Sum of subsets problem

- It is a variant of Knapsack problem.
- A knapsack with capacity W is to be filled with items of certain weights.
- Sum of subsets problem is to check whether it is possible to find subsets of items whose sum equals W.

Subset sum Problem

- problem of finding a subset such that the sum of elements equal a given number.
- The backtracking approach generates all permutations in the worst case
- but in general, performs better than the recursive approach towards subset sum problem.

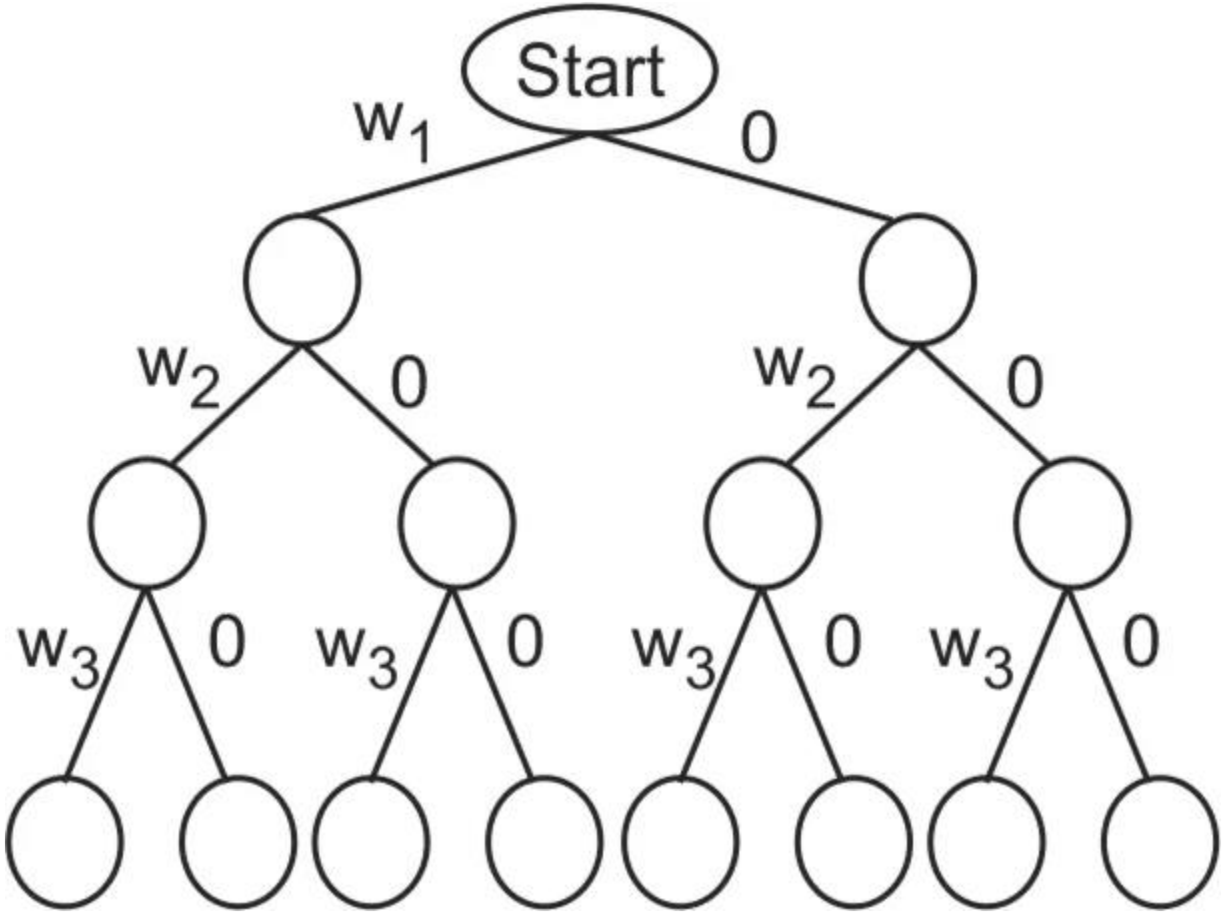
Steps

- A state space tree is drawn as a binary tree
- Every node at level i has two branches at level i+1
- one branch '1' indicates that item is included
- other branch '0' indicates that the item is NOT included
- Lower nodes keep on accumulating the sum of items.

- As we go down along depth of tree we add elements so far,
- and if the added sum is satisfying explicit constraints,
- we will continue to generate child nodes further.

- Whenever the constraints are not met, we stop further generation of sub-trees of that node,
- and backtrack to previous node to explore the nodes not yet explored.

- We need to explore the nodes along the breadth and depth of the tree.



Start
w_1
0
w_2
0
w_2
0
w_3
0
w_3
0
w_3
0
w_3
0

- Weights are usually kept in sorted order.
- Let W be the desired sum.
- Let w_{old} be sum of all weights till level i, and let next weight be w_{i+1}.
-
- The node is non promising if $w_{old} + w_{i+1} > W$, as it crosses the capacity W.
- Let w_{rem} tot be the sum of remaining weights.
- If $w_{old} + w_{rem} < W$, the node is also non-promising.

- Consider an example where the items are 3, 5, 6, 7
- and M = 15

- w1:3 w2:5 w3:6 w4:7 M=15

- M 0
- w1 3 0
- M 3
- w2 5 0
- M 8 3
- w3 6 0
- M 14 8
- w4 7 0 7 0
- M 21 14 15 8

[1 1 0 1]

- w1:3 w2:5 w3:6 w4:7 M=10

- M 0
- w1 3 0
- M 3
- ~~w2~~ 5 0
- M 8 3
- w3 6 0 6 0
- M 14 8 9 3
- w4 7 0 7 0 7 0 7 0
- M 21 14 15 8 16 9 10 3

- w1:3 w2:5 w3:6 w4:7 M=18

- M 0
- w1 3 0
- M 3 0
- w2 5 0 5
- M 8 3 5
- w3 6 0 6 0 6 0
- M 14 8 9 3 11 5
- w4 7 0 7 0 7 0 7 0 7 0 7 0
- M 21 14 15 8 16 9 10 3 18 11 12 5

w1 not included

[0 1 1 1]

- w1:3 w2:5 w3:6 w4:7 M=13

1 - M 0
- w1 3 0
2 - M 3 0
- w2 5 0 5 0
4 - M 8 3 5 0
- w3 6 0 6 0 6 0 6 0
8 - M 14 8 9 3 11 5 6 0
- w4 7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0
16 - M 21 14 15 8 16 9 10 3 18 11 12 5 13 6 7

- How many maximum solutions generated?

- For n=4, it is 16 possible solutions

- w1:3 w2:5 w3:6 w4:7

- M 0
- w1 3 0
- M 3 0
- w2 5 0 5 0
- M 8 3 5 0
- w3 6 0 6 0 6 0 6 0
- M 14 8 9 3 11 5 6 0
- w4 7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0
- M 21 14 15 8 16 9 10 3 18 11 12 5 13 6 7

- w1:3 w2:5 w3:6 w4:7

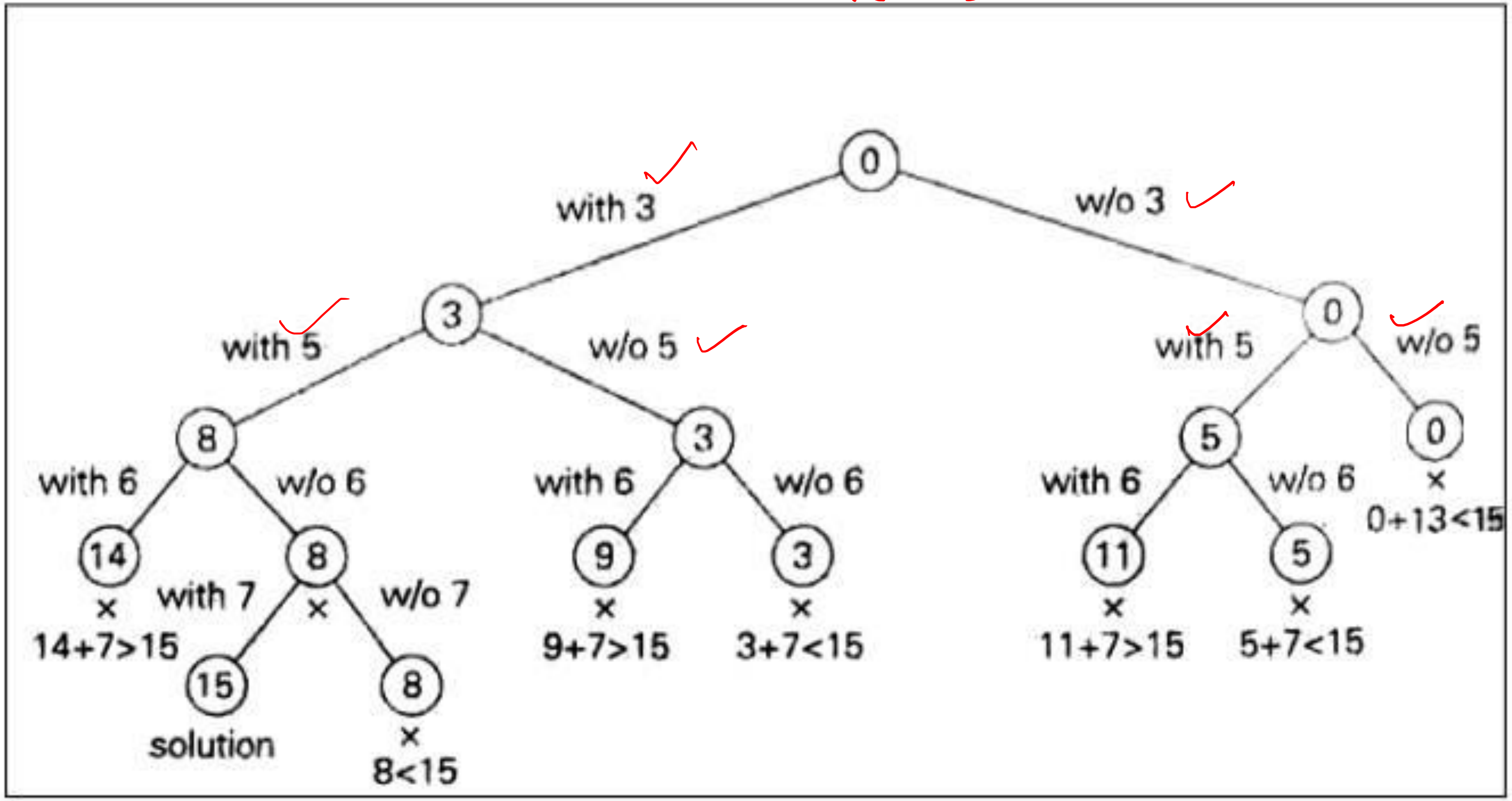
- M 0
- w1 3 0
- M 3 0
- w2 5 0 5 0
- M 8 3 5 0
- w3 6 0 6 0 6 0 6 0
- M 14 8 9 3 11 5 6 0
- w4 7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0
- M 21 14 15 8 16 9 10 3 18 11 12 5 13 6 7

- w1:3 w2:5 w3:6 w4:7

- M 0
- w1 3 0
- M 3 0
- w2 5 0 5 0
- M 8 3 5 0
- w3 6 0 6 0 6 0 6 0
- M 14 8 9 3 11 5 6 0
- w4 7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0
- M 21 14 15 8 16 9 10 3 18 11 12 5 13 6 7

- w1: 3 w1:0
- M:3 M:0
- w2: 5 w2:0, w2:5 w2: 0
- M:8 M:3 M:5 M:0
- w3:6 w3:0 w3:6 w3:0 w3:6 w3:0 w3:6 w3:0
- M:14 M:3 M:8 M:9 M:3 M:11 M:5 M:6
- w4:7 w4:0 w4:7 w4:0 w4:7 w4:0 w4:7 w4:0

W=15


0
with 3
w/o 3
3
0
with 5
w/o 5
with 5
w/o 5
8
3
5
0
with 6
w/o 6
with 6
w/o 6
with 6
w/o 6
×
0+13<15
14
8
9
3
11
5
with 7
w/o 7
×
14+7>15
×
×
9+7>15
×
3+7<15
×
11+7>15
×
5+7<15
15
8
solution
×
8<15

- Possible solutions
- [1 1 1 1] all weights Binary vector
- [1 1` 1 0] → First weight not included
- . . .. . .
- . . . . . .
- [0 0 0 0]
- how many? $2^4 = \underline{16}$

Basic idea of subset sum

- Go on adding the items of the subset.
- if for any item, sum > W, drop that item
- Do not proceed further. Backtrack to previous solution
- Take the next item and continue.

subset sum Algorithm

1. Start with an empty set.
2. Add to the subset, the next element from the list.

3. If the subset is having sum W then stop with that subset as solution.
4. If the subset is not feasible or if we have reached the end of the set then backtrack through the subset until we find the most suitable value.

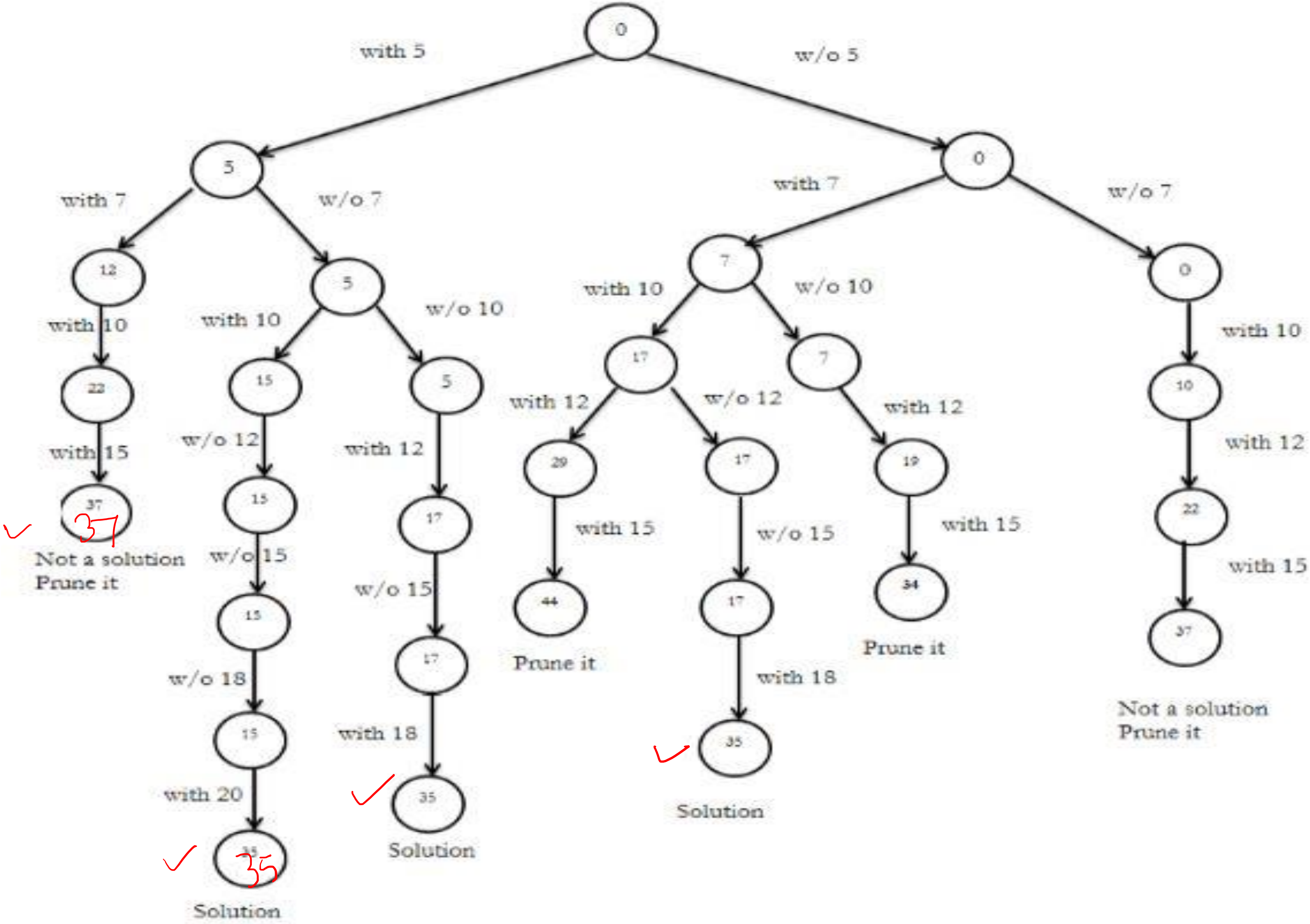
5. If the subset is feasible then repeat step 2.
6. If we have visited all the elements without finding a suitable subset and if no backtracking is possible then stop without solution.

Example 2

- Solve following problem and draw portion of state space tree
- {5, 7, 10, 12, 15, 18, 20}
- W = 35

- One solution is shown in next slide.
- Proceed for more solutions

Initially subset = {}	Sum = 0	Description
5	5	Then add next element.
5, 7	12,i.e. 12 < 35	Add next element.
5, 7, 10	22,i.e. 22 < 35	Add next element.
5, 7, 10, 12	34,i.e. 34 < 35	Add next element.
5, 7, 10, 12, 15	**49**	**Sum > 35. Hence backtrack.**
5, 7, 10, 12, 18	**52**	**Sum > 35. Hence backtrack.**
5, 7, 10, 12, 20	**54**	**Sum > 35. Hence backtrack.**
5, 12, 15	32	Add next element.
5, 12, 15, 18	50	Not feasible. Therefore backtrack
5, 12, 18	35	Solution obtained as M = 35



0
with 5
w/o 5
5
0
with 7
w/o 7
with 7
w/o 7
12
5
7
0
with 10
with 10
w/o 10
with 10
w/o 10
with 10
22
15
5
17
7
10
with 15
w/o 12
with 12
with 12
w/o 12
with 12
with 12
37
37
15
17
29
17
19
22
Not a solution
Prune it
w/o 15
w/o 15
with 15
w/o 15
with 15
with 15
15
17
44
17
34
37
w/o 18
with 18
with 18
Prune it
Prune it
15
35
35
Not a solution
Prune it
with 20
Solution
Solution
35
Solution

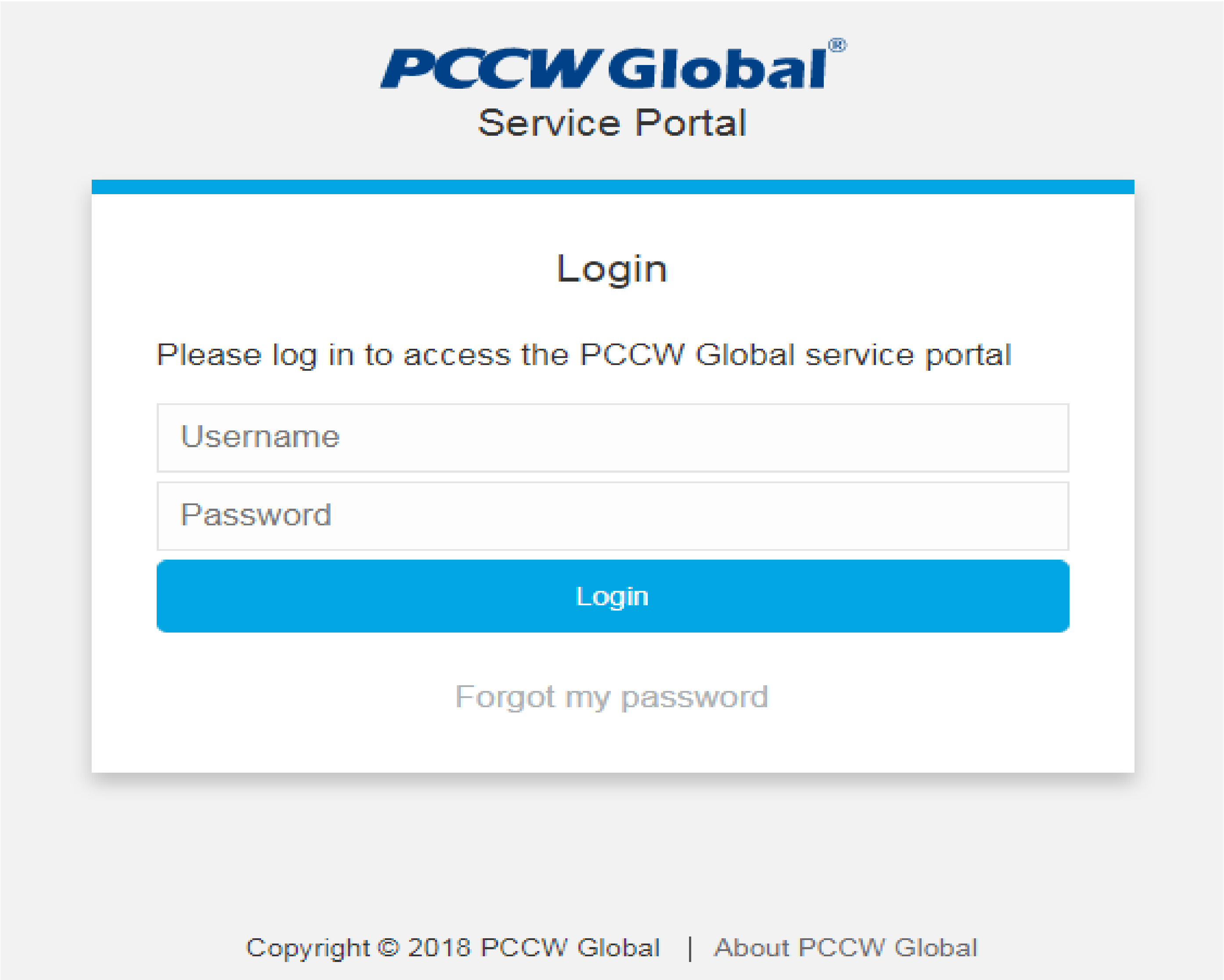
Number of possible subsets

- The various solutions can be described by
- [1 1 1 1 1 1 1] all 7 weights included
- [1 1 1 1 1 1 0] all weights included except first one
- [1 1 1 1 1 0 0] weights do not include first two
- [. . . . .]
- [. . . . .]
- [0 0 0 0 0 0 0]
- 128 possible subsets

Complexity of sum of subsets problem

- Generated state space tree is a binary tree.
- At every stage , a node generates 2 child nodes.
- Add Number of nodes generated at each level:
- $1 + 2 + 2^2 + 2^3 + \ldots + 2^n$
- $= 2^{n+1} - 1$
- $= \mathbf{O(2^n)}$ EXPONENTIAL PROBLEM

- One possible application of subset sum problem
- Checking passwords


PCCW Global®
Service Portal
Login
Please log in to access the PCCW Global service portal
Username
Password
Login
Forgot my password
Copyright © 2018 PCCW Global | About PCCW Global

- How does a computer verifies a user’s password?
- 1. The simplest system: machine keeps a copy of the password in an internal file.
- **Drawback:**
- anyone with access to internal file could misuse it.

- 2. Here is a possible alternative scheme:

- The computer generates 500 distinct values of a in an internal file.
- 13, 25, 45, 49, 58, 60, 77, 82, 102, 123, 131, 138, 144, ..., 921, 938
- When a user types in a password, a program converts symbols of passwords to numbers of this set.
- These numbers constitute a subset.
- The computer simply checks the sum of this subset and tries to match with the number stored for that user.

- Consider the following password:
- k7ts6#

- The program converts these symbols to numbers of the subset.
- k: 45, 7: 123, t: 60, s: 82, 6: 25, #:231
- TOTAL: 45+123+60+82+25+231 = 566
- The computer does not store the subset but keeps the total associated with the appropriate subset (566).
- When the user types the subset, the computer tests whether the sum of subset matches with the total stored for that user.
- Even if you know the total, can you generate a subset with sum as 566?
- If you have patience try generating 2^{500} different combinations. . . . . .

- https://codecrucks.com/sum-of-subsets-how-to-solve-using-backtracking/

Hamiltonian Cycle Problem

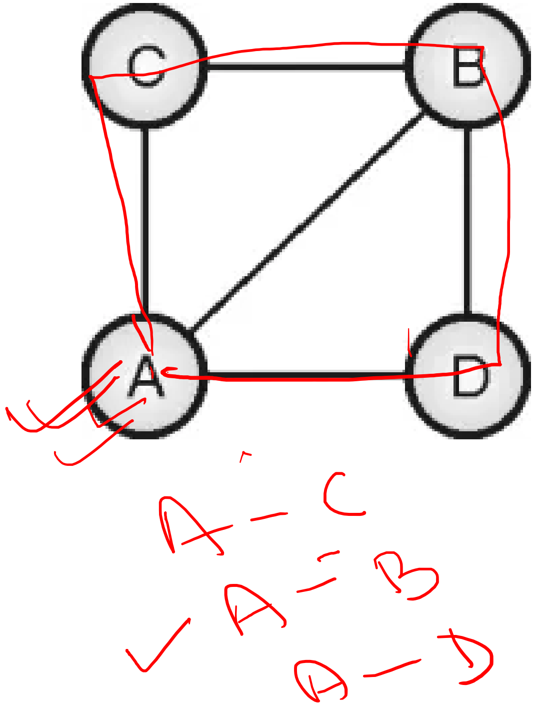
- The Hamiltonian Cycle Problem is concerned with finding paths through a given graph,
- such that those paths visit each node exactly once, ending at the start node.
- It may not include all the edges.

- The TSP can be considered as Ham. Cycle prob. which is concerned with computing the lowest cost Hamiltonian cycle on a weighted graph.

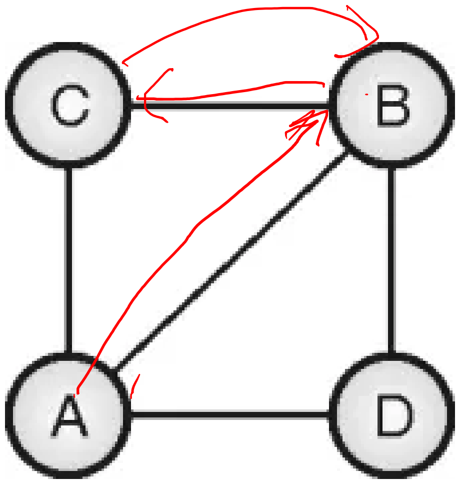
Constraints for Hamiltonian cycle problem

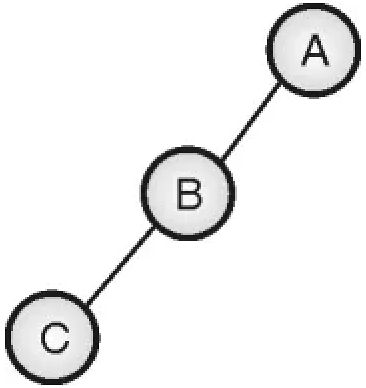
1. In any path, vertex i and vertex (i + 1) must be adjacent nodes.
2. First and (n – 1)th vertex must be adjacent (to go back to start)
3. Vertex i must not appear in the first (i – 1) vertices of any path (vertices appear only once)

- How to check if two nodes are adjacent?
- From the Adjacency matrix representation of the graph

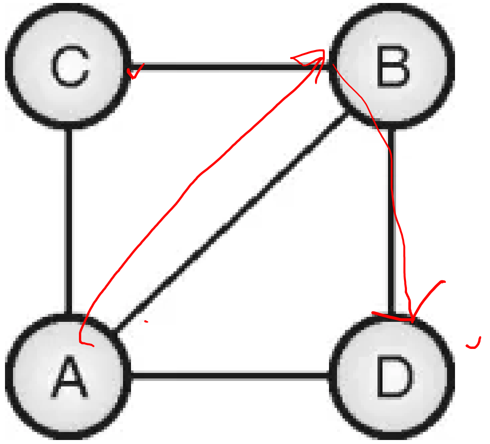


- Figure out if given graph has a Hamiltonian cycle
- Start with vertex A.
- visit Neighbor B
- Does path include all nodes?
- solution not complete
- explore neighbors of B {C,D}

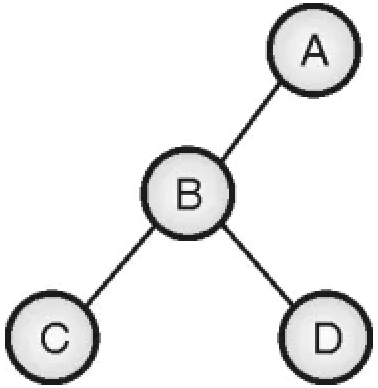




- explore neighbors of B {C,D}
- include neighbor C.
- Path A-B-C reaches Deadend, as C has no new neighbors
- *Backtrack* and check other neighbors of B.

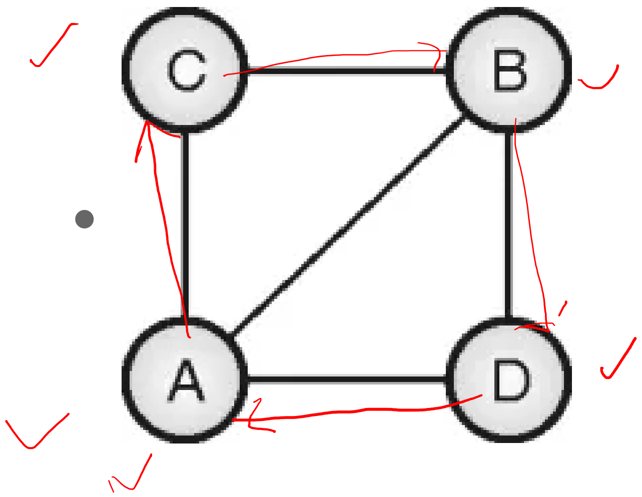


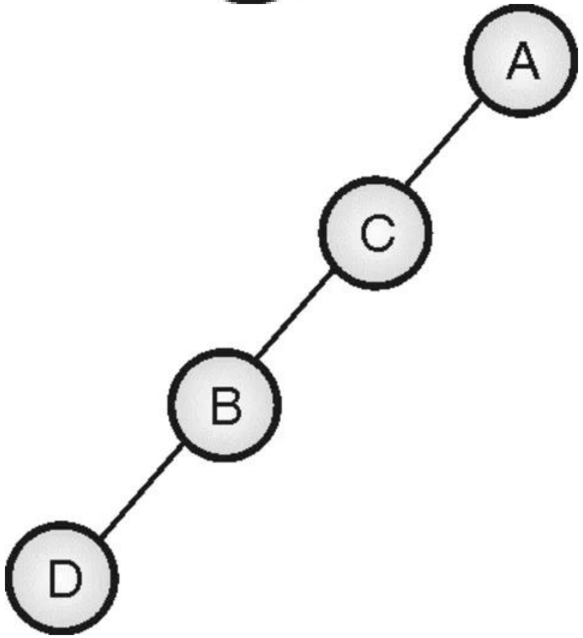
A B C
A B D



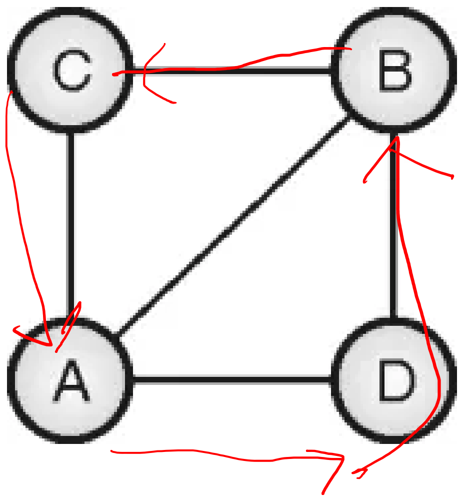
- Explore vertex D.
- The inclusion of D does not lead to a complete solution, as all adjacent vertices of D are already a member of the path formed so far.
- So A-B-D also leads to a dead end.
- Backtrack to B, no new neighbors
- Further Backtrack to A.

A - B - Dead end X
A - C
A - D





- New neighbor is C.
- This leads to path A-C-B-D
- From where we can reach the start vertex A.
- So the Hamiltonian cycle is
- A-C-B-D-A
- Backtrack to A, and check if any other neighbor is still to be explored.

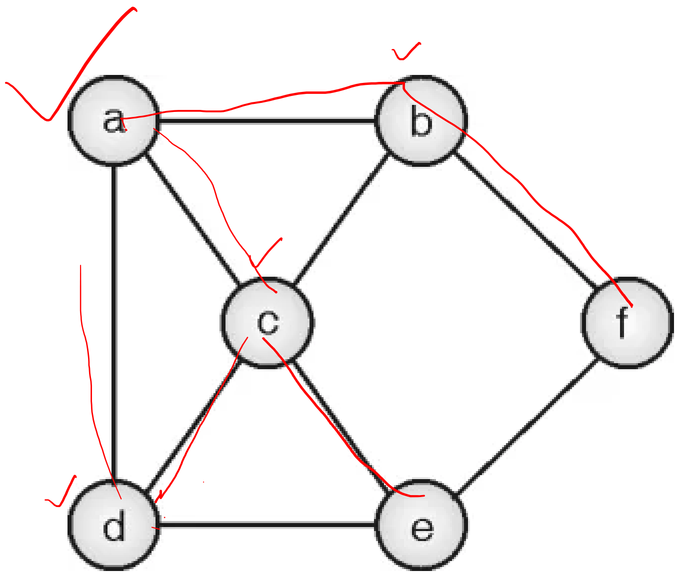


A D B C A

- Unexplored neighbor is D. Leads to path A-D.
- From D neighbor is B.
- This leads to path A-D-B.
- Following neighbor C, path is A-D-B-C.
- From where we can reach the start vertex A.
- Another Hamiltonian cycle is

  A-D-B-C-A

Discover Hamiltonian cycles

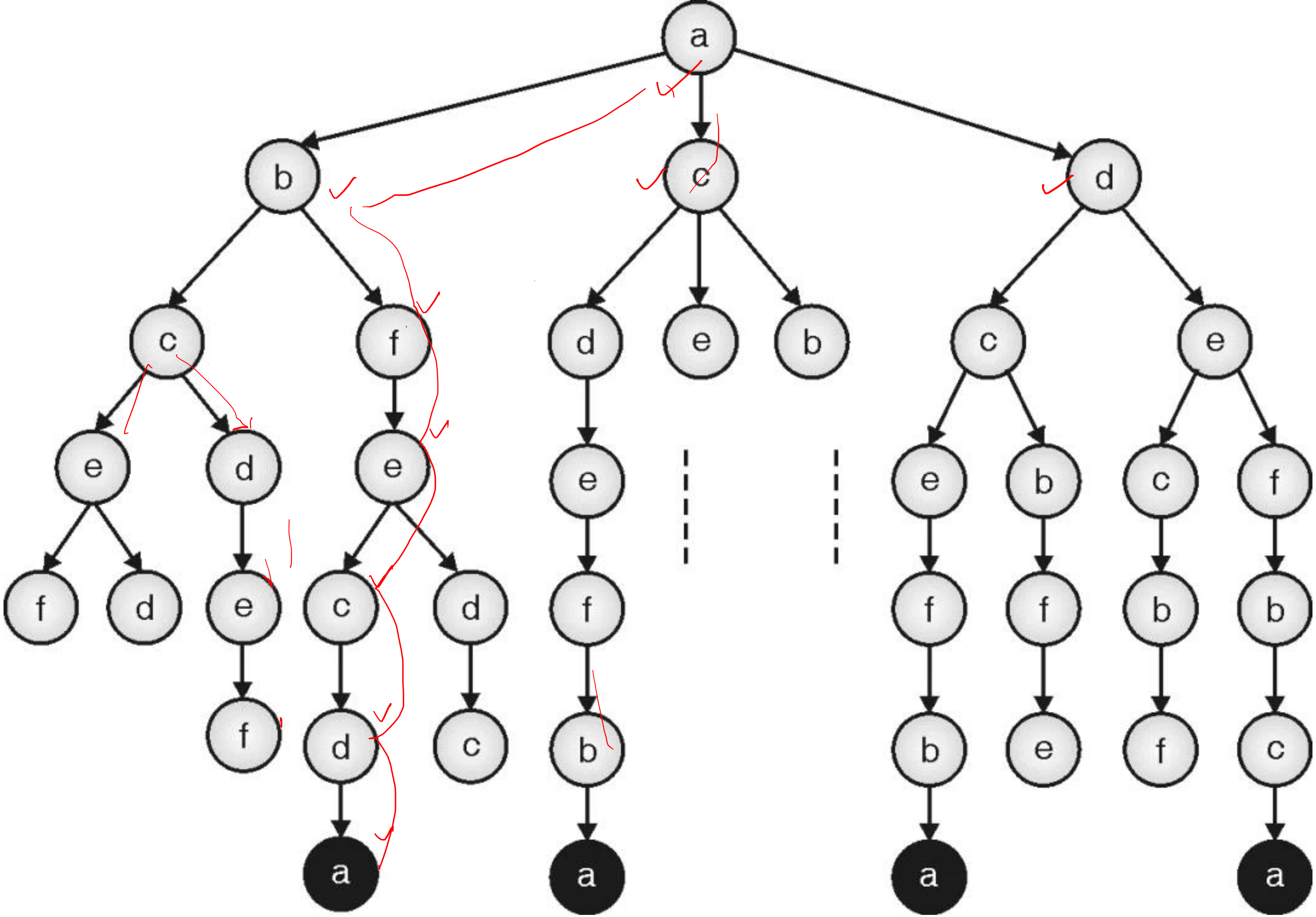


a – b – . . . . .

a – c – . . . . .

a – d . . . . . .

- Start vertex is a.
- Initial path could be
- a-b
- a-c, or
- a-d
- Create search tree using these leads one by one.
- Keep record of all discovered Hamiltonian Cycles.


a
b
c
d
c
f
d
e
b
c
e
e
d
e
e
b
c
f
f
d
e
c
d
f
f
f
b
f
b
f
d
c
b
b
e
f
c
a
a
a
a

Algorithm for Hamiltonian Cycle Problem

- We use two functions
- First function is *promising_Hamiltonian(i)*, which indicates if node can be added,
- This is used in second function named *Hamiltonian(i)*
- when all nodes are covered, it prints the solution

promising_Hamiltonian(i)

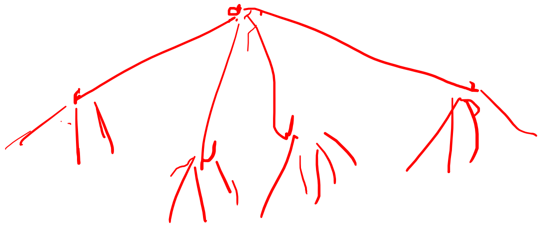
- flag = true
- for j = 1 to i – 1
-     if (v_i and v_j are neighbors)   //check vertices are distinct
-         flag = false
- if (v_i and v_{i-1} are neighbors)   // check last and first vertices
-         flag = true   // are neighbors
- else flag = false
- return(flag)

Algorithm

Hamiltonian(i)

- Input : Node i - the starting node
- if promising_Hamiltonian(i)
- if (i == n – 1) // all vertices are covered, print sol.
- print v[0] . . . . . . v[n – 1]
- else
- j = 2 //starts from node 2, as 1 is start node
- while (j < = n) { // for all vertices
- v[i] = j // assign vertex j
- Hamiltonian(i + 1)
- j = j + 1 }

Complexity of Hamiltonian cycle



- It is basically the number of nodes in the state space tree.
- $1 + (n-1) + (n-1)^2 + (n-1)^3 + ... + (n-1)^n$
- $= ((n-1)^{n+1} - 1) \;/\; (n - 2)$
- $= \mathbf{O(n^n)}$ EXPONENTIAL PROBLEM

Applications

1.TSP (another way to solve is by using Backtracking)

2. Mapping Genomes:

Applications involving genetic manipulation like finding genomic sequence is done using Hamiltonian paths.

Genomic sequence is made up of tiny fragments of genetic code called *reads*

and it is built by calculating the hamiltonian path in the network of these *reads*

where each *read* is considered a node and the overlap between two reads as edge.