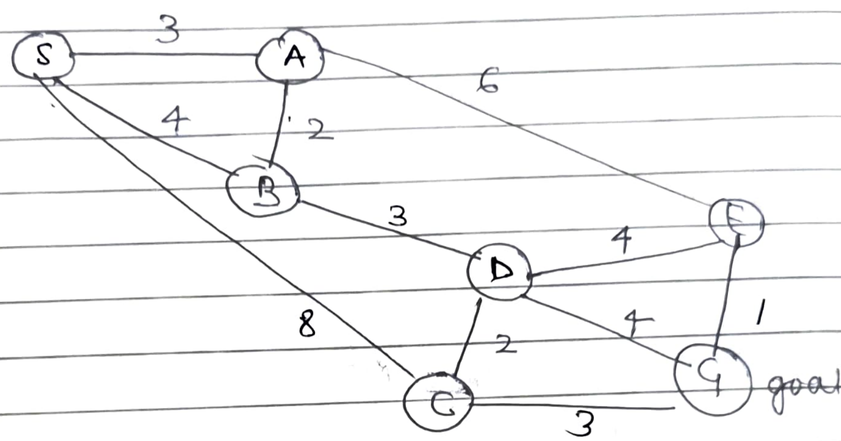
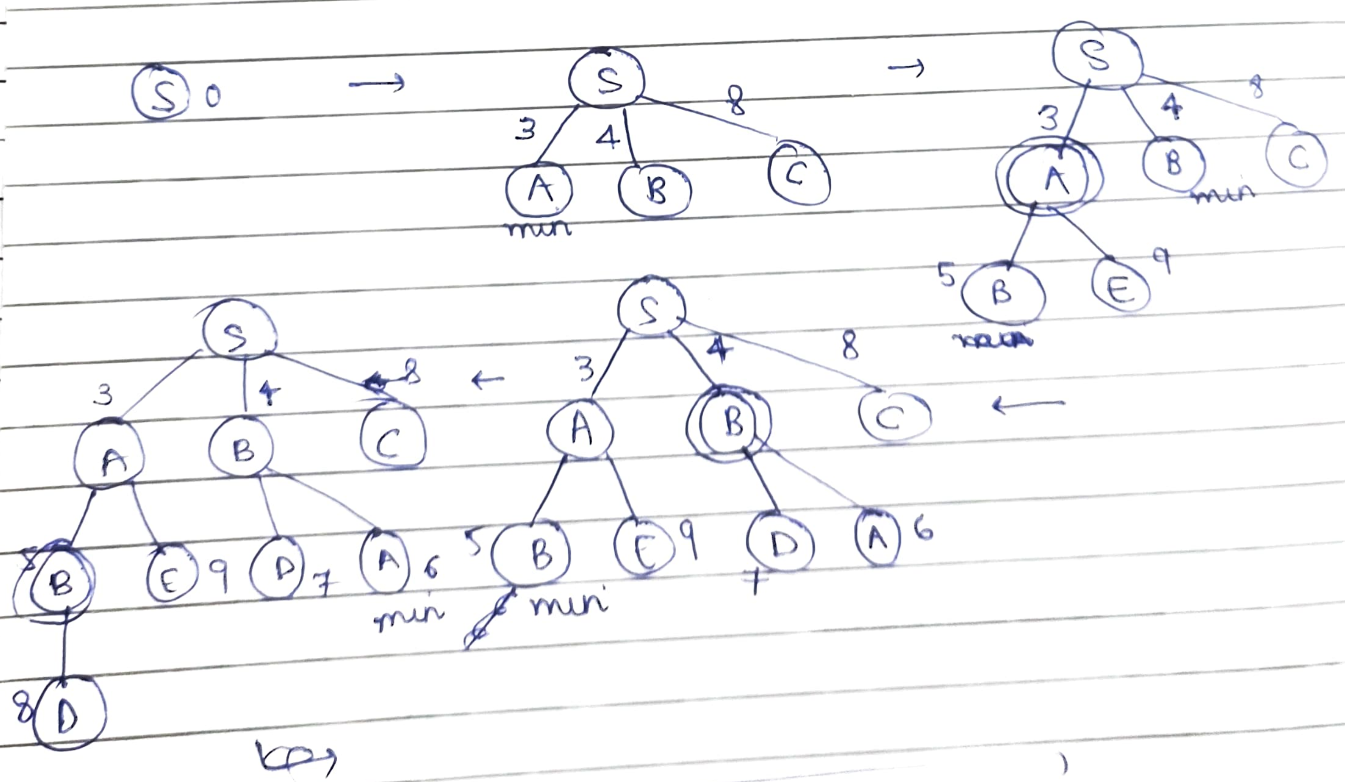
UNIT 5 : Finding optimal paths

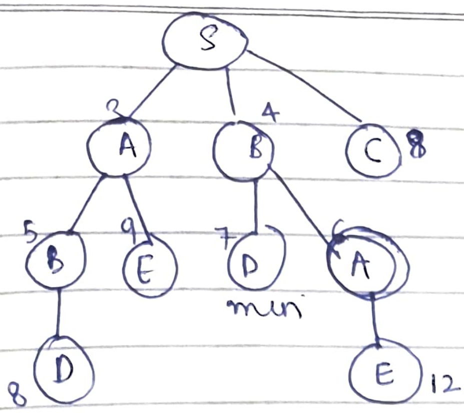
1. BRANCH & BOUND

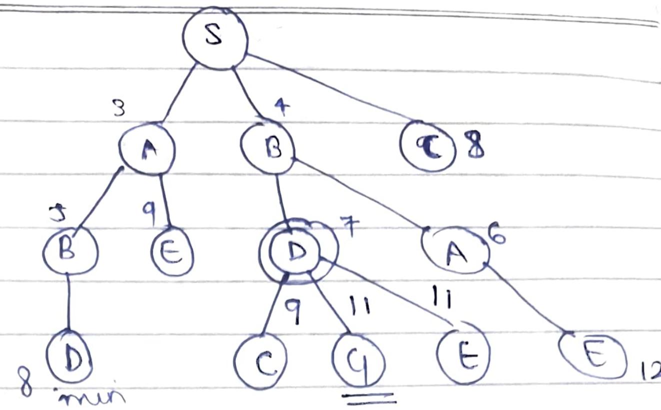
- Branching is the process of generating subproblems
- Bounding refers to ignoring ~~solns~~ partial solns that cannot be better than the current best solution.
- It is a search procedure to find the optimal soln
- It eliminates those parts of a search space which do not contain better soln (pruning)
- We basically extend the cheapest partial path

eg.

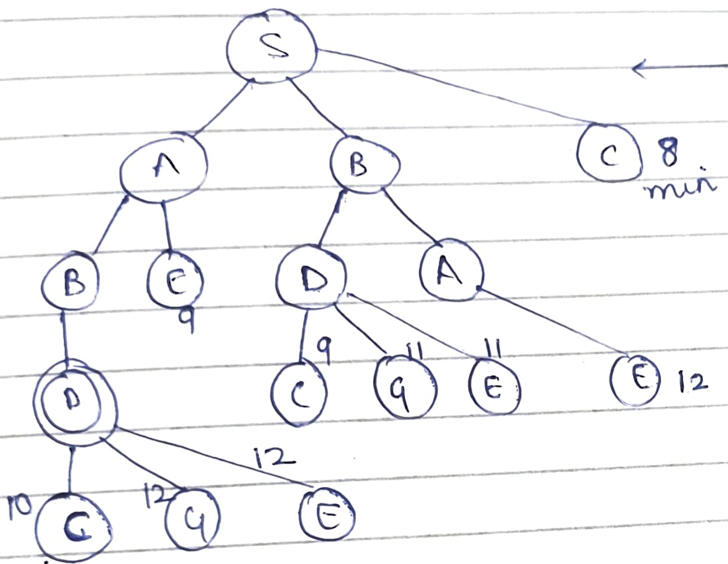


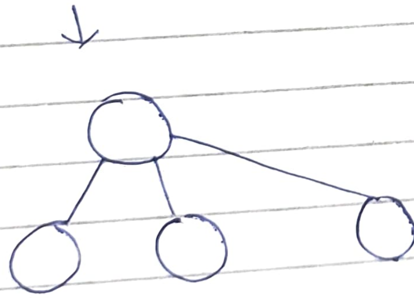






∴ current min path = 11
We still continue to expand





continue so on & so forth
till we find min path
(optimal)

~~2. REFINEMENT SEARCH~~

2. DIJKSTRA'S ALGORITHM

3. A* :

- Combines best features of B&B, Dykstra, BFS
- We use an evaluation function $f(n) = g(n) + h(n)$ heuristic fxn

estimated cost of path from start to goal via node n

$g(n)$: ~~estimated~~ know cost from S to n

$h(n)$: estimated n to G

- Also, $f^*(n) = g^*(n) + h^*(n)$

cost of optimal path — optimal cost from S to n — optimal cost from n to G.

~~$g^*(n) \geq g(n)$, underestimation~~ ~~$h^*(n) \leq h(n)$~~

- Algo
  - → Enter Starting node in OPEN list
  - → If OPEN list is empty, return fail
  - → Select node from OPEN w/ min (g+h) value
    - ↳ if node = GOAL, return success
  - → Expand node n & generate all successors
    - ↳ compute (g+h) for each successor
  - → If node n is already in OPEN/CLOSE, attach to back pointer
  - → Return to

- $g(n) \geq g^*(n)$ overestimation

$h(n) \leq h^*(n)$ underestimation

If the underestimation condn is true, A* is said to be admissible.

1. complete
2. TC, 3. SC } exponential ~~O(b^m)~~
4. Quality: it finds the optimal path when an admissible hf is used.

admissible heuristic

4. ITERATIVE DEEPENING A*

- Extension of DFID
- IDA* : A* = DFID : DFS
- IDA* explores children by logical level of g+h* in a depth first way.
- The path w/ the typical length g+h* is explored & then backtrack.
- Initial cut-off value decided is based on the estimate of the root node, ie, h value.
- It is complete & optimal
- TC: exponential ; SC: linear ($O(b^d)$, $O(d)$)
- ~~SC: polynomial~~ $O(bd)$

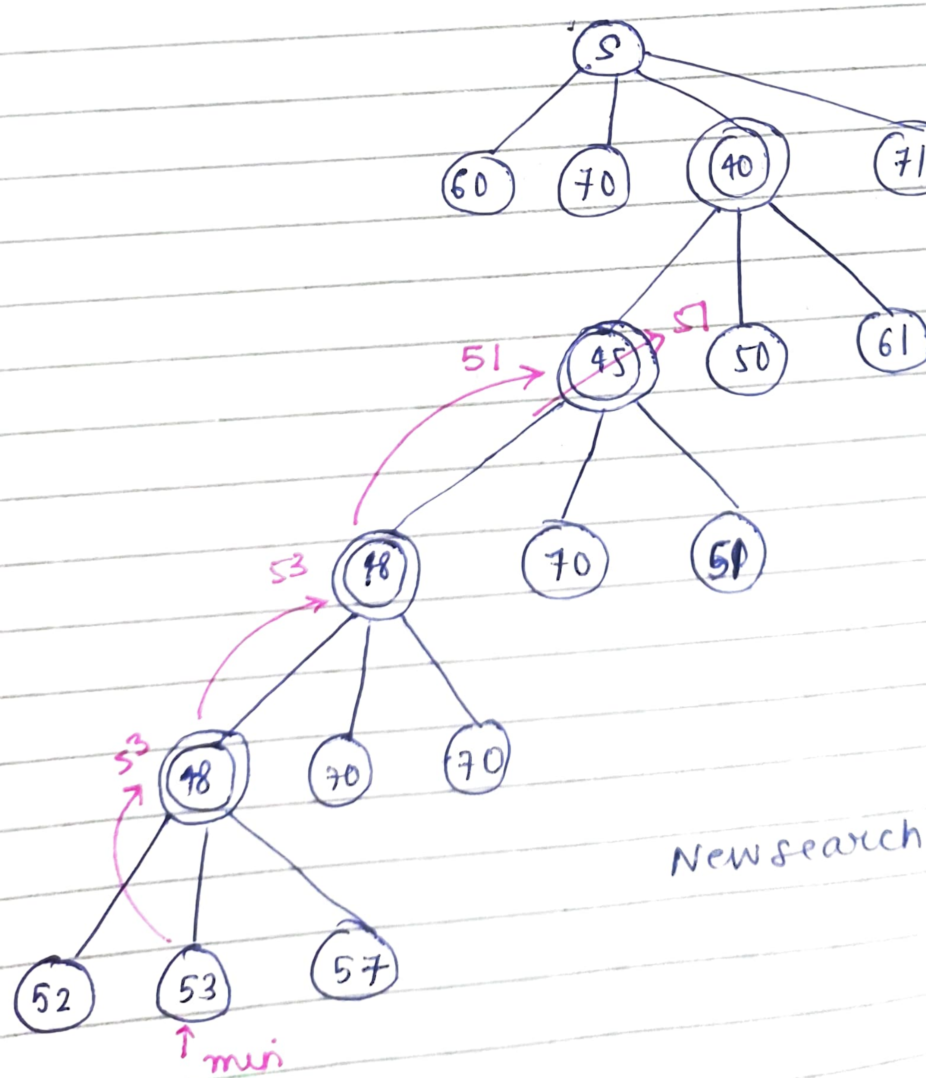
Algo:

1. Set the root node as current node & find g+h (ie f-score)
2. Set the cost limit as threshold for a node, ie, max f-score allowed
3. Expand the current node to its children & find their f scores
4. For any node w/ f-score > threshold, prune it & add it to CLOSED list
5. If goal node is found, return the path from the start node to the goal node
6. If not, repeat step 2 by changing threshold to the min. pruned value from CLOSED list. Repeat till goal node is reached.

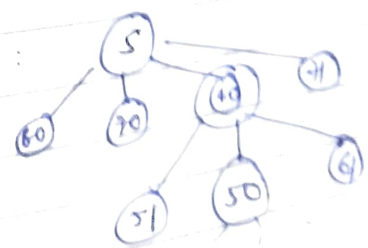
PAGE NO.
DATE: / 202

5. RECURSIVE BEST FIRST SEARCH (RBFS)

- mimics BFS w/ linear space.
- Its structure is similar to recursive DFS, but rather than continuing indefinitely down the current path, it uses f-limit variable to keep track of the f-value of the best alternative path available from any ancestor of the current node.
- If the current node exceeds this limit, the recursion unwinds back to the alternative path, and RBFS replaces f-value of each node along the path w/ a backed up f-value — the best f-value of its children.
- In this way, RBFS remembers the f-value of the best leaf in the forgotten subtree & can ∴ determine whether its worth re-expanding the subtree at some later time.



New search:



Importance of CLOSED list:

- Keeps a check on visited nodes & prevents the search from expanding them again
- It is the means for reconstructing the path after the soln is found.

6. PRUNING THE CLOSED LIST

Objectives: (1) stop 'leaking back'
(2) Reconstruct paths once goal node has been picked up by the algo.

1. To avoid ∞ loops, which may happen bec a node in CLOSED is the neighbour of the node being expanded.

(i) DIVIDE & CONQUER FRONTIER SEARCH (DCFS)

- A tabu list of all disallowed successors is maintained for each node (in OPEN)
- Every time a node X is generated as a successor of some node Y, Y is excluded from being the successor of X.
- Every time a node is expanded, it is put on a tabu list of all its successors
- When a node is expanded, only the non-tabu successors are generated.
- ∴ Every arc in the search graph is traversed only once & only in one direction

Lists maintained:

→ OPEN list

→ list of disallowed moves.

CLOSED list is <u>not</u> maintained.

How to fulfill pruning objectives?

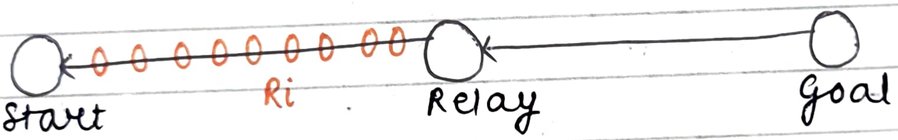
1. List of disallowed moves prevents search from leaking back

2. When frontier search picks ~~up~~ the goal node, it has a pointer to its relay node. We solve 2 problems recursively:

   (i) Start to Relay

   (ii) Relay to Goal.

[Relay: when a node on OPEN has $g(n) \approx h(n)$, mark it as relay & keep pointers from its descendants on OPEN]



If $T(d)$ is the TC to find the goal at 'd' steps then.

$$TC(DCFS) = T(d) + 2 \cdot T(d/2) + 4 \cdot T(d/4) + \ldots$$

If T is exponential, $T(DCFS) = T(d) \times d$

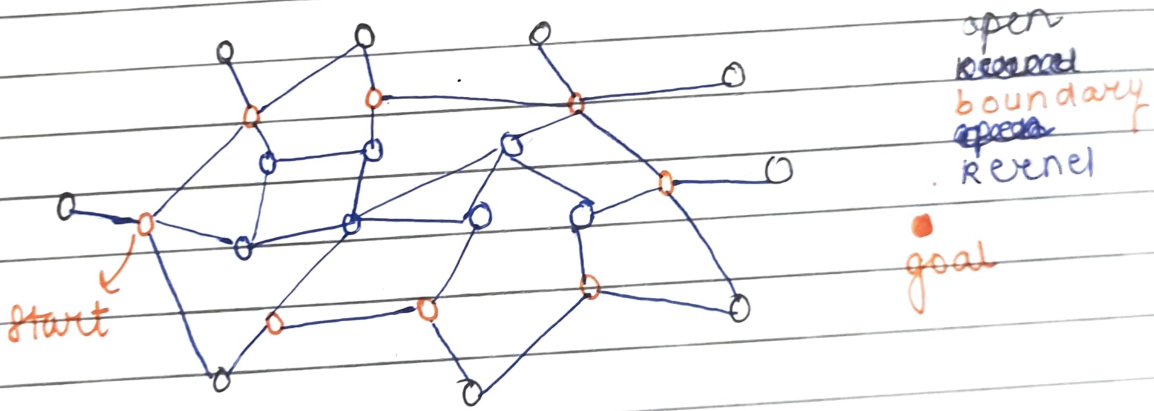
Thus, the algo saves on ~~time~~ space at the expense of running time.

(ii) ~~SMART~~ SPARSE MEMORY GRAPH SEARCH (SMGS)

- ~~It~~ keeps track of available memory
- Creates a relay layer when it senses memory is running out
- It may create many relay layers, or none if the problem is small enough to be solved by A*

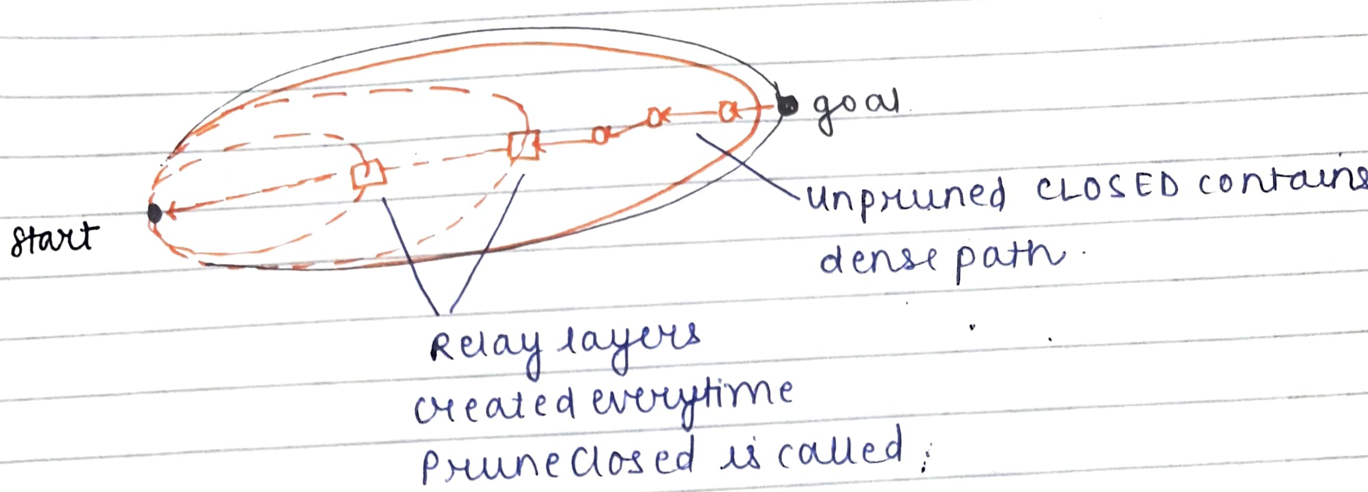
- The algo identifies 3 kinds of nodes :
  - → Boundary : have been inspected but have ~~some nodes on~~ atleast 1 child on OPEN
  - → Kernel : have been inspected & all its neighbours are in CLOSED
  - → Frontier : nodes in OPEN, generated but not inspected.

Kernel + boundary = CLOSED



- The algo begins by keeping all 3 kinds of nodes ~~but~~ & proceeds to pick nodes from OPEN & inspect them.
- Once it senses it is running out of memory, it :
  - (i) converts all boundary nodes into relay nodes
  - (ii) For each boundary node, it traces the back pointer to the relay node, & sets an ancestor pointer to that relay node
  - (iii) deletes all kernel nodes that are not relay nodes.

· Having finished pruning, it continues to pick up nodes from OPEN & inspect them.



Like DCFS, SMGS recursively calls itself with each segment in the $\underbrace{\text{sparse soln path}}_{\text{Relay layers.}}$ to find the Dense soln path.

7. PRUNING THE OPEN LIST :