Dynamic Programming (cont'd)

Example 2: Maximum Subarray Problem

$$\cdots\ 5\ \ 2\ \ -8\ \ \underbrace{\overset{i}{13}\ \ -9\ \ 33\ \ 24\ \ 0\ \ -14\ \ \overset{j}{58}}_{105}\ \ 5\ \ 42\ \ 31\ \cdots$$

Given a sequence of numbers, find a contiguous subarray with maximum sum.

First attempt: compute the sum for $\forall i, j$: $\Theta(n^3)$

First improvement:

Compute $S[i] = \sum_{j=1}^{i} A[j]$ $\quad \Theta(n)$

$\forall i, j$: compute $S[j] - S[i-1]$: $\quad \Theta(n^2)$

Linear time Algorithm (Kadane's alg):

let $B[j]$: max sum of a subarray ending at $A[j]$

Def of subproblem

Goal : $\max_j B[j]$

Recurrence:
$$B[j] = \begin{cases} A[j] & j = 1 \\ \max\{A[j],\ A[j] + B[j-1]\} & j > 1 \end{cases}$$

$\Theta(n)$ total.

Extension to compute best of submatrices:

naive algorithm: $\Theta(n^4)$

Can use the idea above to run in $\Theta(n^3)$

$$A = \begin{bmatrix} -2 & 5 & 0 & -5 & -2 & 2 & -3 \\ 4 & -3 & -1 & 3 & 2 & 1 & -1 \\ -5 & 6 & 3 & -5 & -1 & -4 & -2 \\ -1 & -1 & 3 & -1 & 4 & 1 & 1 \end{bmatrix}$$

above to run in $\Theta(n^3)$

$$A = \begin{bmatrix} -5 & 6 & 3 & -5 & -1 & -4 & -2 \\ -1 & -1 & 3 & -1 & 4 & 1 & 1 \\ 3 & -3 & 2 & 0 & 3 & -3 & -2 \\ -2 & 1 & -2 & 1 & 1 & 3 & -1 \\ 2 & -4 & 0 & 1 & 0 & -3 & -1 \end{bmatrix}$$

13

From slides by KW

- Compute the sum of sub-rows : $S_{ij} = \sum_{k=1}^{j} A_{ik}$
- For each $j < j'$, define another array

$$L_i = S_{ij} - S_{ij'}$$

- Solve the one-dimensional problem on List $L[\cdot]$.

$$A = \begin{array}{c} \begin{array}{ccccccc} & & j & & j' & & \end{array} \\ \begin{bmatrix} -2 & 5 & 0 & -5 & -2 & 2 & -3 \\ 4 & -3 & -1 & 3 & 2 & 1 & -1 \\ -5 & 6 & 3 & -5 & -1 & -4 & -2 \\ -1 & -1 & 3 & -1 & 4 & 1 & 1 \\ 3 & -3 & 2 & 0 & 3 & -3 & -2 \\ -2 & 1 & -2 & 1 & 1 & 3 & -1 \\ 2 & -4 & 0 & 1 & 0 & -3 & -1 \end{bmatrix} \end{array} \qquad x = \begin{bmatrix} -7 \\ 4 \\ -3 \\ 6 \\ 5 \\ 0 \\ 1 \end{bmatrix} \quad 0 - 5 - 2$$

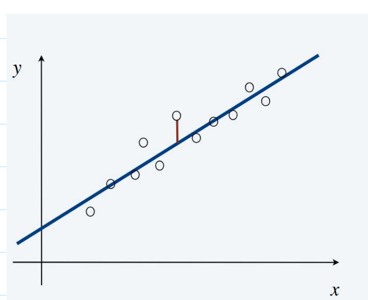
From slides by KW

Example 3: Segmented Least Square

- Given a set of n points on the plane

$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$

$x_1 \le x_2 \le \cdots \le x_n$

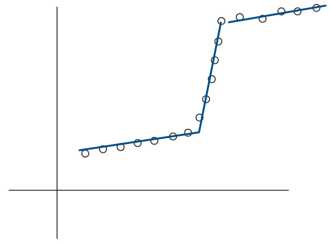


- Find a straight line L s.t. min sum of square of distances of points to L

- $L: y = ax + b$ $\qquad \underbrace{\text{Error}}_{\text{SSD}}(L, P) = \sum_{i=1}^{n} (y_i - ax_i - b)^2$

– Easy to solve: $a = \frac{n \sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n \sum_i x_i^2 - (\sum_i x_i)^2}$ and $b = \frac{\sum_i y_i - a \sum_i x_i}{n}$.

Complication: Points might lie on not one but multiple straight lines.



Reasonable function to optimize

Decide on the number of line segments;

(minimize sum of squares of distances) + (C times the # of lines)

penalty factor

$f(x) = \text{Error} + C \cdot L \longrightarrow$ # of lines

what should be the subproblem? how many lines?

▷ Opt[j]: Cost of best solution for $P_1 \dots P_j$

▷ e_{ij}: SSD for points $P_i, \dots, P_j$

Def of subprob

Goal: Opt[n]

Recurrence: $\text{Opt}[0] = 0$

$$\text{Opt}[j] = \min_{1 \le i \le j} \{ e_{ij} + C + \text{Opt}[i-1] \}$$

- For $j \leftarrow 1$ to n do
    - for $i \leftarrow 1$ to j do
        - compute SSD e_{ij} for $P_i \cdots P_j$
- $Opt[0] \leftarrow 0$
- For $j \leftarrow 1$ to n do

$$Opt[j] \leftarrow \min_{1 \le i \le j} \{ e_{ij} + c + Opt[i-1] \}$$

- $Opt[n]$: cost of optimum
- One can easily find the actual solution.
- Time complexity:
    - $*$ naive alg to compute e_{ij}'s : $O(n^3)$
    - $*$ Can improve : each e_{ij} uses $O(1)$ extra time

    total time : $O(n^2)$

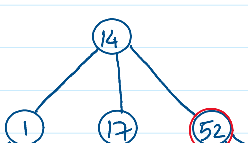
Example 4: Maximum Independent Set on trees

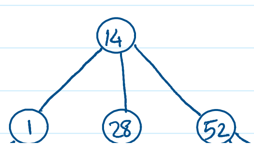
Weighted Independent Set:

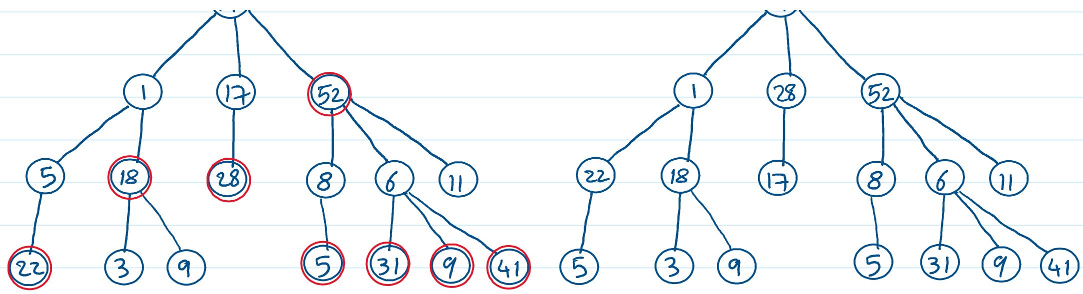
- Given a graph $G = (V, E)$, $w: V \to \mathbb{R}^+$

  A set $V' \subseteq V$ is an independent set iff

  $\forall u, v \in V' : uv \notin E$
- Goal: Find max weight independent set
- Extremly difficult in general graphs.
- what if input graph is a tree T?







- consider the tree rooted, can we solve the problem for each subtree T_v (tree rooted at v)?

- $M[v]$: max weight of an independent set for tree T_v

observation:

1) if $M[v]$ includes v then we look at best solutions of grand children

2) if $M[v]$ does not include v then take the sum of its children best solutions

$$M[v] = \begin{cases} w_v & v \text{ is a leaf} \\ \max\left\{ \sum_{u:\text{ child of } v} M[u],\ w_v + \sum_{u:\text{ grandchild}} M[u] \right\} & \text{o.w.} \end{cases}$$

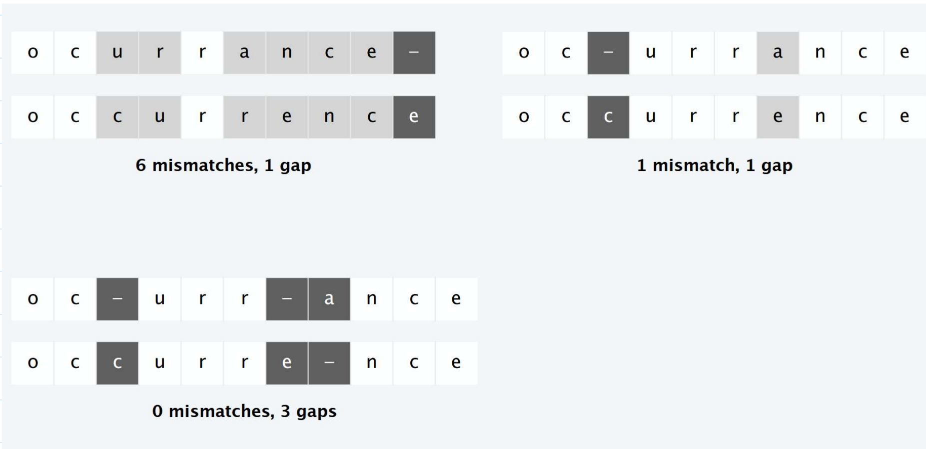
Time complexity: $\Theta(n)$

Example 5: Sequence alignment

- spell checking, DNA sequence checking

- how close are two string/sequence or how one can be changed (easily) to the other?

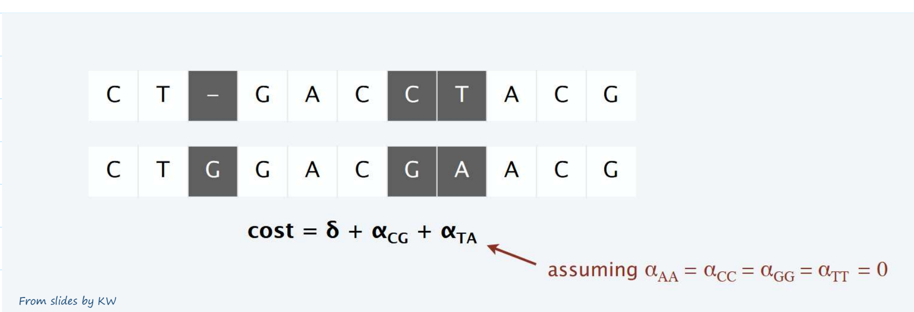
Example: Ocurrance vs Occurrence



From slides by KW

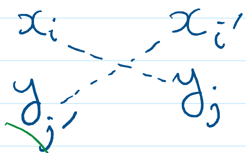
Edit distance:

- Given two strings $X = x_1, \dots, x_m$, $Y = y_1, \dots, y_n$ over alphabet Σ
- Cost functions: gap penalty δ
  mismatch penalty δ_{pq} $\forall p, q \in \Sigma$
- Edit cost: Sum of gap + mismatch penalty



- Applications: Bioinformatics, translation, spell checking, etc.
- Alignment: is a set M of ordered pairs s.t.

each input letter appears in at most one pair and there is no crossings



$$\mathrm{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\left(\sum_{x_i \text{ not matched}} \delta + \sum_{y_j \text{ not matched}} \delta \right)}_{\text{gap cost}}$$

Subproblem: let $A[i,j]$ denote the cost of best alignment of $x_1 \dots x_i$ and $y_1 \dots y_j$

Goal: $A[m,n]$

Claim: In an optimal alignment M one of the following is true:

(i) $(m,n) \in M$

(ii) x_m is not matched

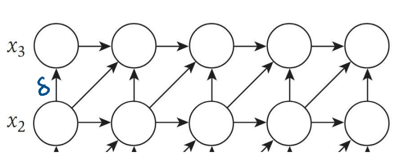
(iii) y_n is not matched.

Recurrence:

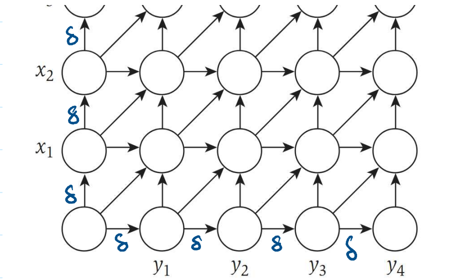
$$A[i,0] = i \cdot \delta \qquad A[0,j] = j \cdot \delta$$

$\forall\ i,j \geq 1$

$$A[i,j] = \min \begin{cases} \alpha_{x_i y_j} + A[i-1, j-1] \\ \delta + A[i-1, j] \\ \delta + A[i, j-1] \end{cases}$$



$\alpha_{x_i y_j}$



$\nearrow$ $\alpha_{x_i y_j}$

$\uparrow$ and $\longrightarrow$: δ

Claim: if $f(i,j)$ is the weight of min_cost path from $(0,0)$ to (i,j) node then $f(i,j) = A[i,j]$

Time and space complexity: $\Theta(m.n)$ for both

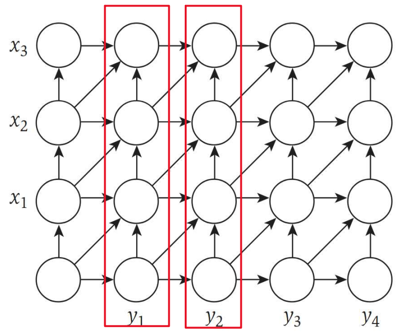
Theorem [Backurs/Indyk '15]: If we can solve Edit_Distance in time $O(n^{2-\varepsilon})$ for any $\varepsilon > 0$ then we can solve SAT instances with N var's & M clauses in time $M^{O(1)} 2^{(1-\delta)N}$ for some $\delta > 0$ (refuting Strong Exponential time Hypothesis SETH).

Question: Can we improve the space complexity?

- A clever idea of mixing DP & divide & Conquer can reduce space to $\Theta(m+n)$ (Hirschberg's alg)
- For now suppose we only want the value of optimum

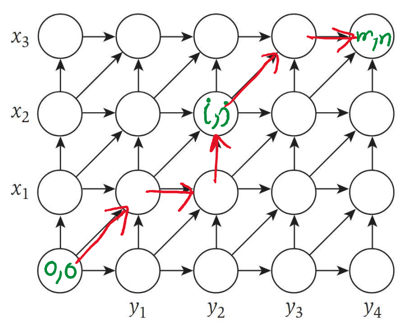
Observation 1: to compute $f(i,j)$ (or $A[i,j]$) we only need values from prev. row or column

- We can maintain the values of prev. column & the curren column in $\Theta(m)$ space



- The information stored is not enough to find the optimum alignment itself.
- More ideas are needed if we want the alignment
- call this Algorithm 2.

Observation 2: Let $g(i,j)$ denote the length of the shortest path from (i,j) to (m,n)



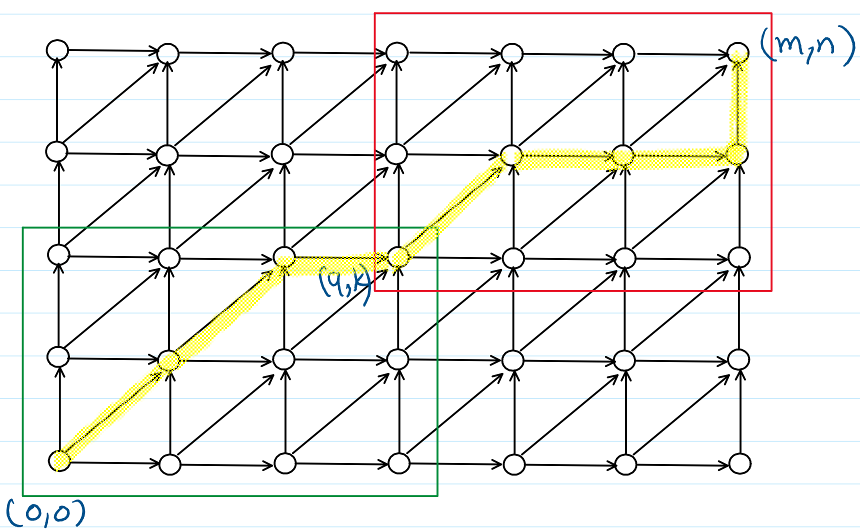
- this is the same as the cost of best alignment for $x_i \cdots x_m$ and $y_j \cdots y_n$

- There is a similar DP formula to compute $g(i,j)$ values (Backward DP)

Observation 3: The length of the shortest path from $(0,0)$ to (m,n) that passes through (i,j) is $f(i,j)+g(i,j)$

proof: See KT.

lemma: Let $k \in \{0, \dots, n\}$ (can assume $k \approx \frac{n}{2}$) and q be an index that minimizes $f(q,k)+g(q,k)$. Then there is a corner-to-corner path of min length that goes through (q,k).



D&C idea: Find the value of q that minimizes $f(q, \frac{n}{2}) + g(q, \frac{n}{2})$ and recurse!

- To compute the value of optimum alignment we compute $f(i, \frac{n}{2})$ and $g(i, \frac{n}{2})$ for all values of i using Algorithm 2.
- we pick the smallest i (save $(i, \frac{n}{2})$) and find the actual minimum path from $(0,0)$ to $(i, \frac{n}{2})$ and then from $(i, \frac{n}{2})$ to (m,n) recursively

Lemma: Space complexity of this algorithm is $\Theta(m+n)$.

Proof: Each recursive call needs linear space to compute $f(i, \frac{n}{2})$ and $g(i, \frac{n}{2})$

- space to store these min-values is linear as # of recursive calls is $O(n)$.

lemma: The running time of D&Q+DP for sequence alignment is $\Theta(mn)$.

proof: let $T(m,n)$ be the time complexity

We prove by strong induction $T(m,n) \leq k \cdot mn$ for some constant $k>0$.

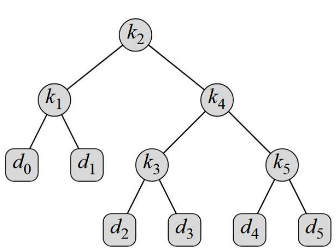
$$T(m,n) \leq c \cdot mn + T(i, \tfrac{n}{2}) + T(m-i, \tfrac{n}{2})$$

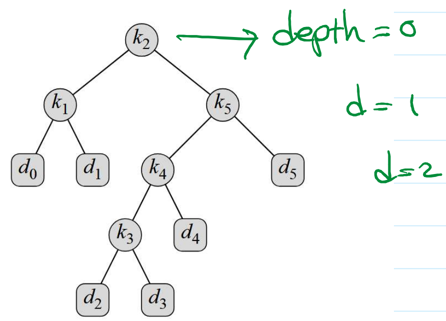
for some $c>0$

$$\leq c \cdot mn + k \cdot i \cdot \frac{n}{2} + k(m-i)\frac{n}{2}$$
$$= c \cdot mn + kmn/2$$
$$= \left(c + \frac{k}{2}\right)mn$$
$$\leq kmn \qquad \text{if } k \geq 2c.$$

Optimal Binary Search Tree

- Suppose we have a seq. of n distinct keys $k_1 < k_2 < \dots < k_n$ that we want to store in a BST.
- Each key k_i has probability/frequency P_i
- We may search for values (dummy keys not in the table), say $d_0 < k_1 < d_1 < k_2 < d_2 \dots < k_n < d_n$





From CLRS

- Suppose searching for any dummy key d_i has frequency (probability) q_i;
- We have $$\sum_{i=1}^{n} P_i + \sum_{i=0}^{n} q_i = 1$$
- let $d_T(k_i)$ for a tree T : depth of key k_i

- Expected cost of search in a tree T:

$$E[\text{cost of } T] = \sum_{i=1}^{n} (d_T(k_i)+1)\, p_i + \sum_{i=0}^{n} (d_T(d_i)+1)\, q_i$$

$$= 1 + \sum_{i=0}^{n} d_T(k_i)\, p_i + \sum_{i=0}^{n} d_T(d_i) \cdot q_i$$

Goal: Given k_i, p_i, q_i build a tree T with optimum (minimum) expected search cost.

Observation: if a subtree of T contains keys $k_i \cdots k_j$ then this subtree must be optimum for these keys.

- Say $O[i,j]$ stores the exp. cost of search for an opt. BST over keys $k_i \cdots k_j$.
- Also let $f[i,j] = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$

- if opt. BST for $k_i \cdots k_j$ is rooted at k_r then

$$O[i,j] = p_r + \left(O[i,r-1] + f[i,r-1]\right) + \left(O[r+1,j] + f[r+1,j]\right)$$

$$= O[i,r-1] + O[r+1,j] + f[i,j]$$

Since we don't know the value for r:

$$O[i,j] = \begin{cases} q_{i-1} & j = i-1 \\ \min\limits_{i \le r \le j} \left\{ O[i,r-1] + O[r+1,j] + f[i,j] \right\} & \end{cases}$$

- We can compute $f[i,j]$ (using DP) in $O(n^2)$ time:

$$F[i,j] = \begin{cases} q_{i-1} & j = i-1 \\ F[i,j-1] + p_i + q_i & j \geq i \end{cases}$$

Opt-BST-DP $(\vec{p}, \vec{q}, n)$

for $i \leftarrow 1$ to $n+1$ do

  $O[i,i-1] = q_{i-1}$

  $F[i,i-1] = q_{i-1}$

for $\ell \leftarrow 1$ to n do → subproblem size $j-i+1$

  for $i \leftarrow 1$ to $n-\ell+1$ do

    $j = i+\ell-1$

    $O[i,j] \leftarrow \infty$; $F[i,j] \leftarrow F[i,j-1] + p_j + q_j$

    for $r \leftarrow i$ to j do

      $temp \leftarrow O[i,r-1] + O[r+1,j] + F[i,j]$

      if $temp < O[i,j]$ then

        $O[i,j] \leftarrow temp$

        $root[i,j] \leftarrow r$

return O and r

Time complexity: $O(n^3)$

Memoized (recursive) implementation:

OPT-BST (i,j):

  if $O[i,j]$ calculated $\neq \infty$ return it for $r \leftarrow i$ to j do

temp $\leftarrow$ OPT_BST$(i, r-1)$ +

OPT_BST$(r+1, j)$ + $F[i,j]$

if temp < $O[i,j]$ then

$O[i,j] \leftarrow$ temp

return $O[i,j]$

Improving time using Monotonicity

In BST example, we use root$[i,j]$ to store the index of root of $O[i,j]$.

Knuth '71 proved the following monotonicity:

$$\forall i,j \quad \text{root}[i,j-1] \leq \text{root}[i,j] \leq \text{root}[i+1,j]$$

i.e. every row & column in root$[\cdot,\cdot]$ is sorted.

– This helps to improve time complexity:

Faster_Opt_BST (p, q, n)

Compute $F[\cdot,\cdot]$

for $i \leftarrow 1$ to n do

$O[i, i-1] \leftarrow q_{i-1}$

root$[i, i-1] \leftarrow i$

for $\ell \leftarrow 0$ to n do

for $i \leftarrow 1$ to $n-\ell+1$ do

Compute_OPT $(i, i+\ell)$

return $O[1, n]$

Compute_OPT (i, j)

$O[i,j] \leftarrow \infty$

for $r \leftarrow$ root $[i, j-1]$ to root $[i+1, j]$ do

$temp \leftarrow O[i, r-1] + O[r+1, j]$

if $O[i,j] > temp$ then

$O[i,j] \leftarrow temp$

$O[i,j] \leftarrow O[i,j] + F[i,j]$

Lemma: Index r increases monotonically from 1 to n during each iteration of the outermost loop of Faster_OPT_BST.

Corollary: Total time complexity is $O(n^2)$.

Note: Hu & Tucker improved this to $O(n \log n)$.

Total monotonicity

The monotonicity property can help in many situations.

Finding row minimums: Suppose we are given a matrix $M[1..n, 1..m]$ & we want to find the minimum element in each row.

Monotone: We say M is monotone if the leftmos smallest element in any row is to the right of leftmost smallest element of prev. rows.

$$\begin{bmatrix} 12 & 21 & 38 & 76 & 27 \\ 74 & 14 & 14 & 29 & 60 \\ 21 & 8 & 25 & 10 & 71 \\ 68 & 45 & 29 & 15 & 76 \\ 97 & 8 & 12 & 2 & 6 \end{bmatrix}$$

From JE

if $LM[i]$ is the index of the leftmost smallest in row i then: $LM[i] \leq LM[i+1] \quad \forall i$

- Computing the minimum of each row naively: $O(mn)$

Faster algorithm if M is Monotone:

- compute $LM[i]$ for all odd rows recursively
- for each even row $2i$ we have:

$$LM[2i-1] \leq LM[2i] \leq LM[2i+1]$$

- Compute $LM[2i]$ by checking indices $LM[2i-1] \ldots LM[2i+1]$

total time to compute even rows:

$$\sum_{i=1}^{m/2} LM[2i+1] - LM[2i-1] = LM[m+1] - LM[1] \leq n$$

$\longrightarrow$ time for even rows $O(n+m)$

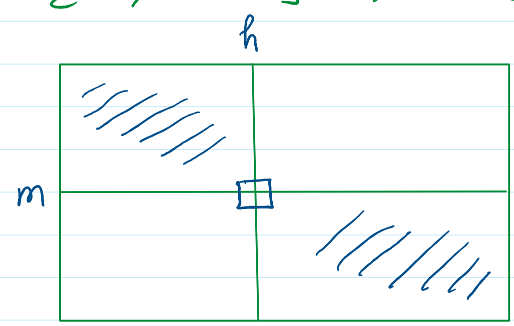
$\longrightarrow$ Time satisfies: $T(m,n) = T(\frac{m}{2}, n) + O(n+m)$

$\longrightarrow T(m,n) = O(m + n\log m)$

We can use a D&C algorithm like Hirschberg's.

- Compute $h = LM[m/2]$ in $O(n)$ time
- recursively compute the leftmost minimum in

$$M[1 \ldots \tfrac{m}{2}-1, 1 \ldots h] \quad , \quad M[\tfrac{m}{2}+1, \ldots, m, h \ldots n]$$



- Worst Caste time:

$$T(m,n) = \begin{cases} 0 & m < 1 \\ O(n) + \max\limits_{k} \{ T(\frac{m}{2}, k) + T(\frac{m}{2}, n-k) \} \end{cases}$$

- The tree for this recursion has depth $\log m$ & $O(m)$ nodes
- # of comparisons at each level : $O(n)$

$\longrightarrow$ time $O(m + n\log m)$

Total Monotonicity : SMAWK algorithm
(Aggrawal, Klawe, Moran, Shor, Wilber '87)

We say M is totally monotone if every subarray is monotone (in particular every 2×2 submatrix is monotone).

$$\begin{bmatrix} 12 & 21 & 38 & 76 & 27 \\ 74 & 14 & 14 & 29 & 60 \\ 21 & 8 & 25 & 10 & 71 \\ 68 & 45 & 29 & 15 & 76 \\ 97 & 8 & 12 & 2 & 6 \end{bmatrix}$$

grey submatrix is not monotone

$\longrightarrow$ not totally monotone

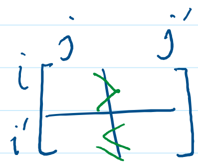
Monge property: matrix M has Monge property if

$\forall\ i < i',\ j < j'$: $\quad M[i,j] + M[i',j'] \leq M[i,j'] + M[i',j]$

lemma: Every Monge matrix is totally monotone.

Proof: if M is not totally monotone $\longrightarrow$

$\exists$ 2x2 submatrix that is not monotone



$M[i,j] > M[i,j']$ and $M[i',j] \leq M[i',j']$

$\longrightarrow$ together contradict Monge property. ∎

Note: Many matrices have this property

Overview of SMAWK algorithm:

– has two main procedures to reduce problem size.

* sparsify

if the current matrix M has $m > n$ (tall) then uses algorithm sparsify by finding minimum of odd rows (as before) recursively and then using monotonicity and $O(m+n)$ time find minimum of even rows.

* Reduce

if $m<n$ (more columns than rows) then tries to identify a subset of m columns that are guaranteed to have the minimum of the m rows. let M' be the Submatrix of there m rows & column. (in time $O(m+n)$) The problem is "Reduce"d to M'. We solve the problem on M'.

— So at each step:

$\begin{cases} \text{– either \# of rows is halved} : m \to \frac{m}{2} \\ \text{– or we change \# of columns to be } m \end{cases}$

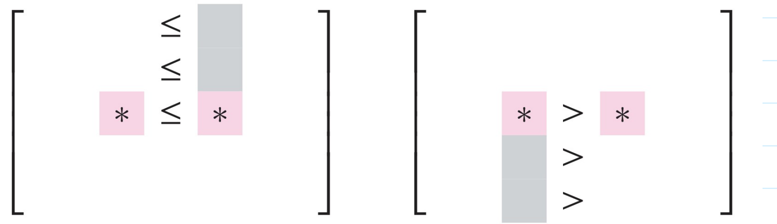
Time complexity will be:

$$T(m,n) \leq \begin{cases} O(m+n) + T(\frac{m}{2}, n) & \text{if } m \geq n \\ O(m+n) + T(m,m) & m<n \end{cases}$$

$\longrightarrow$ total time $O(m+n)$

How does Reduce work?

Observation: For any row i and column $p<q$



From JE

- if $M[i,p] \le M[i,q] \longrightarrow$ for all prev. rows $h \le i$

  $M[h,p] \le M[h,q]$; so $LM[h] \ne q$

- if $M[i,p] \ge M[i,q] \longrightarrow$ for all larger rows $j \ge i$

  $M[j,p] > M[j,q]$; so $LM[j] \ne p$

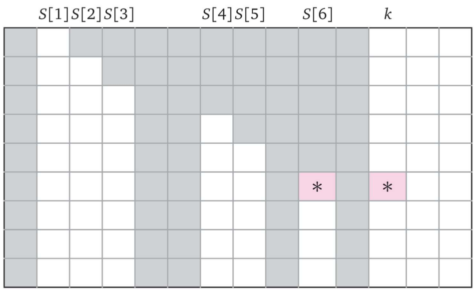
- We say $M[i,j]$ is dead : we know $LM[i] \ne j$

  in the above picure, gray cells are dead.

- the algorithm Reduce (for case $m<n$) maintains a stack $S[1..m]$ of indices of columns

  (# of columns indices will be $\le$ # of rows).

- Goal: At the end $M'_{m \times m}$ will be base on S
- At any time, t : # of items in stack
- columns are considered left to right $1 \le k \le n$
- For each column k it might pop several columns from S (or none) and then push k



From JE

- Algorithm maintains three invariants
  - $S[1..t]$ are sorted (increasing)
  - For all $1 \le j \le t-1$, top $j-1$ entries of $S[j]$ are dead
  - if $j < k$ and j is not on the stack all column j is dead

Reduce($M[1..m, 1..n]$)

$t \leftarrow 1$; $S[1] \leftarrow 1$

for $k \leftarrow 1$ to n do

while $t > 0$ and $M[t, S[t]] \ge M[t, k]$

$t \leftarrow t-1$; <pop>

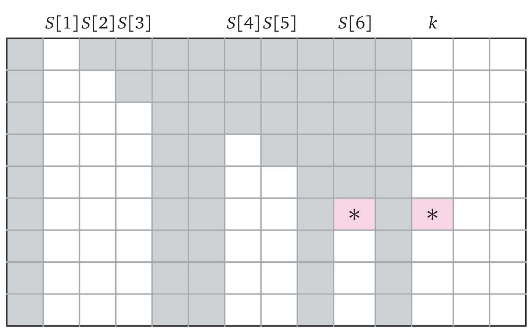
if $t < m$ then

$t \leftarrow t+1$

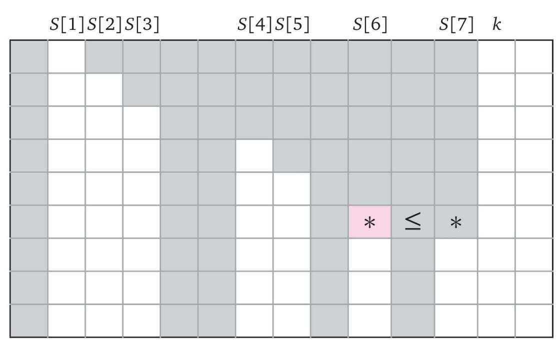
$S[t] \leftarrow k$ <push k>

return S

$t=6$

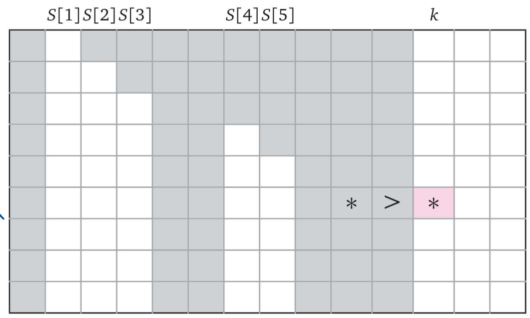


$t=7$



if $M[t, S[t]] \le M[t, k]$: we can push k to $S[\cdot]$

$t = \cancel{4}$ 5



From JE

— if $M[t, S[t]] > M[t, k]$

everthing below $M[t, S[t]]$ is dead; pop