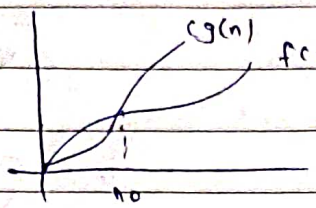
PAGE No
DATE / /

DOA Week 2 Notes



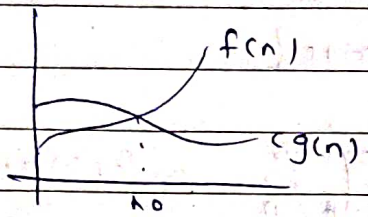
worst case

For all $n \geq n_0$

$0 \leq f(n) \leq c \cdot g(n)$

$f(n) = O(g(n))$

i.e $f(n) \in O(g(n))$

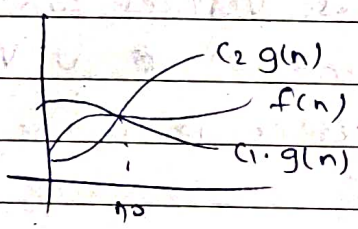


Best case

For all $n \geq n_0$

$0 \leq c \cdot g(n) \leq f(n)$

$f(n) = \Omega(g(n))$



For all $n \geq n_0$

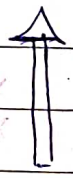
$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

$f(n) = \Theta(g(n))$

Exponential growing

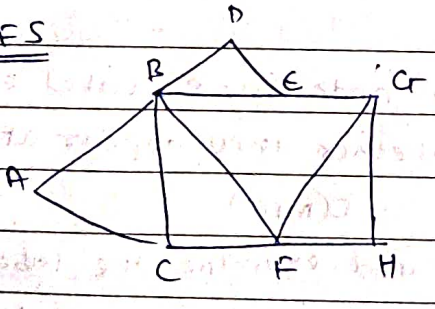
polynomial

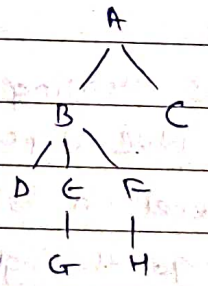
logarithmic



Fastest growing

BFS



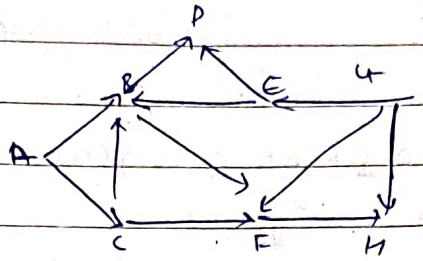


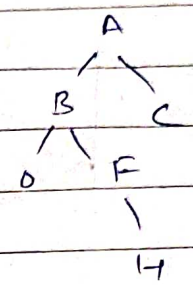
m: no. of edges

n: no. of nodes

$O(m+n)$

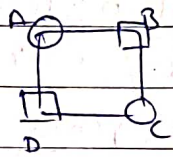
DFS

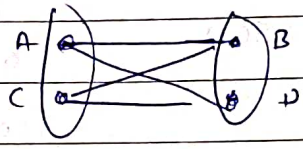




O(m+n)

Determine if graph is bipartite?

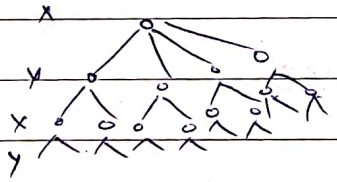




Bipartite ✓

Bipartite graph: Graph whose vertices can be divided into 2 sets (disjoint and independent) U, V ; such that every edge connects a vertex U to one in V.

If a graph G is bipartite it cannot have a odd cycle.



Find if bipartite

Method 1

- Run BFS starting from any node say S : Label each node Red or Blue depending on whether they appear at an odd or even level on BFS tree O(m+n)
- Then go through all edges and examine the labels at the two ends of the edge. If all edges have a red end and a blue end, then graph is bipartite. O(m)

O(m+n)

PAGE No.
DATE / /

(*) Directed graph is strongly connected if there is a path From any point to any other point in the graph.

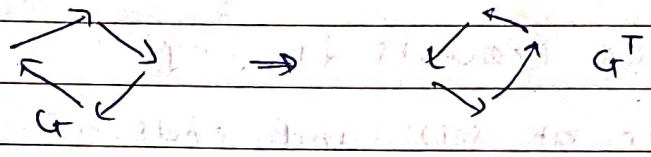
Find if DG is strongly connected?

Method 1

Run BFS/DFS 'n' times once for each node

$$O(n(m+n)) = O(n^2+mn)$$

Method 2



use BFS/DFS to Find reachable nodes from any point s. $O(m+n)$

IF some nodes not reachable:

return False.

else:

Find G^T $O(m+n)$

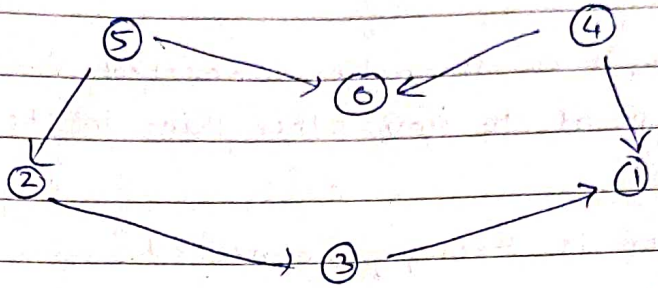
use BFS/DFS to Find all reachable nodes from point s in G^T $O(m+n)$

$O(m+n)$

(*) Topological sorting

Topological sorting for DAG is a linear ordering of vertices such that for every edge (u, v) vertex u comes before v in the ordering

DAG may have many topological sorting



"542310"

"452310"

To Find longest path in DAG Topological ordering is used.

(*) Interval scheduling problem.

Input: set of requests $\{1 \ldots n\}$

i^{th} req starts at s(i) and ends at f(i)

objective: To Find the largest compatible subset of these requests.

Smallest Finish Time First

Algo

R ← Complete set of requests

A ← empty # answer

While R is not empty:

choose a request $i \in R$ that has smallest Finish time

Add req 'i' to A

Delete all req from R that are not compatible with i

Return A.

PAGE No.
DATE / /

Proof of correctness

(1) Show that A is a compatible set
(2) Show that A is optimal set

Algo we have written select req 'i' and removes all incompatible req.
Hence A is compatible set.

optimality proof

Say A is of size k
Say there is an optimal solution O

Req in A: $i_1, i_2, \ldots = i_k$

Req in O: $j_1, j_2, \ldots \ldots j_k$

For all $r \leq k$

$$f(i_r) \leq f(j_r)$$

As our algo is always choosing arc to earliest Finish time

our schedule i_1 will end before or on same time as that of j_1

Similarly For i_2 j_2

i_n j_n

i_1 i_2 $\ldots$ i_j

PAGE No.
DATE / /

i_1 i_2 . . . i_{k-1} i_k

j_1 j_2 . . . j_{k-1} j_k . .

Here i_{k-1} finishes at same time or before j_{k-1}

$\therefore$ i_{k-1} could choose i_k or j_k

But it choose i_k

$\therefore$ ~~If~~ Finish-time(i_k) $\leq$ Finish Time(j_k)

Now To prove $|A| = |O|$

i.e our soln and optimal soln has same no. of schedules

A . . . i_k

O . . . . j_k j_{k+1}

i_k finishes before or at same time of j_k

~~If 'O' ...~~

our algo is not able to fit 'k+1' in schedule

So Algo 'O' Finishing schedule K after our schedule 'K' cannot fit 'K+1'

Hence $|A| = |O|$

Hence A i.e our Algorithm is optimal.

PAGE NO.
DATE / /

(Fractional)

Algo

- Sort req acc to finish Time $O(n \log n)$

- select req in order of inc f(i), always selecting the first.
Then iterate through the intervals in this order until reaching the first interval for which

$$s(j) \geq f(i) \qquad O(n)$$

$O(n \log n)$

Fractional knapsack

knapsack weight → w

Given: set of n objects with their weight and valu

Objective: To fill knapsack to its weight capacity such that the value of items in knapsack is maximized.