2/3/20

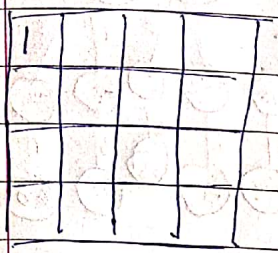
Page No.:
Date:
YOUVA

Back Tracking

The general method

- 8 Queens Problem $\{x_i : s_i\}$
- 4 Square, Queens $\{1, 2, 3, 4, 5, 6, 7, 8\}$ 8^8 problem

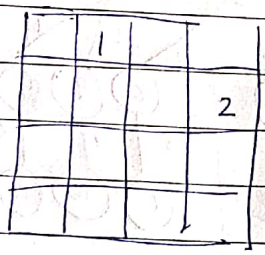
a)



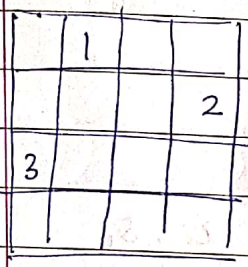
b)

c)

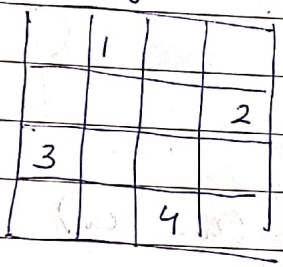
d)

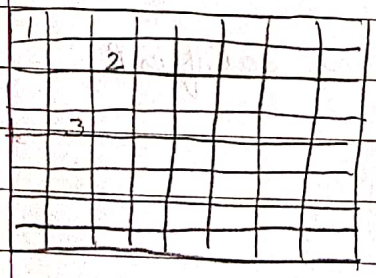


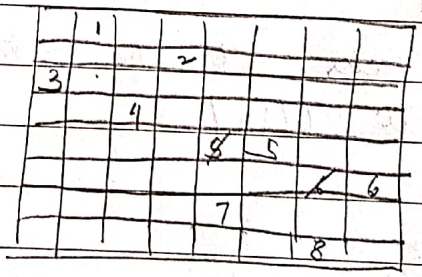
e)

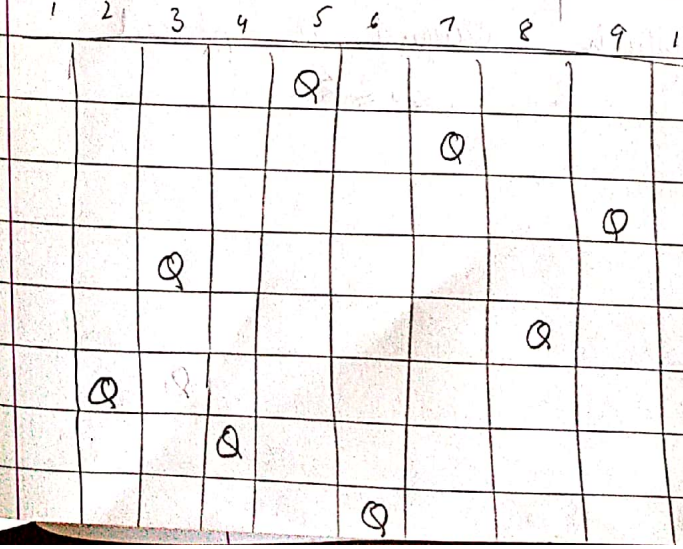


f)

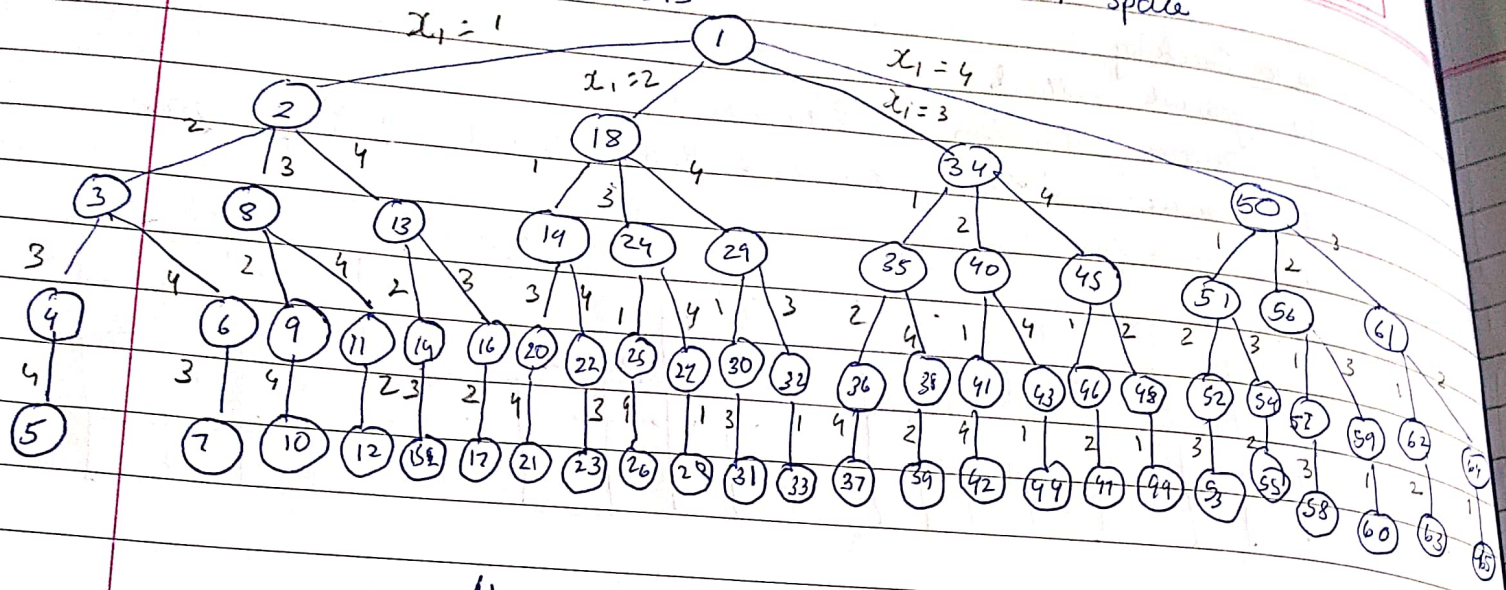








Tree Organisation of the 4 Queen Solution space
Nodes are numbered as in DFS



No of leaves = $4! = 4 \times 3 \times 2 \times 1 = \underline{24}$

The General Method

Defined solution for n tuple $(x_i \ldots x_n)$ where $x_i : S_i$

Finding vector that maximises / minimises a criterion function $P(x_i \ldots x_n)$

Sometimes it seeks all the vectors that satisfies P.

eg. Sort a $[1:n]$.

~~eg. Sort~~

8 Queen:

$(x_1 \ldots x_8)$ i.e. 8 tuples

x_i is column on which Queen i is placed

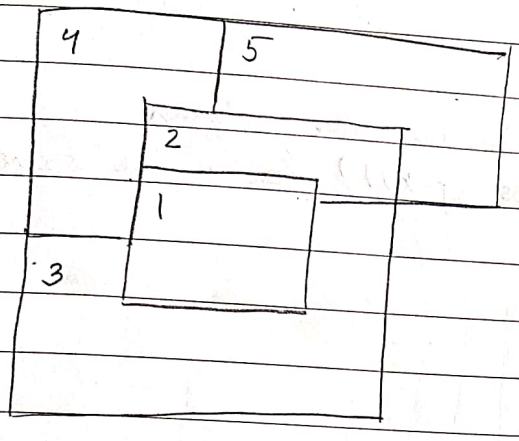
$S_i = \{1, 2, 3 \ldots, 8\}$ & $1 \le i \le 8$.

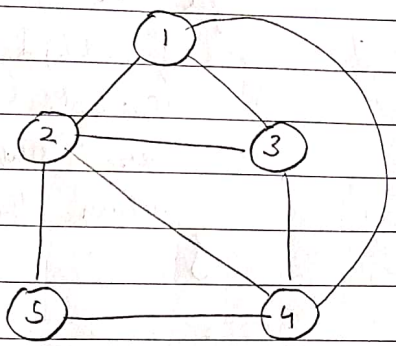
Solution space

Page No.:
Date:

Graph Coloring

A map & it's ~~planar~~ planar graph representation.





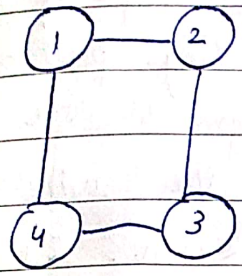

```
Algorithm N Queens (k,n)
// Using Backtracking this procedure prints all
// possible placements on an nxn
// chessboard so that they are non-attacking
 {
  for i=1 to n do
   {
    if Place (k,i) then
     {
       x[k] = i;
       if (k=n) then
          write (x[1:n]);
       else NQueens(k+1,n);
     }
   }
}
```

Page No.:
Date:

// x[] is a global array whose first k-1 values have been set.
// Abs(r) returns the absolute value of r.

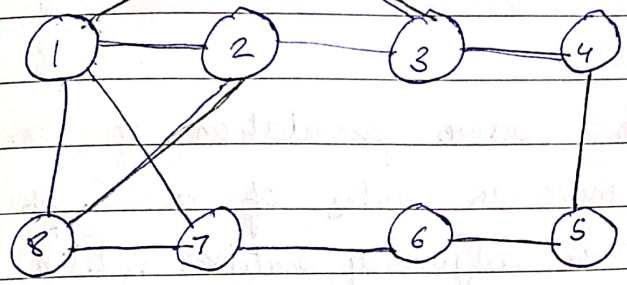
```
{
  for j=1 to k-1 do
    if ((x[j] = i)  // two in the same column
    or (Abs (x[j] - i) = Abs (j-k))) // or in the same diagonal
    then return false;
  return true;
}
```

4/3/20

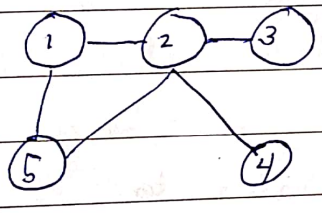


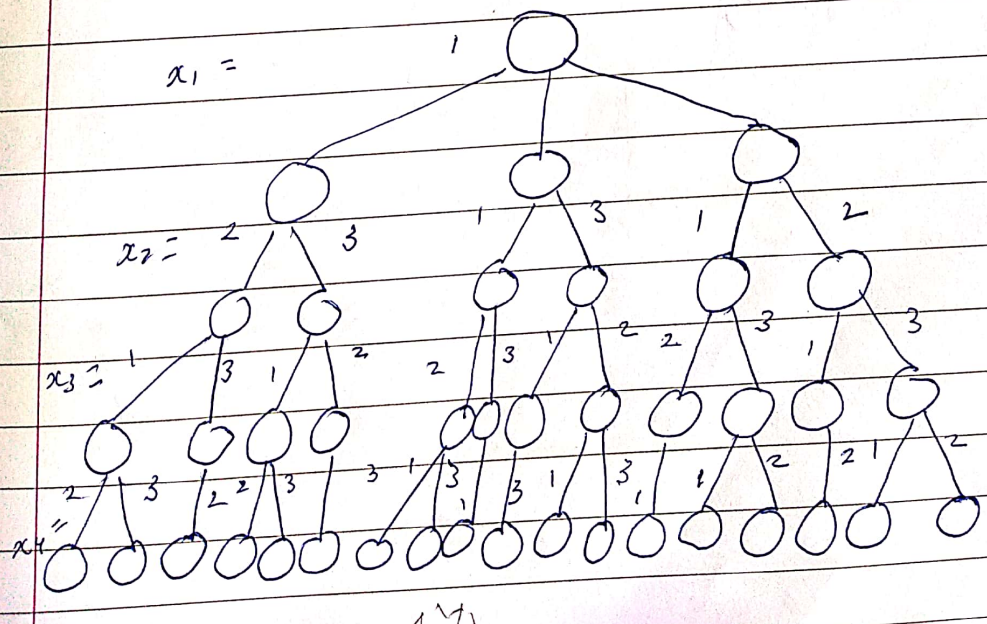
Hamiltonian cycle

G1:



G2:





17

- Let $G = (V, E)$ be a connected graph of n vertices.
- A Hamiltonian Cycle is a round trip path along n edges of G. that visits every vertex once & returns to it's starting position.
- In other words, if a Hamiltonian cycle begins at some vertex $V_i \in G$ & the vertices of G are visited in the order $V_1, V_2 \ldots V_{n+1}$ then the edges (V_i, V_{i+1}) are in E. & $1 \leq i \leq n$. & the V_i are distinct except for V_1 & V_{n+1} which are equal.

Algorithm Hamiltonian(k)
// This algorithm uses the recursive formulation of backtracking to find all the Hamiltonian cycles of a Graph
// The graph is stored as an adjacency matrix. G[1:n, 1:n].
// All cycles begin at node 1.
{

repeat
{ // Generate values for x[k].
NextValue(k); //Assign a legal next value to x[k].
if (x[k] = 0) Then return;
if (k = n) then write (x[1:n]);
else Hamiltonian (k+1);
} until (false);
}

4/3/20

Page No.:
Date:

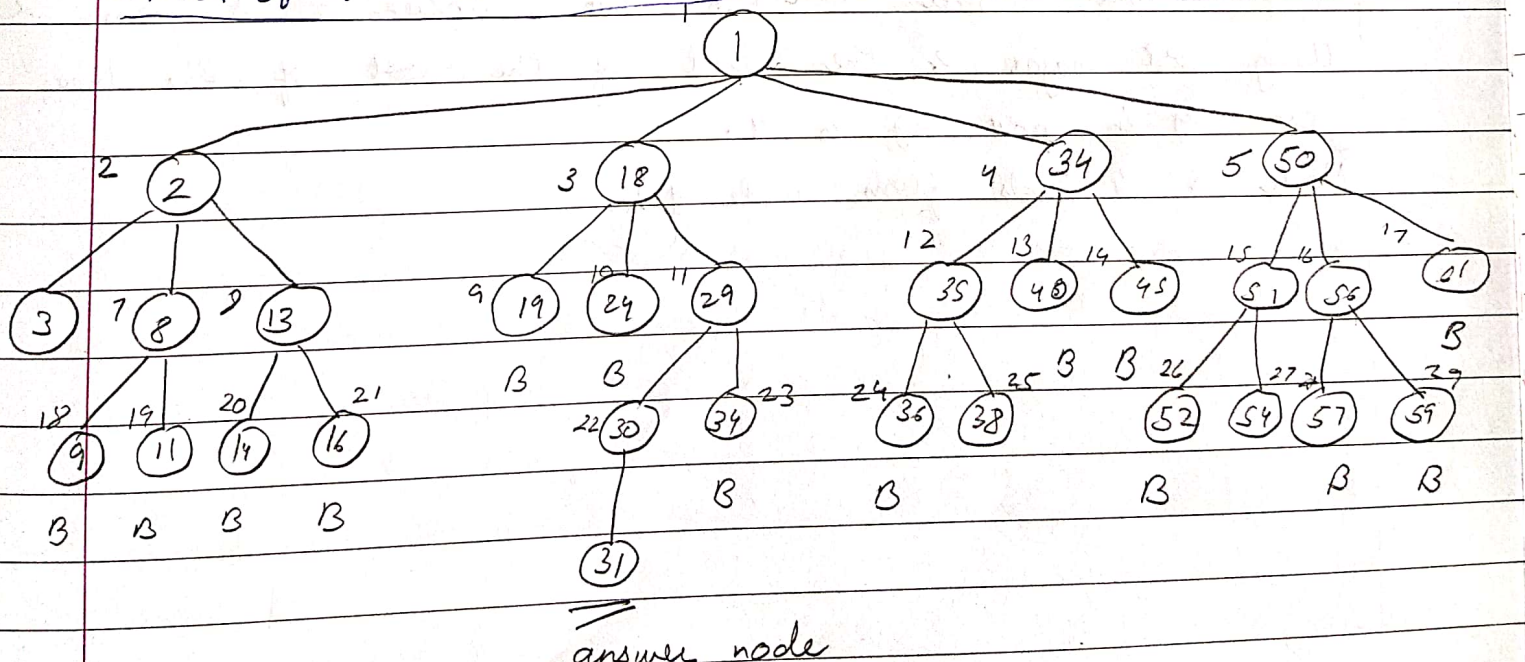
BRANCH & BOUND METHOD

Branch & bound method ref. to all state space search method in which all children of e-node are generated before any other live node can become the e-node Two graph strategies BFS, DFS in which the explorat" of new node cannot begin until the node currently being explored is explored. Both of these generalise to Branch & Bound strategies. In B&B terminology, a BFS like state space search will be called FIFO (First In First Out) as the the list of live nodes is a FIFO list or queue.

A DFS like state space search will be called a LIFO / FILO / stack.

As in case of backtracking, a bounding funct's is used to help avoid a generat" of subtrees that do not contain an ans. node.

Portion of 4 Queens State Space Tree Generated by FIFO Branch & Bound



LC Search (~~Least Cost~~) (Least Cost)

- Intelligent Ranking fun. c() for live nodes

29 - 30 (live)

(31)

- cost (additional computational effort)

live node → answer node

LC

FIFO

The ideal way to assign ranks would be cost needed to reach an ans. node from live node. For any node x, this cost would be 1 - The no. of nodes in the subtree x that need to be generated before an ans node is generated or more simply (2). The no. of levels (the nearest answer node) in the subtree x is from x Using cost major 2 the cost of the root of the tree For eg in prev fig is 4. Node is 4 levels from node 1).

LC Search (~~Least Cost~~) (Least Cost)

- Intelligent Ranking fun. c() for live nodes

29 - 30 (live)

(31)

- cost (additional computational effort)

live node → answer node

$\frac{x}{FIFO}$

The ideal way to assign ranks would be cost needed to reach an ans. node from live node.

For any node x, this cost would be 1 - The no. of nodes in the subtree x that need to be generated before an ans node is generated or more simply (2). The no. of levels (the nearest answer node) in the subtree x is from x.

Using cost major 2 the cost of the root of the tree for eg in prev fig is 4.

(Node is 4 levels from node 1).

1/3/20

The 15 Puzzle

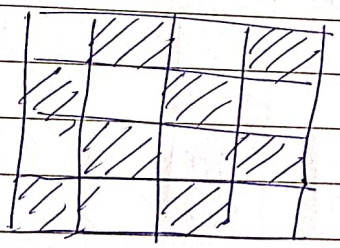
- Consists of 15 numbered tiles on a square frame w/ a capacity of 16 tiles.
- Given an initial arrangement of tiles, transform into goal arrangement of tiles, (through a series of legal move)
  - The only legal moves are ones in which a tile adj. to ES (Empty spot) is to moved to ES. Thus from the initial arrangement, 4 moves are possible.

1	3	4	15
2		5	12
7	6	11	14
8	9	10	13

Steps →

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

(a) Initial arrangement (b) Goal arrangement.



(c)

We can move any one of the numbered 2, 3, 5, 6 to the ES, following this move, other moves can be made. Each move creates a new arrangement of the tiles.

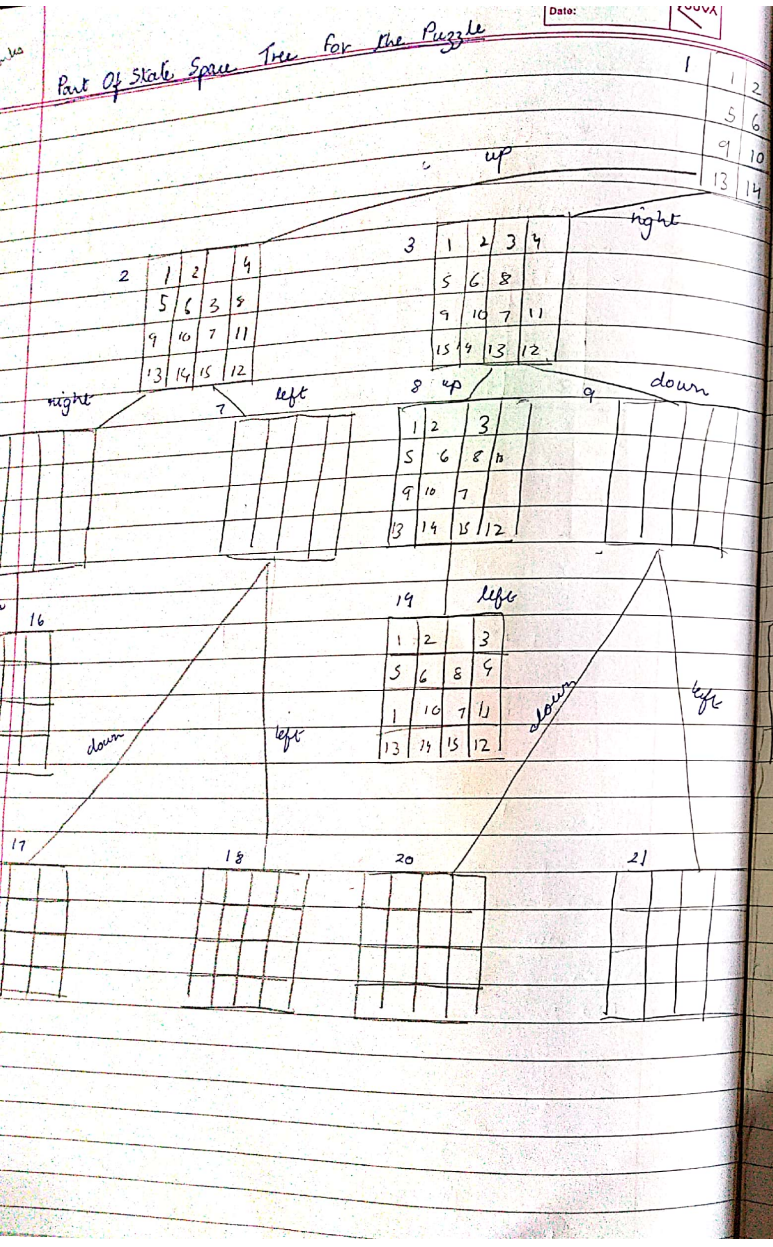
- These arrangements are called the states of the puzzle
- The initial & goal arrangements are called initial & goal state.
- A state is reachable from initial iff there is a seq of legal moves from initial state to this state.

Page No.:
Date:
YOUVA

- The state space of an initial state consists of all states that can be reached from initial state.
- The most straight-forward way to solve the puzzle would be to search space for goal state & use path from initial to goal state for the answer.
- Of these 16! only one half are reachable from any given initial state. Indeed the state space from the problem is very large.

Part of State Space Tree for the Puzzle

Part Of State Space Tree for the Puzzle



DRDO

Edges are labelled according to the direction in which empty space moves.

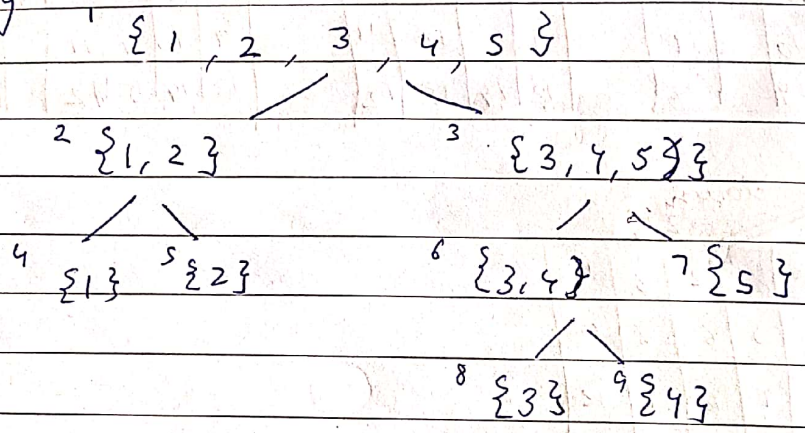
4/3/20

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

up → 2 right → 3 → down (4) → down (5)
→ down (6) → left (7) → 8 (up) → up (1)
→ up (10) → right (11) → down (12)

1	2	8	11
5	6	4	
9	10	3	12
13	14	7	15

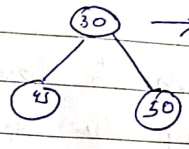
Bounding



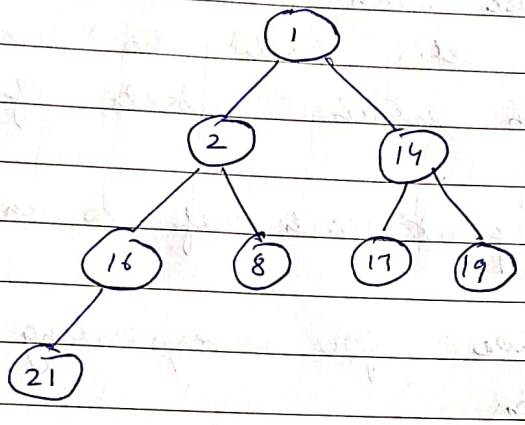
Control Abstraction for Least Search.

t is state space tree, c() is the cost funct" & x is node in t, c(x) is min. cost of any ans. node in sub tree w/ root x that implies c(t) is the cost of min cost ans node in t. However it is not possible to find an easily computed funct" c() as def" above, Instead a heuristic $\hat{c}$ that annals (c) is used. The heuristic should be easy to compute & generally have the property that x is either an ans node or leaf node than c(x) = $\hat{c}$(x). LCS algo uses $\hat{c}$ to find an ans node. This algo uses least(), Add(x) funct" to delete & add a live node from all to the least of live node resp. least() finds the live node w/ least of $\hat{c}$(x). This node

is deleted from the list of live nodes & returned.
Add(x) adds to the new live node x to the list of live nodes. The list of live nodes is usually implemented as a min heap



2, 8, 14, 16, 1, 17, 19, 21



least node = record { listnode *next, *parent, float cost = 0.0 }
Algorithm { LSearch }
// Search t for an ans node
{ if *t is an ans node then output *t and return E = t ; // E node
Initialize the list of live nodes to be empty
repeat
{ for each child x of t do
{ if x is an ans node then o/p the parent for x to t and return;
Add(x) // x is a live node.
(x → parent) = E // Part for path to root.
{ if there are no more live nodes then
{ write ("No ans node"); return;

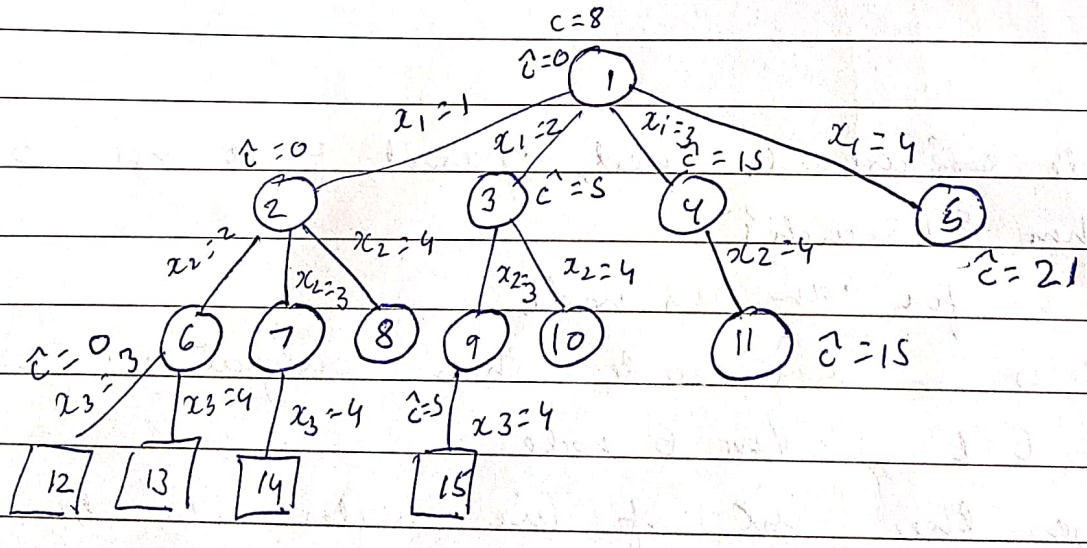
```
} E = least();
} until (false);
```

Branch & Bound

During the execut^n of this, the
⇒ min f(x) where $x \in X$ while f(x) is cost ~~for~~ func^n & x is combing the set X. X may be a set of all real nos, integers, vectors of real, vector of int, Mostly the case with mixed int programming. X is a feasible set. Finds the bound of cost funct^n f given certain subsets of X.

Evaluate funct^n f at each leaf to an ~~ex~~ last

State space tree for corresponding to variable size formation



11/3/20

Page No.:
Date:
YOUVA

Dijkastra's

Dijkastra's Algo is a greedy algorithm & it has time complexity of $O(V \log V)$. However Dijkastra's algo does not take into consideration, the -ve weighted edges.
Given graph & a src vertex in the graph, find shortest paths from src to all vertices in the given graph.
Given src is the root.
Two sets (1 contains vertices included in the shortest path tree & other set includes vertices not yet included in the shortest path tree).
At every step of the algo, we find a vertex which is in the other set (set not yet included) & has min dist from the src.

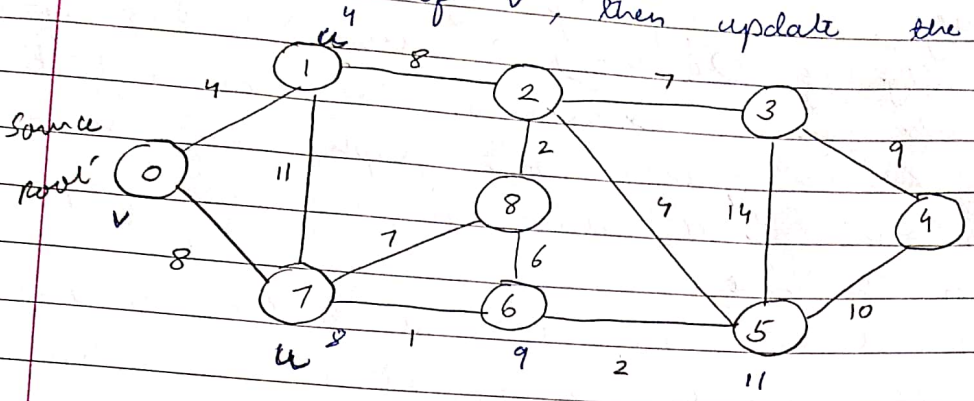
Algorithm:

1) Create a set SPTSET (Shortest Path Tree Set) that keeps track of vertices included in the shortest path tree ie. whose main dist from src is calculated & finalised. Initialising this set as empty.

2) Assign a dist. value to the vertices in i/p graph. Initialize all dist values as infinite. Assign dist value as 0 for src vertex so that it is picked 1st.

3) While SPTSET doesn't include all vertices.
- Pick a vertex 'u' which is not in the SPTSET & has min dist value.
- Include u to SPTSET
- Update dist value of all adj ~~dist~~ vertices of u.
To update the dist. values iterate through all adj vertices.

for every adj vertex v, if sum of dist value of u (from src) & weight of edge u-v is less than the dist value of v, then update the dist value of v.

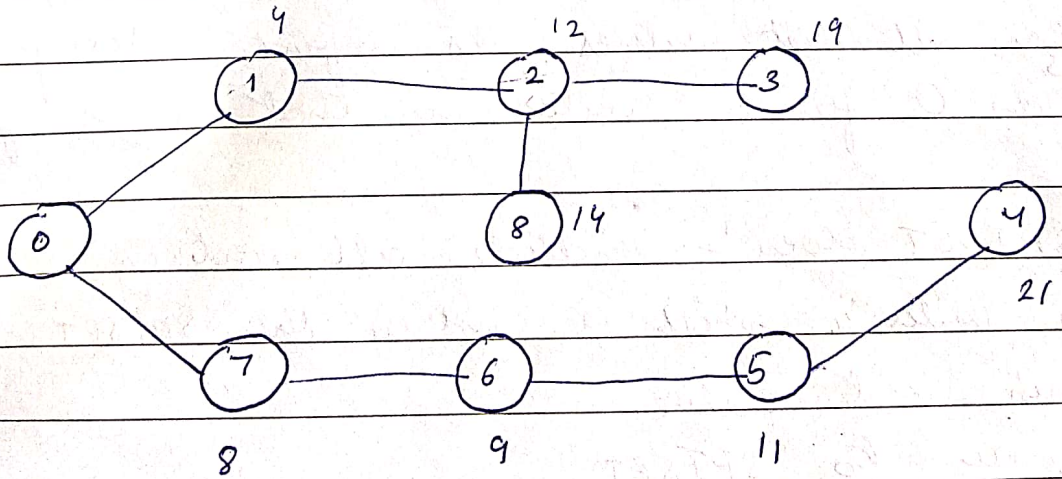


Dijkastra's Algo

SPTSET = Empty

distances assigned to vertices are
{0, INF, INF, INF, INF, INF, INF, INF}

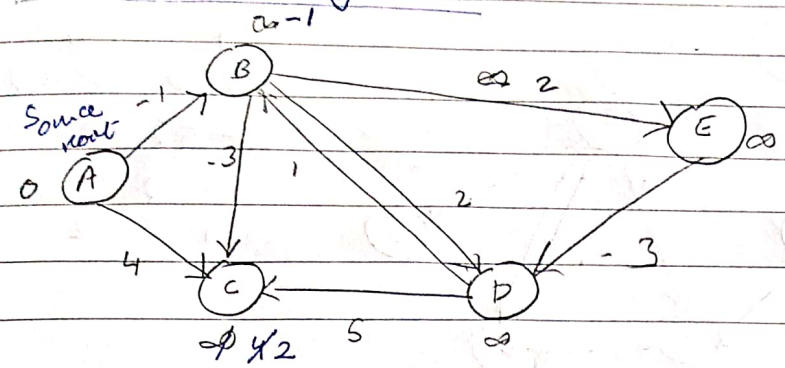
SPT set {0}
SPT set {0, 1}
SPT set {0, 1, 7}
SPT set {0, 1, 7, 6}
SPT set {0, 1, 7, 6, 5} Heap



Time Complexity: O(v log v) with use of Fibonacci Heap.

Page No.:
Date:

Bellman Ford Algorithm



A	B	C	D	E
0	∞	∞	∞	∞

Let all edges be processed in the following order

~~A~~ (B, E) (D, B) (B, D) (A, B)
(A, C)
(D, C) (B, C) (E, D)^{x}

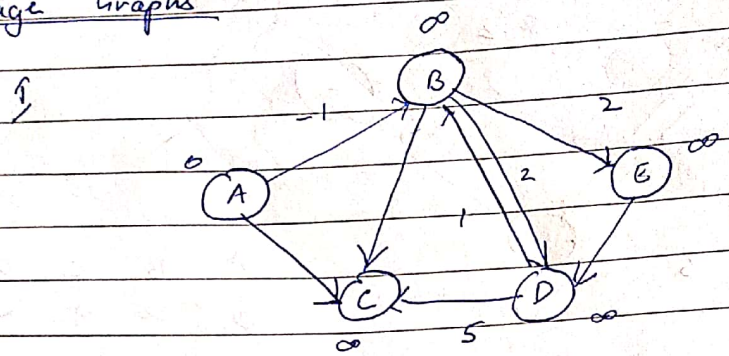
We get the following distances when all edges are processed the first time.

A	B	C	D	E	
0	∞	∞	∞	∞	— Initial
0	-1	∞	∞	∞	
0	-1	4	∞	∞	
0	-1	2	∞	∞	

Date: YOUVA

12/3/20

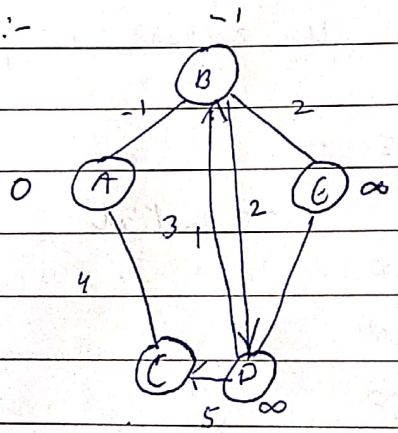
Bellman Ford Algo
Multistage Graphs



(B, E) (D, B) (B, D) (A, B)
~~(A, B)~~ (A, C)
(D, C) (B, C) (E, D)
9 2 -3

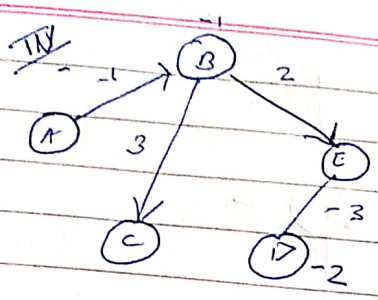
A	B	C	D	E
0	9	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞

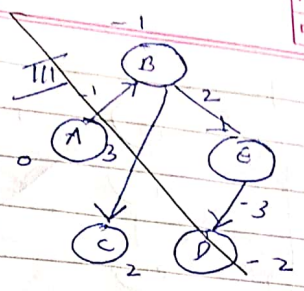
P1 :-

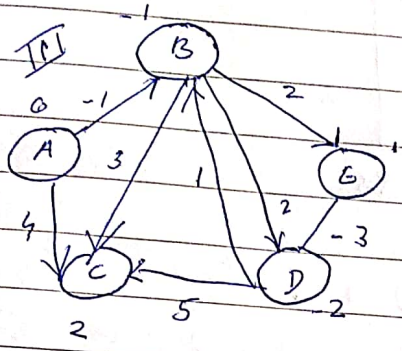


P2 :-

A	B	C	D	E
0	-1	2	∞	1
0	-1	2	1	1
0	-1	-2	2	1

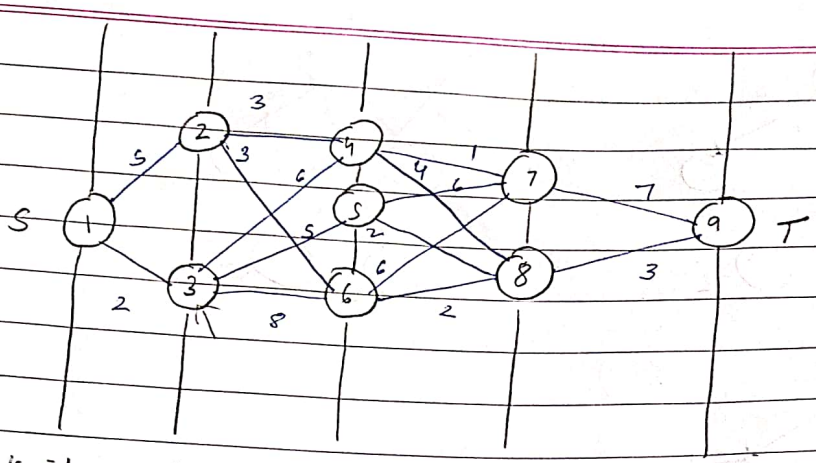






Given a graph & a source vertex src in the graph. Find shortest path from src to all vertices
The graph may contain -ve weight edges.
Simpler than Dijkastra's & suits well for distributed systems.
Time complexity O(v, e) which is more than Dijkastra's

eg. Given src vertex as 0, initialize all distances as infinite. except the dist to src itself.
Total no of edges = 5 so all edges must be processed.
Algo processes all edges 2 more times (in general) dists are minimised after the 2nd iteration. ∴ 3, 4th iterations don't update the distances.



k = 1 k = 2 k = 3 k = 4 k = 5 5 - 2 = 3

Cost (i, j) s-3

Cost (u-2, v)

no. of nodes at 3

Cost (3,4) = min { c(4,7) + cost (7,9), c(4,8) + cost (8,9) } = 7

node no on 3

cost (3,5) = min { c(5,7) + cost (7,9), c(5,8) + cost (8,9) } = 5

Cost (3,6) = min { c(6,7) + cost (7,9), c(6,8) + cost (8,9) } = 5

cost (2,2) = min { c(2,4) + cost (4,8) + cost(8,9), c(2,6) + cost (6,8) + cost (8,9) } = 8

cost (2,3) = min { c(3,4) + cost (4,8) + cost (8,9), c(3,5) + cost (5,8) + cost (8,9), c(3,6) + cost (6,8) + cost (8,9) } = 10

Cost (1,1) = { c(1,2) + Cost (2,6) + Cost (6,8) + cost (8,9), c(1,3) + Cost (3,5) + cos (5,8) + cos (8,9) } = 12

c(1,3) + cost (3,6) + Cost (6,8) + cost (8,9) } = 13

Hence the path having minimum cost is : 1 → 3 → 5 → 8 → 9 = 12