Advanced Dynamic Programming

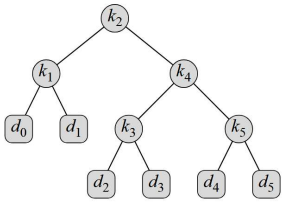
- Recall the example of optimum BST from last lecture.
- Given keys $k_1 \dots k_n$ with frequencies $p_1 \dots p_n$ and also for dummy keys frequencies $q_1 \dots q_n$

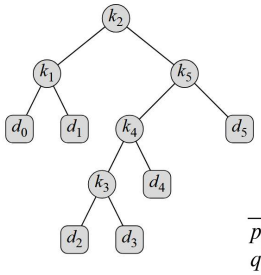
$$d_0 < k_1 < d_1 < k_2 < \dots < k_n < d_n$$

Goal: Find a BST which minimizes expected search cost:

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

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





i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

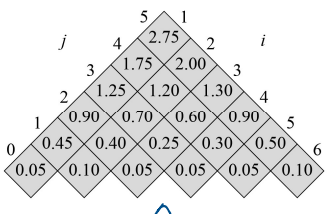
- if opt. BST for $k_i \dots k_j$ is rooted at k_r then

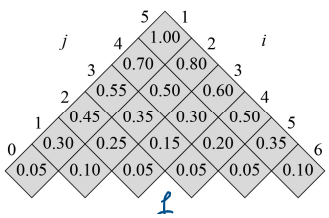
$$O[i,j] = p_r + (O[i,r-1] + f[i,r-1]) + (O[r+1,j] + f[r+1,j])$$

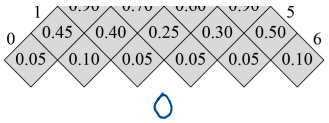
$$= O[i,r-1] + O[r+1,j] + f[i,j]$$

$r=1 \quad r=2$

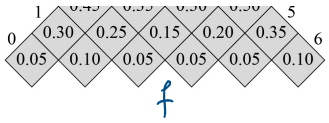
$$O[1,2] = \min \{ O[1,0] + O[2,2] + f[1,2],\ O[1,1] + O[3,2] + f[1,2] \}$$



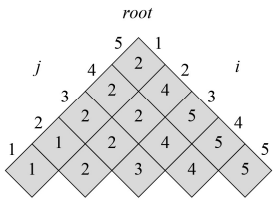




O



f



$O[i,j] + O[s,t] + f[l,r]$

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

- Time complexity to compute the whole table:

  $(O(n^2)$ entries$) \times (n$ per entry$) = \Theta(n^3)$

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:

$$\forall\, i,j \qquad root[i,j-1] \le root[i,j] \le root[i+1,j]$$

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[·,·]
    for i ← 1 to n do
        O[i, i-1] ← q_{i-1}
        root[i, i-1] ← i
    for l ← 0 to n do
```

```
for i ← 1 to n-l+1 do
    Compute_OPT (i, i+l)
return O[1,n]
```

```
Compute_OPT (i, j)
    O[i,j] ← ∞
    for r ← root[i,j-1] to root[i+1,j] do
        temp ← O[i,r-1] + O[r+1,j]
        if O[i,j] > temp then
            O[i,j] ← temp
    O[i,j] ← 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 $O(n \log n)$.

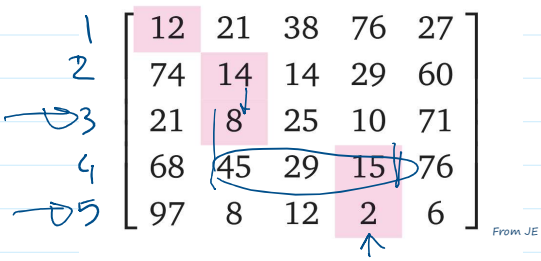
Total monotonicity

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 leftmost smallest element in any row is to the right of leftmost smallest element of prev. rows.



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 naïvely: $O(mn)$

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 $O(n+m)$

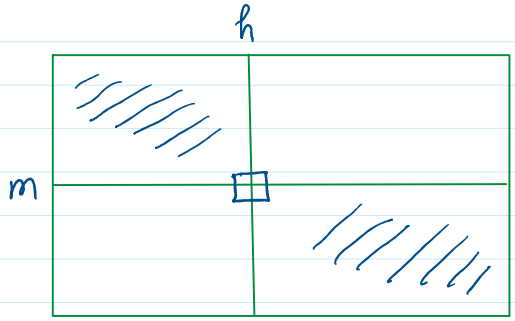
→ Time satisfies : $T(m,n) = T(\frac{m}{2}, n) + O(n+m)$

→ $T(m,n) = O(m + n \log m)$

We can use a D&C algorithm like Hirschbergs.

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

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



- Worst Caste time:

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

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

→ 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 2x2 submatrix is monotone).

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

grey submatrix is not monotone

→ not totally monotone

Monge property: matrix M has Monge property if

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

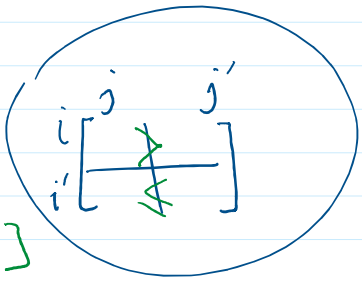
lemma: Every Monge matrix is totally monotone.

proof: if M is not totally monotone →

∃ 2x2 submatrix that is not monotone

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

→ together contradict Monge property. X



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

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:

- either # of rows is halved: $m \to \frac{m}{2}$
- or we change # of columns to be m

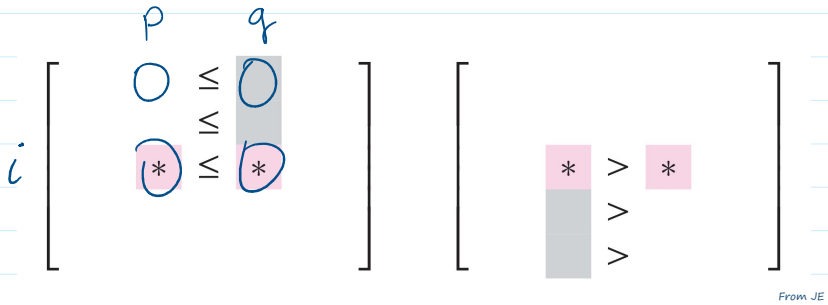
Time complexity will be:

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

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

How does Reduce work?

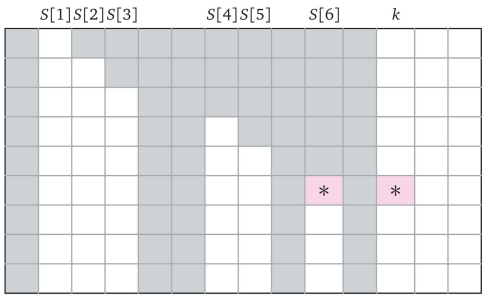
Observation: For any row i and column $p < q$



From JE

- if $M[i,p] \leq M[i,q]$ → for all prev. rows $h \leq i$
  $M[h,p] \leq M[h,q]$; so $LM[h] \neq q$
- if $M[i,p] \geq M[i,q]$ → for all larger rows $j \geq i$
  $M[j,p] > M[j,q]$; so $LM[j] \neq p$

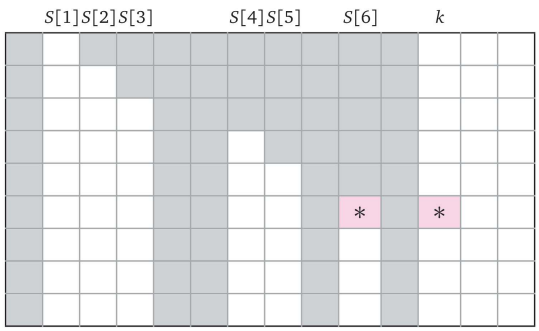
- we say $M[i,j]$ is dead : we know $LM[i] \neq j$
  in the above picure, gray cells are dead.
- the algorithm Reduce (for case $m<n$) maintains a stack $S[1..m]$ of indices of columns
  (# of columns indices will be ≤ # of rows).
- Goal: At the end $M'_{m \times m}$ will be base on S
- At any time, t : # of items in stack
- columns are considered left to right $1 \leq k \leq n$
- For each column k it might pop several columns from S (or none) and then push k

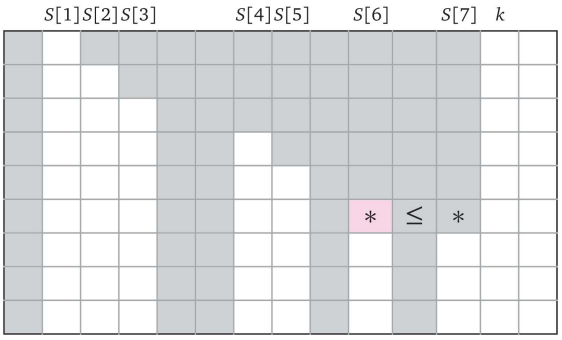


From JE

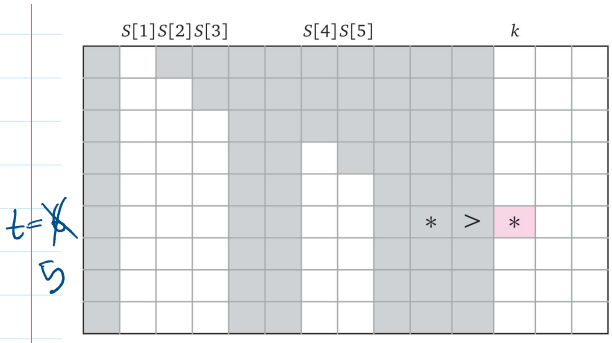
- Algorithm maintains three invariants
  - $S[1..t]$ are sorted (increasing)
  - For all $1 \le j \le t-1$, top $j-1$ entries of $S[j]$ are dead
  - if $j < k$ and j is not on the stack all column j is dead

```
Reduce(M[1..m, 1..n])
    t ← 1 ; S[1] ← 1
    for k ← 1 to n do
        while t > 0 and M[t, S[t]] ≥ M[t, k]
            t ← t − 1 ;  <pop>
        if t < m then
            t ← t + 1
            S[t] ← k        <push k>
    return S
```





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



From JE

— if $M[t, S[t]] > M[t, k]$

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