Monday, July 6, 2020 6:29 PM

Longest increasing subseq (LIS)

- single seq of number

A = [5, 1, 3, 2, 7, 9]

inc. sub. seq. [5, 7, 9] ✓

[1, 3, 7, 9] ✓

[5, 2, 7] ×

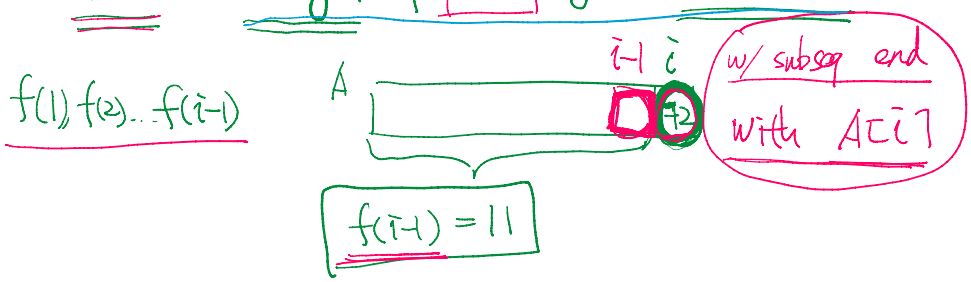
LIS of A [1, 2, 7, 9]

or [1, 3, 7, 9]

- output: length of LIS

---

f(i) := length of LIS given A[1...i]



① if A[i-1] < A[i]

f(i) can be f(i-1)+1

② if A[i-1] > A[i]

f(i-1) is useless

$f(i) = \max\{$ f(1)+1 if A[1] < A[i]; f(2)+1 if A[2] < A[i]; ... f(i-1)+1 if A[i-1] < A[i] $\}$

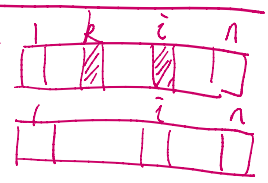
$\max\{\max\{f(k)+1\} \mid 1\}$

$\max\{\max_{k=1\ldots i-1}\{f(k)+1\}, 1\}$

A[1] A[2] A[i+1]

$f(0)=0$ $f(1)=1$

LIS (A[1 ... n]) {



dp[1] = 1;

for (i = 2, ..., n) {

dp[i] = 1;

for (k = 1 ... i-1) {

if (A[k] < A[i]) {

dp[i] = max(dp[i], dp[k]+1);

}

}

return max{dp[]};

}

r.t. $O(n^2)$

space $O(n)$

A

LIS is similar to LCS

B

reduction

LIS / A $\Rightarrow$ [A, B = sort(A)] $\Rightarrow$ LCS

$O(n \lg n)$ $O(n^2)$

---

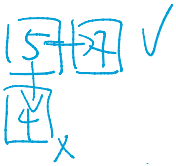
given matrix of positive integer

A =

5	1	2
3	7	1
4	5	6

- start anywhere on A
- 

inc path

inc. path  ✓

inc path

- Q: length of longest path?

$f(i,j)$ = Long inc. path that end with position (i,j)

$f(i,j) = \max$
① if $A[i,j-1] < A[i,j]$ → $f(i,j-1)+1$
② if $A[i-1,j] < A[i,j]$ → $f(i-1,j)+1$
③ 1

```
LIP(A[,]) {
  for(i=2... n){
    if(A[i-1,1]<A[i,1]){
      dp[i,1]=dp[i-1,1]+1;
    }else{
      dp[i,1]=1
    }
  }
  for(j=2 ... m){
    if(A[1,j-1]<A[1,j]){
      dp[1,j]=dp[1,j-1]+1;
    }else{
      dp[1,j]=1
    }
  }
  for(i=2 ... n){
    for(j=2 ... m){
      dp[i,j]=1;
      if(A[i-1,j]<A[i,j])
```

① fill the 1st col in dp

LIS

② f(1)

fill the 1st row in dp.

③ fill rest

```
if (A[i-1,j] < A[i,j])
    dp[i,j] = max(dp[i,j], dp[i-1,j]+1)
if (A[i,j-1] < A[i,j])
    dp[i,j] = max(dp[i,j], dp[i,j-1]+1)
```

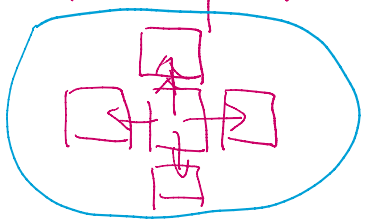
fill rest of dp

```
    }
}
return max{dp[]}
}
```

(A) easy (B) okay (C) hard (?)

```
for (i = 1 ... n) {
    for (j = 1 ... n) {
        dp[i,j] = 1;
        if (i-1 >= 1 && A[i-1,j] < A[i,j])
            dp[i,j] = max(dp[i,j], dp[i-1,j]+1);
        if (j-1 >= 1 && A[i,j-1] < A[i,j])
            dp[i,j] = max(dp[i,j], dp[i,j-1]+1)
    }
}
return max(dp[]);
```

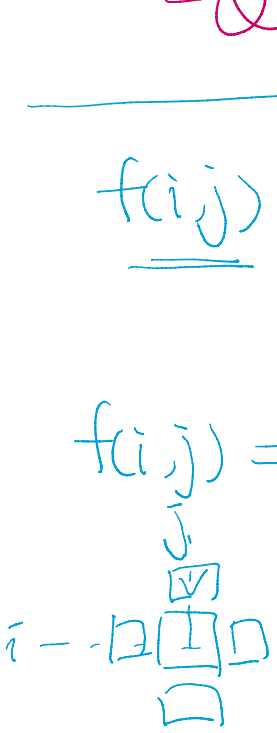
- given matrix A (positive integer)
- inc path

- inc path
- 
- can choose any starting position
- Q: length of LIP

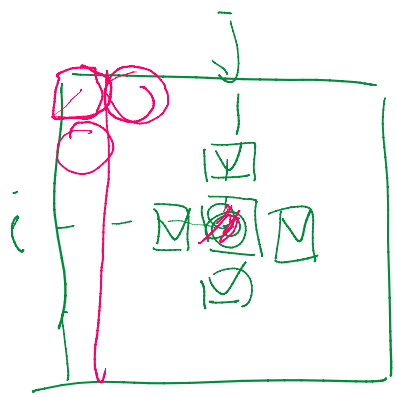
---

f(i,j) : length of LIP ended w/ A[i,j]

$$f(i,j) = \begin{cases} 1 & \\ f(i-1,j)+1 & \text{if } A[i-1,j] < A[i,j] \\ f(i,j-1)+1 & \text{if } A[i,j-1] < A[i,j] \\ f(i+1,j)+1 & \text{if } A[i+1,j] < A[i,j] \\ f(i,j+1)+1 & \text{if } A[i,j+1] < A[i,j] \end{cases}$$



bottom up vs top down ??



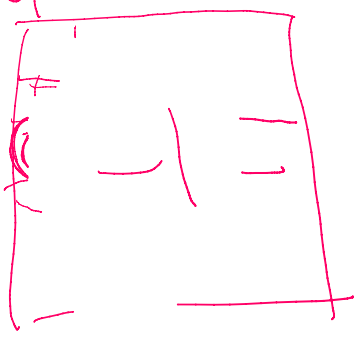
```
LIP(A[ ], dp[ ], i, j)
if (dp[i,j] ≠ -1)
    return dp[i,j];
```

f(i,j)

return dp[i][j];

```
val = 1;
if (i-1 >= 1 && A[i-1,j] < A[i,j])
    val = max(val, LIP(A, dp, i-1, j));
if (j-1 >= 1 && A[i,j-1] < A[i,j])
    val = max(val, LIP(A, dp, i, j-1));
if (i+1 <= n ...)
if (j+1 <= m ...)
dp[i][j] = val;
return val;
}
```

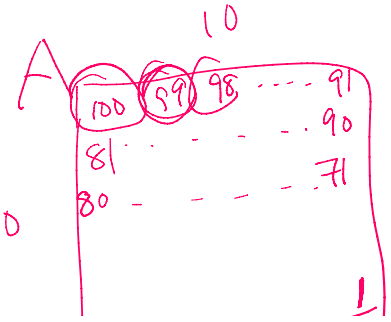
dp



```
for (i = 1 ... n)
    for (j = 1 ... m)
        LIP(A, dp, i, j)
return max(dp[])
```

running time -

O(nm)

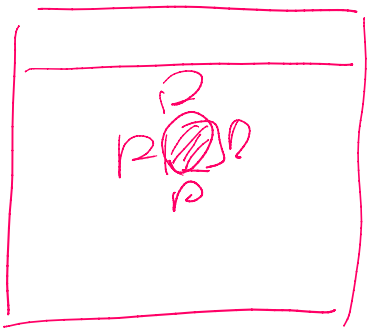


space O(nm)

---

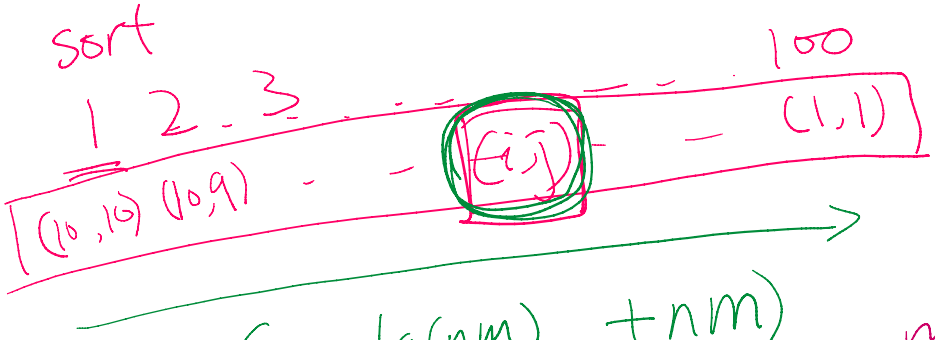
bottom up?? Yes

dp



- find minimun

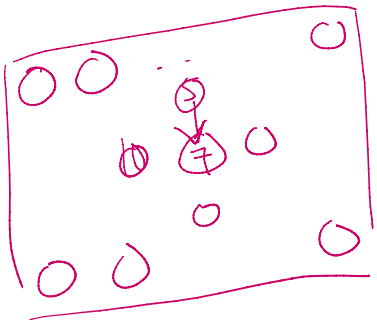
- sort



O(nmlg(nm) + nm)

- topological sort

O(nm)



- 4 steps
- recursion design
- bottom up vs top down

- 2 seq (LCS, Ed)
- 1 seq (palindrom, LIS)
- knapsack
- interval (matrix multiplication)

- bottom up vs top down
  - interval (matrix multiply)
  - position (<IP)

Graph algorithms

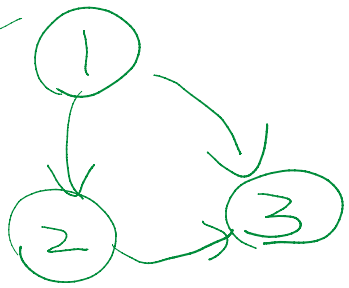
- graph search (bfs, dfs) } directed
- shortest path
- mst — undirected

Graph $G = (V, E)$

- V: set of vertices
- E: set of edge.

{ directed graph
undirected graph

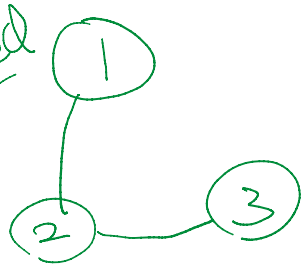
directed



$V = \{1, 2, 3\}$

$E = \{(1,2), (2,3), (1,3)\}$

undirected



$V = \{1, 2, 3\}$

$E = \{ (1,2), (2,1)$

$(2,3), (3,2) \}$

$|V| = n \quad |E| = m$

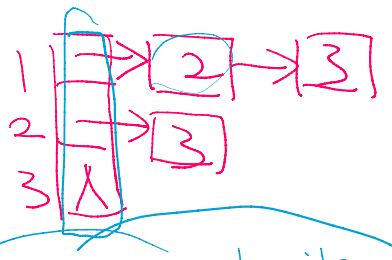
$|V| = n \quad |E| = m$

$O(n+m) \quad O(|V|+|E|)$

---

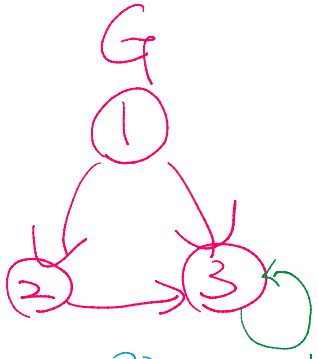
graph representation

adj. list



space complexity

$O(n+m)$



$O(n^2)$ self loop

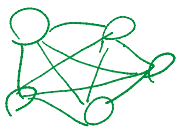
adj matrix

A	1	2	3
1	T/F	T	T
2	F	T/F	T
3	F	F	T/F

$A[i,j] = \begin{cases} T & (i,j) \in E \\ F & \text{o/w} \end{cases}$

density

- sparse graph → social network
- dense graph → complete graph



- run time for operation

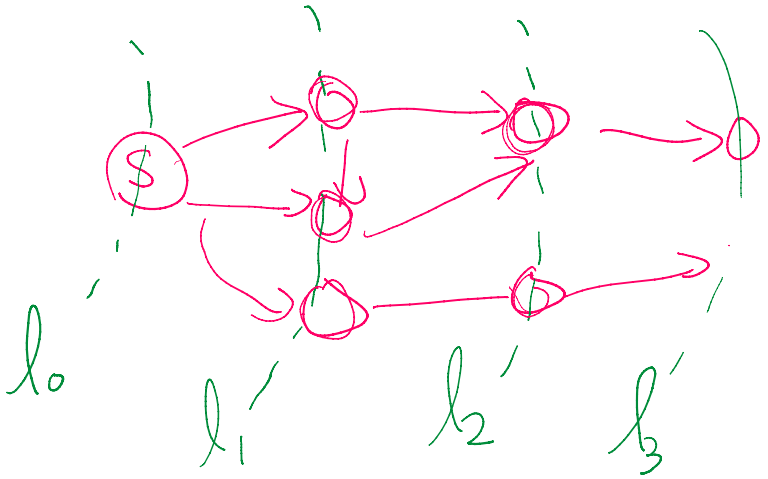
① given vertex u, traverse all its neighbors

② $(u, v) \in E$?

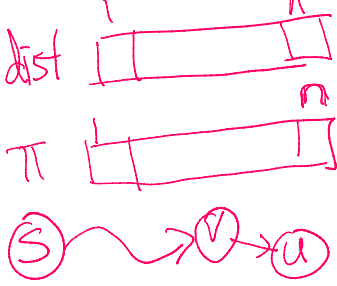
BFS (Breath First Search)

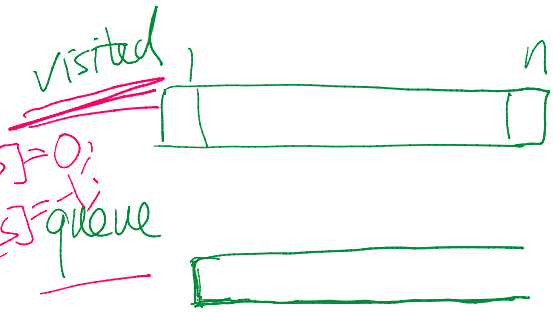
- given $G = (V, E)$. starting point S

- given $G=(V,E)$, starting point S
- visit all reachable vertice "level by level"



```
BFS(G, s) {
    q.enqueue(s);
    visited[s] = True;
    while (q.empty() == False) {
        u = q.dequeue();
        print u;  // processing.
        for (v in G.neighbors(u)) {
            if (visited[v] == False) {
                q.enqueue(v);
                visited[v] = True;
                dist[v] = dist[u] + 1;
                π[v] = u;
```





array

vs

queue

FIFO

insert

get first

delete first

BSF ⟹ shortest path.

BFS ⟹ ① process reachable vertices in some order

② shortest distant

③ shortest path