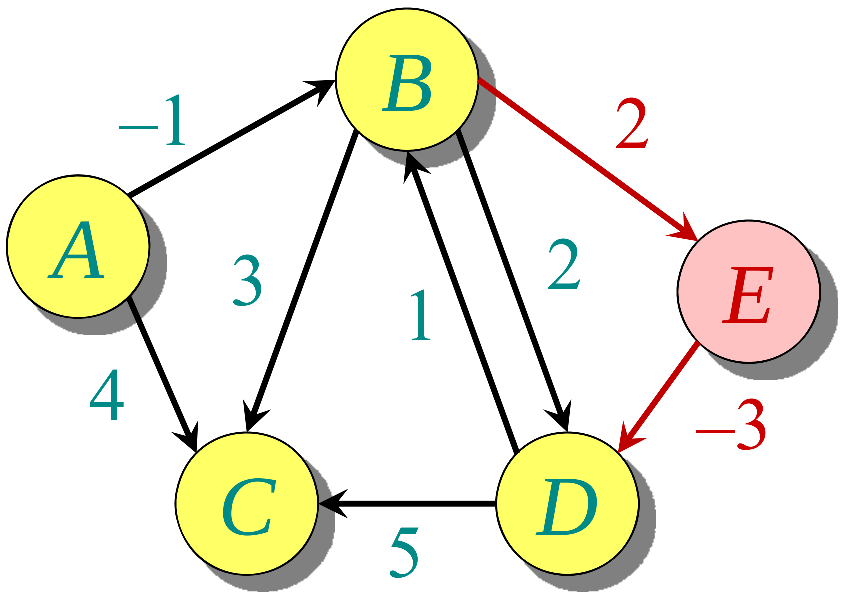
Setting	Weights	Principle	Algorithm
Single source	=1	Greedy	BFS: $O(V+E)$
Single source	≥0	Greedy	Dijkstra: $O(E+V\lg V)$
Single source	General	$\|V\|$-1 passes	Bellman-Ford: $O(V\cdot E)$
All pairs	General	DP-length	Matrix Mult: $O(V^3\lg V)$
All pairs	General	DP-vertices	Floyd-Warshall: $O(V^3)$
All pairs	General	Reweigh	Johnson: $O(V\cdot E+V^2\lg V)$

5. Floyd-Warshall algorithm

Consider one additional vertex each time

Intuition: Extend one vertex at a time

"Now considering all paths that also include E"



	A	B	C	D	E
A	0	-1	2	~~1~~ -2	1
B	∞	0	3	~~2~~ -1	2
C	∞	∞	0	∞	∞
D	∞	1	5	0	3
E	∞	-2	2	-3	0

$$\text{Cost}[i \rightarrow j] = \min_k \{\text{Cost}[i \rightarrow k] + \text{EdgeWeight}(k \rightarrow j)\}$$

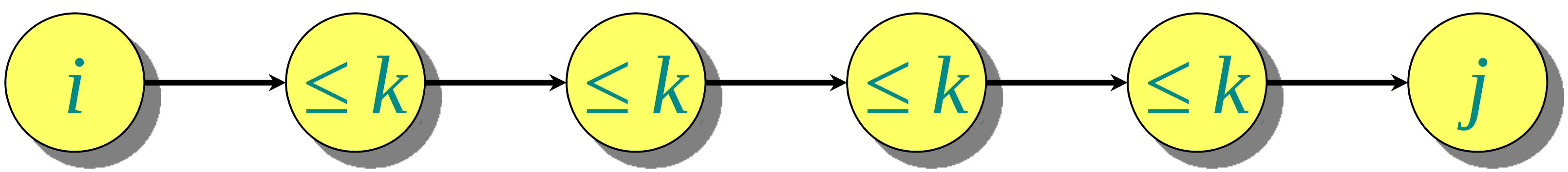
Already computed for all vertices <k

Floyd-Warshall algorithm

Different way of ordering the computation, considering increasing numbers of vertices (instead of increasing lengths of paths).

Also dynamic programming, but faster!

Define $c_{ij}^{(k)} =$ weight of a shortest path from i to j with intermediate vertices belonging to the set $\{1, 2, \ldots, k\}$.

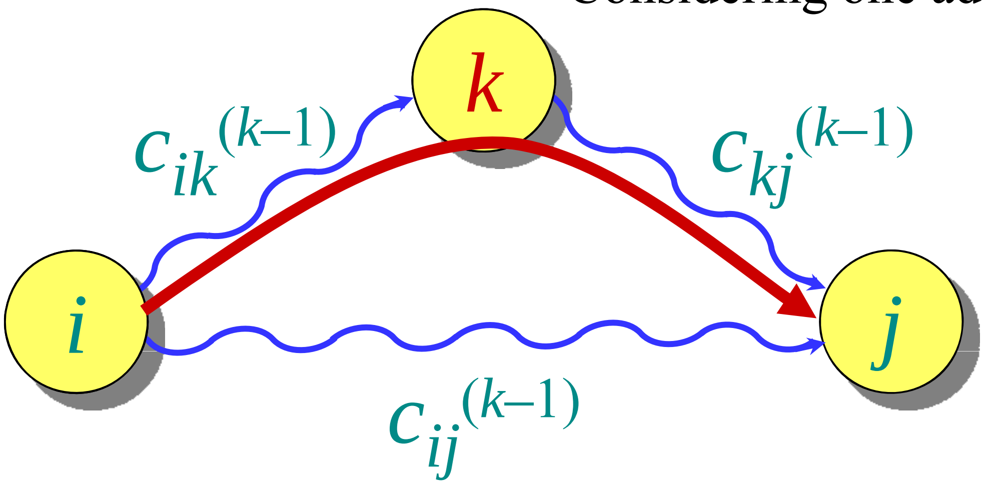


Thus, $\delta(i, j) = c_{ij}^{(n)}$. Also, $c_{ij}^{(0)} = a_{ij}$.

Floyd-Warshall recurrence

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

Considering one additional vertex k



 intermediate vertices in $\{1, 2, \ldots, k-1\}$

 intermediate vertices in $\{1, 2, \ldots, k-1, \boldsymbol{k}\}$

Pseudocode for Floyd-Warshall

Considering each vertex in order: n

Updating all n² path lengths

O(1) work for each

for $k \leftarrow 1$ **to** n
do for $i \leftarrow 1$ **to** n
do for $j \leftarrow 1$ **to** n
do if $c_{ij} > c_{ik} + c_{kj}$
then $c_{ij} \leftarrow c_{ik} + c_{kj}$

} ***relaxation***

Notes:

- Okay to omit superscripts, since extra relaxations can't hurt.
- Runs in $\Theta(n^3)$ time.
- Simple to code.
- Efficient in practice.

Ex: now considering paths through k=*D*

To:

From:

O(n^2) ‘updates’ at step k, each costing O(1)

	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

	A	B	C	D	E
A	0	?	?	?	?
B	?	0	?	?	?
C	?	?	0	?	?
D	?	?	?	0	?
E	?	?	?	?	0

Application: Transitive closure of directed graph

(all vertices ***j*** reachable from each vertex ***i***)

Compute $t_{ij} = \begin{cases} 1 & \text{if there exists a path from } i \text{ to } j, \\ 0 & \text{otherwise.} \end{cases}$

IDEA: Use Floyd-Warshall, but with $(\vee, \wedge)$ instead of $(\min, +)$:

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}).$$

Time $= \Theta(n^3)$.

Shortest path algorithms

Setting	Weights	Principle	Algorithm
Single source	=1	Greedy	BFS: $O(V+E)$
Single source	≥0	Greedy	Dijkstra: $O(E+V\lg V)$
Single source	General	$\|V\|-1$ passes	Bellman-Ford: $O(V\cdot E)$
All pairs	General	DP-length	Matrix Mult: $O(V^3\lg V)$
All pairs	General	DP-vertices	Floyd-Warshall: $O(V^3)$
All pairs	General	Reweigh	Johnson: $O(V\cdot E+V^2\lg V)$

Today: Dynamic programming II

- Optimal sub-structure, repeated subproblems
- Review: Simple DP problems

1. Fibonacci numbers: Top-down vs. bottom-up
2. Crazy Eights: One-dimensional optimization

- 3. LCS, Edit Distance, Sequence alignment

– Two-dimensional optimization: Matrix/path duality

– Setting up the recurrence, Fill Matrix, Traceback

- All pairs shortest paths (naïve: 2^n. n*BelFo: n^4)

4. DP by number of hops: Matrix multiplication: $n^3 \lg n$.
5. DP by vertices considered: Floyd-Warshall: n^3

Dynamic Programming module

- Optimization technique, widely applicable
  - ➢Optimal substructure ➢Overlapping subproblems
- Tuesday: Simple examples, alignment
  - Simple examples: Fibonacci, Crazy Eights
  - Alignment: Edit distance, molecular evolution
- Today: More DP
  - Alignment: Bound, Linear Space, Affine Gaps
  - Back to paths: All Pairs Shortest Paths DP1,DP2
- Next week:
  - Knapsack (shopping cart) problem
  - Text Justification
  - Structured DP: Vertex Cover on trees, phylogeny

Happy Patriot's Day!

Unit	Pset	Week	Date		Lecture (Tuesdays and Thursdays)		Recitation (Wed and Fri)
Intro	PS1	1	Tue Feb 01	1	Introduction and Document Distance	1	Python and Asymptotic Complexity
Binary	Out: 2/1		Thu Feb 03	2	Peak Finding Problem	2	Peak Finding correctness & analysis
Search	Due: Mon 2/14	2	Tue Feb 08	3	Scheduling and Binary Search Trees	3	Binary Search Tree Operations
Trees	HW lab: Sun 2/13		Thu Feb 10	4	Balanced Binary Search Trees	4	Rotations and AVL tree deletions
Hashing	PS2 Out: 2/15	3	Tue Feb 15	5	Hashing I : Chaining, Hash Functions	5	Hash recipes, collisions, Python dicts
	Due: Mon 2/28		Thu Feb 17	6	Hashing II : Table Doubling, Rolling Hash	6	Probability review, Pattern matching
	HW lab:Sun 2/27	4	Tue Feb 22	-	President's Day - Monday Schedule - No Class	-	No recitation
			Thu Feb 24	7	Hashing III : Open Addressing	7	Universal Hashing, Perfect Hashing
Sorting	PS3. Out: 3/1	5	Tue Mar 01	8	Sorting I : Insertion & Merge Sort, Master Theorem	8	Proof of Master Theorem, Examples
	Due: Mon 3/7		Thu Mar 03	9	Sorting II : Heaps	9	Heap Operations
	HW lab: Sun 3/6	6	Tue Mar 08	10	Sorting III: Lower Bounds, Counting Sort, Radix Sort	10	Models of computation
			Wed Mar 09	Q1	Quiz 1 in class at 7:30pm. Covers L1-R10. Review Session on Tue 3/8 at 7:30pm.		
Graphs	PS4. Out: 3/10		Thu Mar 10	11	Searching I: Graph Representation, Depth-1st Search	11	Strongly connected components
and	Due: Fri 3/18	7	Tue Mar 15	12	Searching II: Breadth-1st Search, Topological Sort	12	Rubik's Cube Solving
Search	HW lab:W 3/16		Thu Mar 17	13	Searching III: Games, Network properties, Motifs	13	Subgraph isomorphism
Shortest	PS5	8	Tue Mar 29	14	Shortest Paths I: Introduction, Bellman-Ford	14	Relaxation algorithms
Paths	Out: 3/29		Thu Mar 31	15	Shortest Paths II: Bellman-Ford, DAGs	15	Shortest
	Due: Mon 4/11	9	Tue Apr 05	16	Shortest Paths III: Dijkstra	16	Speeding
	HW lab:Sun 4/10		Thu Apr 07	17	Graph applications, Genome Assembly	17	Euler To
Dynamic	PS6	10	Tue Apr 12	18	DP I: Memoization, Fibonacci, Crazy Eights	18	Limits of dynamic programming
Program	Out: Tue 4/12		Wed Apr 13	Q2	Quiz 2 in class at 7:30pm. Covers L11-R17. Review Session on Tue 4/13 at 7:30pm.		
ming	Due: Fri 4/29		Thu Apr 14	19	DP II: Shortest Paths, Genome sequence alignment	19	Edit Distance, LCS, cost functions
	HW lab:W 4/27	11	Tue Apr 19	-	Patriot's Day - Monday and Tuesday Off	-	No recitation
			Thu Apr 21	20	DP III: Text Justification, Knapsack	20	Saving Princess Peach
		12	Tue Apr 26	21	DP IV: Piano Fingering, Vertex Cover, Structured DP	21	Phylogeny
Numbers	PS7 out Thu4/28		Thu Apr 28	22	Numerics I - Computing on large numbers	22	Models of computation return!
Pictures	Due: Fri 5/6	13	Tue May 3	23	Numerics II - Iterative algorithms, Newton's method	23	Computing the nth digit of π
(NP)	HW lab: Wed 5/4		Thu May 5	24	Geometry: Line sweep, Convex Hull	24	Closest pair
		14	Tue May 10	25	Complexity classes, and reductions	25	Undecidability of Life
Beyond			Thu May 12	26	Research Directions (15 mins each) + related classes		
		15	Finals week	Q3	Final exam is cumulative L1-L26. Emphasis on L18-L26. Review Session on Fri 5/13 at 3pm		

Dynamic Programming